Emulator Snap/File Formats

This file is a combination of the Z80 emulator documentation for filetypes and is copyright to to that, plus other information I have found. The following emulator file formats are described:

< KGB | SNA | Z80 (and SLT) | TAP | TZX | HOB | TRD | 000 | MDR | OUT | DAT >

First of all, here's where each is used and what it's used for:

- KGB:
- SNA: Memory snapshot created by (old) Jpp emulator - just a plain RAM snap with some header to store register status.
- Z80: Memory snapshot, compressed and with other hardware settings (eg 48/128, MF1, +D, AY chip). Created by Z80 emulator and a very common format. Several versions. Use Zip for better compression.
- TAP: Tape files. Stores the bytes as saved to tape with a short header. Can be concatenated easily. Emulated with patched calls to the ROM tape routines at 1366/1218. Z80 emulator created it, now supported by several emulators.
- TZX: New format that can store everything under the sun, under evolution and backwards compatible. Data stored in blocks with a header per block as well as general tzx information. Recent emulator versions are starting to support it, converters/tools exist.
- HOB: Hobeta format, from the Hobeta program on PCs used to transfer TRDOS files about. (TRDOS is a format used by the
- TRD:
- 000: Used by
- MDR:
- OUT:
- DAT:
- VOC: *(not described here)* A common format for Spectrum tape audio samples (like WAVs) of programs. The Z80 emulator can load from these instead of the LINE IN socket, for example.

KGB Snap format

```
Version 1.2-1.3
Contributed by Troels Norgaard

Notice, that in 680x0 the most significant byte goes first.
```

Offset	Size	Description	
0	49284		RAM dump 16252..65535
49284	132		unused, make 0
49416	10	dc.w	10,10,4,1,1 (different settings)
49426	1	dc.b	InterruptStatus (0=DI/1=EI)
49427	2	dc.b	0,3
49429	1	dc.b	ColorMode (0=BW/1=Color)
49430	4	dc.l	0
49434	16	dc.w	BC,BC',DE,DE',HL,HL',IX,IY
49450	2	dc.b	I,R
49452	2	dc.w	0
49454	8	dc.b	0,A',0,A,0,F',0,F
49462	8	dc.w	0,PC,0,SP
49470	2	dc.w	SoundMode (0=Simple/1=Pitch/2=RomOnly)
49472	2	dc.w	HaltMode (0=NoHalt/1=Halt)
49474	2	dc.w	IntMode (-1=IM 0/0=IM 1/1=IM 2)
49476	10		unused, make 0
Total: 49486 bytes			

Back to Top

Mirage Microdrive .SNA format

Used by Spectrum 1.7 and JPP. Notice that on Intel CPUs the least significant byte goes first, like on the Z80.

When the registers have been loaded, a RETN command is required to start the program. IFF2 is short for interrupt flip-flop 2, and for all practical purposes is the interrupt-enabled flag. Set means enabled.

Offset	Size	Description	
0	1	db	I
1	8	dw	HL',DE',BC',AF'
9	10	dw	HL,DE,BC,IY,IX
19	1	db	Interrupt (bit 2 contains IFF2, 1=EI/0=DI)
20	1	db	R
21	4	dw	AF,SP
25	1	db	IntMode (0=IM 0/1=IM 1/2=IM 2)
26	1	db	BorderColor (0..7, not used by Spectrum 1.7)
27	49152		RAM dump 16384..65535
Total: 49179 bytes			

Back to Top

Z80 Snap Format

Nettverksgruppa, 5/10-94, www@nvg.unit.no Specification of the .z80 snapshot format

This text was cribbed from the Z80 documentation and massaged slightly. See also the .sna specification.

The old .Z80 snapshot format (for version 1.45 and below) looks like this:

```
    Offset  Length  Description

    0       1       A register
    1       1       F register
    2       2       BC register pair (LSB, i.e.  C, first)
    4       2       HL register pair
    6       2       Program counter
    8       2       Stack pointer
    10      1       Interrupt register
    11      1       Refresh register (Bit 7 is not significant!)
    12      1       Bit 0  : Bit 7 of the R-register
                    Bit 1-3: Border colour
                    Bit 4  : 1=Basic SamRom switched in
                    Bit 5  : 1=Block of data is compressed
                    Bit 6-7: No meaning
    13      2       DE register pair
    15      2       BC' register pair
    17      2       DE' register pair
    19      2       HL' register pair
    21      1       A' register
    22      1       F' register
    23      2       IY register (Again LSB first)
    25      2       IX register
    27      1       Interrupt flipflop, 0=DI, otherwise EI
    28      1       IFF2 (not particularly important...)
    29      1       Bit 0-1: Interrupt mode (0, 1 or 2)
                    Bit 2  : 1=Issue 2 emulation
                    Bit 3  : 1=Double interrupt frequency
                    Bit 4-5: 1=High video synchronisation
                             3=Low video synchronisation
                             0,2=Normal
                    Bit 6-7: 0=Cursor/Protek/AGF joystick
                             1=Kempston joystick
                             2=Sinclair 2 Left joystick (or user
                               defined, for version 3 .Z80 files)
                             3=Sinclair 2 Right joystick
```

Because of compatibility, if byte 12 is 255, it has to be regarded as being 1. After this header block of 30 bytes the 48 K bytes of Spectrum memory follows in a compressed format (if bit 5 of byte 12 is one). The compression method is very simple: it replaces repetitions of at least five equal bytes by a four-byte code `ED ED xx yy`, which stands for "byte yy repeated xx times". Only sequences of length at least 5 are coded. The exception is sequences consisting of ED's; if they are encountered, even two ED's are encoded into `ED ED 02 ED`. Finally, every byte directly following a single ED is not taken into a block, for example ED 6*00 is not encoded into `ED ED ED 06 00` but into `ED 00 ED ED 05 00`. The block is terminated by an end marker, `00 ED ED 00`.

This format is used up to version 1.45. From version 2.0, a different format is used, since from then on, 128K snapshots had to be supported. The new format is used for all snapshots, either 48K or 128K. However, the emulator still understands the old format.

Also note that, although old .Z80 file formats are still understood, the emulator will never procude an old format .Z80 file, and also 48K snapshots are written in the new format. But, then again, new .Z80 file formats can be translated back to the old one (provided that it is an 48K snapshot of course) by using ConvZ80.

Version 2.01 and 3.0 .Z80 files start with the same 30 byte header as old .Z80 files used. Bit 4 and 5 of the flag byte have no meaning anymore, and the program counter (byte 6 and 7) are zero to signal a version 2.01 or version 3.0 snapshot file.

After the first 30 bytes, the additional header follows:

	Offset	Length	Description
*	30	2	Length of additional header block (see below)
*	32	2	Program counter
*	34	1	Hardware mode (see below)
*	35	1	If in SamRam mode, bitwise state of 74ls259. For example, bit 6=1 after an OUT 31,13 (=2*6+1). If in 128 mode, contains last OUT to 7ffd
*	36	1	Contains 0FF if Interface I rom paged
*	37	1	Bit 0: 1 if R register emulation on Bit 1: 1 if LDIR emulation on
*	38	1	Last OUT to fffd (soundchip register number)
*	39	16	Contents of the sound chip registers
	55	2	Low T state counter
	57	1	Hi T state counter
	58	1	Flag byte used by Spectator (QL spec. emulator) Ignored by Z80 when loading, zero when saving
	59	1	0FF if MGT Rom paged
	60	1	0FF if Multiface Rom paged. Should always be 0.
	61	1	0FF if 0-8191 is RAM
	62	1	0FF if 8192-16383 is RAM
	63	10	5x keyboard mappings for user defined joystick
	73	10	5x ascii word: keys corresponding to mappings above
	83	1	MGT type: 0=Disciple+Epson,1=Discipls+HP,16=Plus D
	84	1	Disciple inhibit button status: 0=out, 0ff=in
	85	1	Disciple inhibit flag: 0=rom pageable, 0ff=not

The value of the word at position 30 is 23 for version 2.01 files, and 54 for version 3.0 files. The starred fields are the ones that constitute the version 2.01 header, and their interpretation has remained unchanged except for byte 34:

Value:	Meaning in v2.01	Meaning in v3.0
0	48k	48k
1	48k + If.1	48k + If.1
2	SamRam	48k + M.G.T.
3	128k	SamRam
4	128k + If.1	128k
5	-	128k + If.1
6	-	128k + M.G.T.

The hi T state counter counts up modulo 4. Just after the ULA generates its once-in-every-20-ms interrupt, it is 3, and is increased by one every 5 emulated milliseconds. In these 1/200s intervals, the low T state counter counts down from 17472 to 0, which make a total of 69888 T states per frame.

The 5 ascii words (high byte always 0) at 73-82 are the keys corresponding to the joystick directions left, right, down (!), up (!), fire respectively. Shift, Symbol Shift, Enter and Space are denoted by [,],/,\ respectively. The ascii values are used only to display the joystick keys; the information in the 5 keyboard mapping words determine which key is actually pressed (and should correspond to the ascii values). The low byte is in the range 0-7 and determines the keyboard row. The high byte is a mask byte and determines the column. Enter for example is stored as 0x0106 (row 6 and column 1) and 'g' as 0x1001 (row 1 and column 4).

Byte 60 must be zero, because the contents of the Multiface RAM is not saved in the snapshot file. If the Multiface was paged when the snapshot was saved, the emulated program will most probably crash when loaded back

Bytes 61 and 62 are a function of the other flags, such as byte 34, 59, 60 and 83.

Hereafter a number of memory blocks follow, each containing the compressed data of a 16 K block. The compression is according to the old scheme, except for the end-marker, which is now absent. The structure of a memory block is:

Byte	Length	Description
0	2	Length of data (without this 3-byte header)
2	1	Page number of block
3	[0]	Compressed data

The pages are numbered, depending on the hardware mode, in the following way:

Page	In '48 mode	In '128 mode	In SamRam mode
0	48K rom	rom (basic)	48K rom
1	Interface I, Disciple or Plus D	rom (reset)	rom, according to setting
2	-	rom (reset)	samram rom (basic)
3	-	page 0	samram rom (monitor,..)
4	8000-bfff	page 1	Normal 8000-bfff
5	c000-ffff	page 2	Normal c000-ffff
6	-	page 3	Shadow 8000-bfff
7	-	page 4	Shadow c000-ffff
8	4000-7fff	page 5	4000-7fff
9	-	page 6	-
10	-	page 7	-
11	Multiface rom	Multiface rom	-

In 48K mode, pages 4,5 and 8 are saved. In SamRam mode, pages 4 to 8 are saved. In '128 mode, all pages from 3 to 10 are saved. This version saves the pages in numerical order. There is no end marker.

SLT extension to Z80 format

.SLT files (which stands for 'super level loader trap files') are like .Z80 files except that after the ordinary data another section follows, containing things like level data (previously stored in .DAT files, and giving .SLT files their name) or loading screens. The ".Z80 "-file preceding the SLT must conform the format of v2.01 or v3.0x files (long header).

Though it is agreed that .Z80 files will not contain SLT data, and files with SLT data will always have extension .SLT, Z80 does in fact not distinguish between the extensions.

The SLT format was cooked up by Damien Burke, James McKay and yours truly. It starts as an ordinary >= v2.01 .Z80 file. Directly following this comes a SLT identifier:

```
Offset  Length  Description

0       6       Separator (0,0,0,'S','L','T')
```

Then a table follows, each entry describing a piece of data. The format of a single table entry is:

```
Offset  Length  Description

0       2       Data type: 1=level data, 3=loading screen
2       2       Id word: Level number (between 0 and 255 inclusive)
                 for type 1, border colour (between 0 and 7) for
                 type 3.
4       4       Length of data block in bytes (Note: long word)
```

Data types other than type 1 and 3 are not supported by Z80 v3.04, and are ignored. The table ends with an all-zero end marker:

```
Offset  Length  Description

0       8       End marker (all zeroes)
```

Finally, the data blocks follow. The blocks are stored in the order in which they appear in the table, i.e. the offset of a particular data block is the sum of the lengths of the blocks that precede it. The internal format of these blocks depend on the data type:

```
Type:   Format:

0       (no data)
1       Compressed data (ED ED xx yy scheme, see above) expanding
        to anything up to 48K
3       Compressed data expanding to exactly 6912 bytes of data
```

See the end of section 5.9 in the Z80 emulator "techinfo.doc" file for an explanation of how to load these blocks using the ED FB opcode, and error handling.

All words in the format have the least significant byte first.

Nettverksgruppa, 19/1-95, www@nvg.unit.no

Back to Top

TAP tape file format

The .TAP files contain blocks of tape-saved data. All blocks start with two bytes specifying how many bytes will follow (not counting the two length bytes). Then raw tape data follows, including the flag and checksum bytes. The checksum is the bitwise XOR of all bytes including the flag byte. For example, when you execute the line SAVE "ROM" CODE 0,2 this will result:

```
   |------ Spectrum-generated data -------|       |---------|

 13 00 00 03 52 4f 4d 7x20 02 00 00 00 00 80 f1 04 00 ff f3 af a3

 ^^^^^..... first block is 19 bytes (17 bytes+flag+checksum)
       ^^... flag byte (A reg, 00 for headers, ff for data blocks)
          ^^ first byte of header, indicating a code block

 file name ..^^^^^^^^^^^^^
 header info ..............^^^^^^^^^^^^^^^^^
 checksum of header ........................^^
 length of second block ......................^^^^^
 flag byte ..........................................^^
 first two bytes of rom ................................^^^^^
 checksum (checkbittoggle would be a better name!).............^^
```

The emulator will always start reading bytes at the beginning of a block. If less bytes are loaded than are available, the other bytes are skipped, and the last byte loaded is used as checksum. If more bytes are asked for than exist in the block, the loading routine will terminate with the usual tape-loading-error flags set, leaving the error handling to the calling Z80 program.

Note that it is possible to join .TAP files by simply stringing them together, for example `COPY /B FILE1.TAP + FILE2.TAP ALL.TAP`

For completeness, I'll include the structure of a tape header. A header always consists of 17 bytes:

Byte	Length	Description
0	1	Type (0,1,2 or 3)
1	10	Filename (padded with blanks)
11	2	
Length of data block		
13	2	Parameter 1
15	2	Parameter 2

The type is 0,1,2 or 3 for a Program, Number array, Character array or Code file. A screen$ file is regarded as a Code file with start address 16384 and length 6912 decimal. If the file is a Program file, parameter 1 holds the autostart line number (or a number >=32768 if no LINE parameter was given) and parameter 2 holds the start of the variable area relative to the start of the program. If it's a Code file, parameter 1 holds the start of the code block when saved, and parameter 2 holds 32768. For data files finally, the byte at position 14 decimal holds the variable name.

Back to Top

TZX Format

Tzx is one of the newer formats around and underwent a great deal of discussion in the comp.sys.sinclair newsgroup before it was finalised. It is an extensible format (I think) that can cater for just about any information about a Spectrum program, the producer/author information, hardware required for it, and various other settings.

Rather than include full details here and duplicate/get out of date, I will just point you to the TZX information at the World of Spectrum archive (documents section).

Back to Top

MDR Microdrive format

The emulator uses a cartridge file format identical to the 'Microdrive File' format of Carlo Delhez' Spectrum emulator Spectator for the QL, who devised the format. This format is now also supported by XZX of Dean Harriot. The following information is adapted from Carlo's documentation. It can also be found in the 'Spectrum Microdrive Book', by Ian Logan (co-writer of the excellent 'Complete Spectrum ROM Disassembly').

A cartridge file contains 254 'sectors' of 543 bytes each, and a final byte flag which is non-zero is the cartridge is write protected, so the total length is 137923 bytes. On the cartridge tape, after a GAP of some time the Interface I writes 10 zeros and 2 FF bytes (the preamble), and then a fifteen byte header-block-with-checksum. After another GAP, it writes a preamble again, with a 15- byte record-descriptor-with-checksum (which has a structure very much like the header block), immediately followed by the data block of 512 bytes, and a final checksum of those 512 bytes. The preamble is used by the Interface I hardware to synchronise, and is not explicitly used by the software. The preamble is not saved to the microdrive file:

```
offset length name    contents

  0      1   HDFLAG   Value 1, to indicate header block
  1      1   HDNUMB   sector number (values 254 down to 1)
  2      2            not used
  4     10   HDNAME   microdrive cartridge name (blank padded)
 14      1   HDCHK    header checksum (of first 14 bytes)

 15      1   RECFLG   - bit 0: always 0 to indicate record block
                      - bit 1: set for the EOF block
                      - bit 2: reset for a PRINT file
                      - bits 3-7: not used (value 0)
 16      1   RECNUM   data block sequence number (value starts at 0)
 17      2   RECLEN   data block length (<=512, LSB first)
 19     10   RECNAM   filename (blank padded)
 29      1   DESCHK   record descriptor checksum (of previous 14 bytes)
 30    512            data block
542      1   DCHK     data block checksum (of all 512 bytes of data
                       block, even when not all bytes are used)
```

Repeated 254 times

(Actually, this information is 'transparent' to the emulator. All it does is store 2 times 254 blocks in the .MDR file as it is OUTed, alternatingly of length 15 and 528 bytes. The emulator does check checksums, see below; the other fields are dealt with by the emulated Interface I software.)

A used record block is either an EOF block (bit 1 of RECFLG is 1) or contains 512 bytes of data (RECLEN=512, i.e. bit 1 of MSB is 1). An empty record block has a zero in bit 1 of RECFLG and also RECLEN=0. An unusable block (as determined by the FORMAT command) is an EOF block with RECLEN=0.

The three checksums are calculated by adding all the bytes together modulo 255; this will never produce a checksum of 255. Possibly, this is the value that is read by the Interface I if there's no or bad data on the tape.

In normal operation, all first-fifteen-byte blocks of each header or record block will have the right checksum. If the checksum is not right, the block will be treated as a GAP. For instance, if you type OUT 239,0 on a normal Spectrum with interface I, the microdrive motor starts running and the cartridge will be erased completely in 7 seconds. CAT 1 will respond with 'microdrive not ready'. Try it on the emulator...

Back to Top

OUT files

These files are produced when logging OUTs; see menu option O in the Extra Functions menu. For the specified I/O ports, all OUTs to these ports are recorded in the .OUT file, together with the exact time at which the OUTs were executed.

An .OUT file consists of a string of 5-byte blocks. The first word is the timing word; it has a value between 0 and 17471 inclusive (or between 0 and 17726 inclusive when a 128K Spectrum is emulated), and a unit value corresponds to 1 T state (=1/3494400 s). After this, the OUT port that was written to follows, then the value OUTed itself:

```
Offset  Length  Description

0       2       Time (0-17471 or 0-17726)
2       2       Port address
4       1       Value
```

Every 1/200th of an emulated second, that is, every 69888/4=17472 T states (or 70908/4=17727 T states on 128K Spectrums), a time-wraparound block is written to the .OUT file:

```
Offset  Length  Description

0       2       Flag word #FFFF indicating wraparound-block
2       2       Length of preceding block (17472 or 17727 T)
4       1       Not used
```

So even when the Spectrum does not OUT to the logged ports at all, 1000 bytes get written to the log file every second.

By default, an OUT to an even I/O address which does not change the state of the MIC and EAR outputs is not written to the .OUT file, to save disk space when recording music. If you want all OUTs, specify -xg on the command line.

The .OUT files are also used to make a simple trace of a running Spectrum program. Specify -xy on the command line; as soon as you activate OUT logging, a trace is dumped also. For each instruction encountered during emulation, the following block is written to the .OUT file:

```
Offset  Length  Description

0       2       Flag word (#FFFE)
2       2       Program counter
5       1       A register
```

Furthermore, no time-wraparound blocks are written to the .OUT file when tracing. HALT instructions (118 decimal) are special in that they do not generate a block in the log file; this is to make comparisons between different logs of the same program easier. Admittedly, this is a very crude way of tracing a program, but it's better than nothing, and very useful very occasionally.

Back to Top

DAT game data files

These files are used to store level data; blocks of memory that are loaded when the opcode ED FB is executed. The block is loaded at the address pointed to by the HL register. The level number is in the A register. If the emulator fails to load a level data block, it complements the carry flag. Versions of the emulator prior to v3.04 did not do this.

When an ED FB opcode is encountered, the emulator first looks for the snapshot file last loaded, and checks whether it contains the level requested (see the description of the .Z80 format). If this fails, it looks for a .DAT file of the level; the name of this file is derived from the name of the last snapshot loaded by appending the (decimal) value of the A register to the name of the snapshot file, dropping characters from the original snapshot name to make the total length 8 characters if necessary. .DAT files simply contain the plain level data (in contrast to the level data in .Z80 files, which are compressed).

Emulators (though World of Spectrum has a better page)
Back to Speccy Techy
Speccy Intro

File last updated: 25 August 1998
Author: John Garner / john@breezer.demon.co.uk