VICE, the Versatile Commodore Emulator

1 GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:
   a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
   b. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
   c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

   The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

   If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.
5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.
7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

   This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.
8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.
9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.
10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software

which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
one line to give the program's name and an idea of what it does.
Copyright (C) 19yy  name of author

This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) 19yy name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details
type 'show w'.  This is free software, and you are welcome
to redistribute it under certain conditions; type 'show c'
for details.
```

The hypothetical commands '`show w`' and '`show c`' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than '`show w`' and '`show c`'; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright
interest in the program 'Gnomovision'
(which makes passes at compilers) written
by James Hacker.

signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

2 About VICE

VICE is the one and only *Versatile Commodore Emulator*. It provides emulation of the Commodore C64, C64DTV, C128, VIC20, PET, PLUS4 and CBM-II computers within a single package. The emulators run as separate programs, but have the same user interface, share the same settings and support the same file formats.

Important notice: If you have no idea what a Commodore 8-bit computer is, or have questions about how these machines are used, how the file formats work or anything else that is not strictly related to VICE, you should read the appropriate FAQs *first*, as that kind of information is not available here. See Chapter 18 [Contacts], page 176. for information about how to retrieve the FAQs.

All the emulators provide an accurate 6502/6510 emulator, with emulation of all the opcodes (both documented and undocumented ones) and accurate timing. Unlike other emulators, VICE aims to be cycle accurate; it tries to emulate chip timings as precisely as possible and does so *efficiently*.

Please do *not* expect the C64DTV, C128, PET, PLUS4 and CBM-II emulators to be as good as the C64 or VIC20 one, as they are still under construction.

Notice: This documentation is written for the Unix release of VICE.

2.1 C64 emulator features

As of version 2.3, two C64 emulators are provided: '`x64`' (fast) and '`x64sc`' (accurate).

The fast C64 emulator, called '`x64`', features a fairly complete emulation of the VIC-II video chip: sprites, all registers and all video modes are fully emulated. The emulation has been fully cycle-accurate since version 0.13.0.

The accurate C64 emulator, called '`x64sc`', features a cycle-based and pixel-accurate VIC-II emulation. This requires a much faster machine than the old '`x64`'.

A rather complete emulation of the SID sound chip is also provided. All the basic features are implemented as well as most of the complex ones including synchronisation, ring modulation and filters. There are three emulators of the SID chip available: first is the "standard" VICE emulator, available since VICE 0.12; the second is Dag Lem's reSID engine and the third one is reSID-fp. The reSID engines are a lot more accurate than the standard engine, but they are also a lot slower, and only suitable for faster machines.

Naturally, also both CIAs (or VIAs, in some cases) are fully emulated and cycle accurate.

2.2 C64DTV emulator features

The C64DTV emulator, called '`x64dtv`', features emulation of C64DTV revisions 2 and 3. The emulator is under construction, but most of the DTV specific features are already supported (with varying accuracy).

Video cache is disabled by default as it currently doesn't work with some of C64DTV's new video modes. The new video modes have a simple "fake" video cache implementation that may give incorrect results and decreased performance.

2.3 C128 emulator features

The C128 emulator, called '`x128`', features a complete emulation of the internal MMU (*Memory Management Unit*), 80 column VDC screen, fast IEC bus emulation, 2 MHz mode, Z80 emulation plus all the features of the C64 emulation.

2.4 VIC20 emulator features

The VIC20 emulates all the internal hardware, including the VIA chips. The VIC-I video chip is fully emulated except NTSC interlace mode, so most graphical effects will work correctly.

Sound support is implemented, but is still at an experimental stage. If you think it could be improved and know how to do so, feel free to contact us (see Chapter 18 [Contacts], page 176).

The VIC20 emulator now allows the use of the VIC1112 IEEE488 interface. You have to enable the hardware (by menu, resource, or commandline option) and then load the IEEE488 ROM (see for example `http://www.funet.fi/pub/cbm/schematics/cartridges/vic20/ieee-488/325329-04.bin`, but you have to double the size to 4k for now). The IEEE-488 code is then started by `SYS45065`.

2.5 PET emulator features

The PET emulator emulates the 2001, 3032, 4032, 8032, 8096, 8296 and SuperPET (MicroMainFrame 9000) models, covering practically the whole series. The hardware is pretty much the same in each and that is why one single program is enough to emulate all of them. For more detailed information about PET hardware please refer to the '`PETdoc`' file.

Both the 40 column and 80 column CRTC video chips are emulated (from the 4032 onward), but a few of the features are not implemented yet (numbers of rasterlines per char and lines per screen). Fortunately, they are not very important for average applications.

Sound is available for the PET as well, but like the VIC20's it is still under construction.

The PET 8096 is basically a PET 8032 with a 64k extension board which allows remapping the upper 32k with RAM. You have to write to a special register at `$fff0` to remap the memory. The PET 8296 is a 8096 but with a completely redesigned motherboard with 128k RAM in total. Of the additional 32k RAM you can use only some in blocks of 4k, but you have to set jumpers on the motherboard for it. VICE uses the command line options '`-petram9`' and '`-petramA`' instead. Also, the video controller can handle a larger address range. The PET 8x96 model emulations run the Commodore LOS-96 operating system - basically an improved BASIC 4 version with up to 32k for BASIC text and 32k for variables. See '`PETdoc`' for more information.

The SuperPET also is a PET 8032 with an expansion board. It can map 4k at a time out of 64k into the `$9***` area. Also it has an ACIA 6551 for RS232 communication. The 6809 that is built into the SuperPET is not emulated, though.

The PET computers came with three major ROM revisions, so-called BASIC 1, 2 and 4, all of which are provided. The PET 2001 uses the version 1, the PET 3032 uses version 2, and the others use version 4. The 2001 ROM is horribly broken with respect to IEEE488

(they shipped it before they tested it with the floppy drive, so only tape worked. Therefore the emulator patches the ROM to fix the IEEE488 routines.

As well as other low-level fixes the 2001 patch obtains the load address for a program file from the first two bytes of the file. This allows the loading of both PET2001-saved files (that have $0400 as their load address) and other PET files (that have $0401). The PET2001 saves from $0400 and not from $0401 as other PETs do.

Moreover, the secondary addresses used are now `0` and `1` for load and save, respectively, and not arbitrary unused secondary addresses.

To select which model to run, specify it on the command line with the `-model MODEL` option, where `MODEL` can be one of a list of PET model numbers, all described in see Section 7.6.1 [PET model], page 80

2.6 CBM-II emulator features

The CBM-II emulator emulates several types of CBM-II models. Those models are known under different names in the USA and Europe. In the States they have been sold as `B128` and `B256`, in Europe as `CBM 610`, `CBM 620` (low-profile case) or `CBM 710` and `CBM 720` (high-profile case with monitor). In addition to that now an experimental C510 emulation is included. The C510 (also known as P500) is the little brother of the C600/700 machines. It runs at roughly 1 MHz and, surprise, it has a VIC-II instead of the CRTC. Otherwise the different line of computers are very similar.

These computers are prepared to take a coprocessor board with an 8088 or Z80 CPU. Indeed there are models `CBM 630` and `CBM 730` that supposedly had those processors. However these models are not emulated.

The basic difference is the amount of RAM these machines have been supplied with. The `B128` and the `CBM *10` models had 128k RAM, the others 256k. This implies some banking scheme, as the 6502 can only address 64k. And indeed those machines use a 6509, that can address 1 MByte of RAM. It has 2 registers at addresses 0 and 1. The indirect bank register at address 1 determines the bank (0-15) where the opcodes `LDA (zp),Y` and `STA (zp),Y` take the data from. The exec bank register at address 0 determines the bank where all other read and write addresses take place.

The business line machines (C6xx/7xx) have the RAM in banks 1-2, resp. 1-4. All available banks are used for BASIC, where program code is separated from all variables, resp. from normal variables, strings and arrays that are distributed over other banks. The C510 instead has RAM in banks 0 and 1, and uses bank 1 for program and all variables. Bank 0, though, can be accessed by the VIC-II to display graphics.

Many models have been expanded to more than the built-in memory. In fact some machines have been expanded to the full 1M. Bank 15 is used as system bank, with only little RAM, and lots of expansion cartridge ROM area, the I/O and the kernal/basic ROMs. Some models have been modified to map RAM into the expansion ROM area. Those modifications can be emulated as well.

The different settings are described in see Section 7.7.1 [CBM-II model], page 85.

2.7 The keyboard emulation

There are two ways of emulating the keyboard in VICE.

The default way (*symbolic mapping*) is to map every key combination to the corresponding key combination on the real machine: for example, if you press ⟨*⟩, which is bound to *Shift-8* on a U.S. keyboard, in the C64 emulator, the emulated machine will have just the *unshifted* ⟨*⟩ key pressed (as ⟨*⟩ is unshifted on the C64 keyboard). Likewise, pressing ⟨'⟩ on the same U.S. keyboard without any shift key will cause the combination *Shift-7* to be pressed in the emulated C64. This way, it becomes quite obvious what keys should be typed to obtain all the symbols.

There is, however, one problem with symbolic mapping: some keys really need to be mapped specially regardless. The most important examples being, in the VIC20, C64 and C128 emulators, that ⟨CTRL⟩ is mapped to ⟨Tab⟩ and that the ⟨Commodore⟩ key is mapped to the left ⟨Control⟩). The ⟨RUN/STOP⟩ key is mapped to the ⟨ESC⟩ key on the PC keyboard. The PET emulator, lacking the ⟨Commodore⟩ key but having an ⟨ESC⟩ key, uses the left ⟨Control⟩ key as ⟨RUN/STOP⟩ and the ⟨ESC⟩ key as ⟨ESC⟩ of course.

The second way (*positional mapping*) is to map every key on the "real" keyboard to the key which has the same position on the keyboard of the emulated machine. This way, no ⟨Shift⟩ key is forced by the program (with the exception of the function keys ⟨F2⟩, ⟨F4⟩, ⟨F6⟩ and ⟨F8⟩, which require ⟨Shift⟩ on the Commodore keyboards), and the keyboard is more comfortable to use in those programs (such as some games) that require the keys to be in the correct positions.

Warning: unlike the real C64, VICE "presses" the ⟨Shift⟩ key *together* with the key to shift when the ⟨Shift⟩ must be forced. In most cases this should work fine, but some keyboard routines are quite picky and tend not to recognize the shift key because of this. For instance, *F6* (which on the real C64 is obtained with *Shift + F5*) could be recognized as *F5*. In that case, use the shift key manually (i.e., type *Shift + F5* in the example). Yes, we know this is a bug.

The *RESTORE* key is mapped to *Page Up* (or *Prev*) by default.

2.8 The joystick emulation

Joysticks can be emulated both via the keyboard and via a real joystick connected to the host machine (the latter only works on GNU/Linux systems).

There are two keyboard layouts for joystick use, known as *numpad* and *custom*.

The *numpad* layout uses the numeric keypad keys, i.e., the numbers ⟨1⟩...⟨9⟩ which emulate all the directions including the diagonal ones; ⟨0⟩ emulates the fire button.

The *custom* layout uses the keys ⟨w⟩, ⟨e⟩, ⟨r⟩, ⟨s⟩, ⟨d⟩, ⟨f⟩, ⟨x⟩, ⟨c⟩, ⟨v⟩ for the directions and ⟨space⟩ for the fire button instead.

2.9 The disk drive emulation

All the emulators support up to 4 external disk drives as devices 8, 9, 10 and 11. Each of these devices can emulate virtual Commodore 1541, 1541-II, 1571, 1581, 2031, 2040, 3040, 4040, 1001, 8050 and 8250 drives in one of four ways:

- using disk images, i.e., files that contain a dump of all the blocks contained in a real floppy disk (if you want more information about what a disk image is, consult the `comp.emulators.cbm` FAQ);

- accessing file system directories, thus giving you the use of files without having to copy them to disk images; this also allows you to read and write files in the `P00` format (again, consult the `comp.emulators.cbm` FAQ for more info).
- accessing a real device connected to the host machine. As of VICE 1.11 it is possible to connect real drives like Commodore 1541 to the printer port of the host using the XA1541 or XM1541 cable. Currently this only works on Linux or Windows using the OpenCBM library. You can get it from `http://www.lb.shuttle.de/puffin/cbm4linux` (cbm4linux, Linux version) or from `http://cbm4win.sf.net/` (cbm4win, Windows version).
- directly using the disk drive of the host. The 3.5" disk drive of the host can be used to read or write Commodore 1581 formatted disks. Currently this raw drive access feature is only available for Linux hosts.

When using disk images there are two available types of drive emulation. One of them the *virtual drive* emulation. It does *not* really emulate the serial line, but patches the kernal ROM (with the so-called *kernal traps*) so that serial line operations can be emulated via C language routines. This emulation is very fast, but only allows use of standard DOS functions (and not even all of them). For real device or raw drive access it is required to enable this type of emulation.

The IEEE488 drives (2031, 2040, 3040, 4040, 1001, 8050 and 8250) do not use kernal traps. Instead the IEEE488 interface lines are monitored and the data is passed to the drive emulation. To use them on the C64, you need to enable the IEEE488 interface emulation. Only if the IEEE488 emulation is enabled, those drives can be selected.

The other alternative is a *true drive* emulation. The Commodore disk drives are provided with their own CPU (a 6502 as the VIC20 and the PETs) and their own RAM and ROM. So, in order to more closely emulate its features, a complete emulation of this hardware must be provided and that is what the *hardware level* emulation does. When the *hardware level* emulation is used, the kernal routines are remain unpatched and the serial line is fully emulated. The problem with this emulation is that it needs a lot of processing power, mainly because the emulator has to emulate two CPUs instead of one.

The PETs do not use a serial IEC bus to communicate with the floppy drive but instead use the parallel IEEE488 bus. This does *byte by byte* transfers, as opposed to the *bit by bit* transfers of the C64 and VIC20, so making it feasible to emulate the parallel line completely while emulating the drive at DOS level only. The IEEE488 line interpreter maps the drives 8-11 (as described above) to the IEEE488 disk units, and no kernal traps are needed. The same emulation of the Commodore IEEE488 bus interface is available for the C64 and the VIC20. With IEEE488 drives you can have true 2031 emulation at unit #8, and still have filesystem access at units #10 or #11, because monitoring the IEEE488 lines does not interfere with the true drive emulation.

The IEEE488 disk drives 3040, 4040, 8050 and 8250 are Dual Drive Floppy Disks. This means that these drives handle two disks. To Accomplish the emulation, only one disk can be emulated, namely unit #8. The attached image, track display and LED display of unit #9 are used for the second drive of the dual disk drives. On unix the unit number display (8 or 9) in the emulation window changes to the drive number display (0 or 1).

The Commodore 3040, 4040, 1001, 8050 and 8250 disk drives are so-called "old-style" disk drives. Their architecture includes not one, but two processors of the 6502 type,

namely a 6502 for the file handling and communication with the PET (IP), and a 6504 (which is a 6502 with reduced address space) for the drive handling (FDC). Both processors communicate over a shared memory area. The IP writes commands to read/write blocks to this area and the FDC executes them. To make the emulation feasible, the FDC processor is not emulated cycle-exactly as a 6504, but simply by checking the commands and executing them on the host. This provides a fast FDC emulation, but disallows the sending the FDC processor commands to execute code. Applications where this is necessary are believed to be rather seldom. Only the format command uses this feature, but this is checked for.

The dual disk drive 2040 emulates one of the very first CBM disk drives. This drive has DOS version 1. DOS1 uses an own disk type, that is closely related to the 1541 disk image. Only on tracks 18-24 DOS1 disks have a sector more than 1541 disks. DOS1 disk images have the extension .d67.

The dual disk drives 3040 and 4040 use the same logical disk format as the VC1541 and the 2031. In fact, the 4040 was the first disk with DOS version 2. The 3040 emulated here originally was the same as 2040, only for the european 30xx PET series. As many of the original DOS1 disk drives were upgraded (a simple ROM upgrade!) to DOS2, I use the 3040 number for a DOS 2.0 disk drive, and 4040 for a revised DOS 2 disk drive. It is, however, not yet clear whether the disks here are write compatible to the 1541, as rumors exist that the write gap between sectors is different. But read compatible they are. As VICE emulates the FDC processor in C and not as 6504 emulation, this does not matter in VICE.

The drives 1001, 8050 and 8250 do actually have the very same DOS ROM. Only the code in the FDC is different, which is taken care of by VICE. So for all three of those disk drives, only `dos1001` is needed. The DOS version used is 2.7.

2.10 Supported file formats

VICE supports the most popular Commodore file formats:

- `X64` (preferred) or `D64` disk image files; Used by the 1541, 2031, 3040, 4040 drives.
- `G64` GCR-encoded 1541 disk image files;
- `D67` CBM2040 (DOS1) disk image format
- `D71` VC1571 disk image format
- `D81` VC1581 disk image format
- `D80` CBM8050 disk image format
- `D82` CBM8250/1001 disk image format
- `T64` tape image files (read-only);
- `P00` program files;

An utility (`c1541`, see Chapter 12 [c1541], page 112) is provided to allow transfers and conversions between these formats.

Notice that the use of the `X64` file format is depreciated now.

You can convert an `X64` file back into a `D64` file with the UNIX `dd` command:

```
dd bs=64 skip=1 if=IMAGE.X64 of=IMAGE.D64
```

See Chapter 15 [File formats], page 120. for a technical description of the supported file formats.

2.11 Common problems

This section tries to describe the most common known problems with VICE, and how to resolve them.

2.11.1 Sound problems

VICE should compile and run without major problems on many UNIX systems, but there are some known issues related to the sound driver. In fact, the sound code is the least portable part of the emulator and has not yet been thoroughly tested on all the supported platforms.

Linux, AIX and SGI systems should play sound without any problems; if you are running Linux please use a 2.x kernel, as VICE needs some features that were not implemented in older versions of the Linux sound driver.

On the other hand, HP-UX and Solaris machines are known to cause troubles. If you think you can help debugging the code for these systems, your help would be really appreciated. We are having troubles finding HP-UX and SUN consoles to work at...

Some problems have been reported with the proprietary version of the Open Sound System for Linux. With a Crystal sound card, sound output was significantly delayed and, apparently, the allocated buffer size was completely wrong. This is not a VICE bug, but rather an OSS bug.

2.11.2 Shared memory problems

If you cannot start VICE because you get errors about shared memory, try to run it with the '`+mitshm`' command-line option (see Section 6.4.2 [Video options], page 28). This will completely disable usage of the MITSHM extensions, that are normally used to speed up the emulation window updates. Of course, this will also result in a big loss in speed.

Reasons for this failure could be:

- IPC support has been disabled at the system level; some system administrators disable this for security reasons. If *you* are the system administrator, use a kernel that has IPC support compiled in and enabled.
- You are attempting to run the emulator across the network (i.e., the emulator runs on one machine, and the output is displayed on another machine that works as an X terminal) and for some reason VICE does not recognize this fact. In this case, you have found a bug, so please report it to us.

If you want to avoid running the emulator with '`+mitshm`' every time, run it once with '`+mitshm`' and then choose "Save settings" from the right-button menu.

2.11.3 Printer problems

VICE supports the emulation of a printer either on the userport or as IEC device 4. Unfortunately the Commodore IEC routines do not send all commands to the IEC bus. For example an `OPEN 1,4` is not seen on the IEC bus. Also a `CLOSE 1` after that is not seen. VICE can see from printing that there was an `OPEN`, but it cannot see when the close was. Also a "finish print job" cannot be seen on the userport device. To flush the printer buffer (write to `print.dump` or to the printer) now a menu entry can be used. Disabling and re-enabling the printer should work as well.

The printing services have not been extensively tested but apart from the problem mentioned above it should work fine now.

2.11.4 PET keyboard problems

If you find that the German keyboard mapping (plus German charset) does not print uppercase umlauts, then you are right. The umlauts replace the [,\ and] characters in the charset. The keys that make these characters do not have a different entry in the PET editor ROM tables when shifted. Thus it is not possible to get the uppercase umlauts in the editor. Nevertheless other programs are reported to change the keyboard mapping table and thus allow the use of the shifted (uppercase) umlauts.

Anyway, the VICE keyboard mappings are far from being perfect and we are open to any suggestions.

3 Invoking the emulators

The names of the available emulators are:

- `x64`, the fast C64 emulator
- `x64sc`, the accurate C64 emulator
- `x64dtv`, the C64DTV emulator
- `x128`, the C128 emulator
- `xvic`, the VIC20 emulator
- `xpet`, the PET emulator
- `xplus4`, the PLUS4 emulator
- `xcbm2`, the CBM-II emulator

You can run each of them by simply typing the name from a shell. If you want to run them from another application (e.g., a window manager or some other sort of program launcher) you should always run them from a terminal window such as `xterm` or `rxvt` since VICE provides a lot of debugging information that is sent to the terminal and has built-in monitor that also appears there. For example, you could do

```
xterm -e x64
```

3.1 Command-line options used during initialization

There are several options you can specify on the command line. Some of them are used to specify emulation settings and will be described in detail later (see Chapter 6 [Settings and resources], page 25 for a complete list). The remaining options are used only to give usage information or to initialize the emulator in some way:

`-help`
`-?` List all the available command-line options and their meaning.

`-default` Set default resources (see Chapter 6 [Settings and resources], page 25). This will override all the settings specified before, but not the settings specified afterwards on the command line.

`-config <filename>`
: Specify config file

`-logfile <name>`
: Specify log file name

`-verbose` Enable verbose log output.

`-keybuf <string>`
: Put the specified string into the keyboard buffer.

`-console` Console mode (for music playback)

`-chdir <directory>`
: Change the working directory.

`-autostart IMAGE`
: Autostart '`IMAGE`' (see Section 3.2 [Command-line autostart], page 16).

`-autoload <name>`
: Attach and autoload tape/disk image <name>

`-basicload`
: On autostart, load to BASIC start (without ',1')

`+basicload`
: On autostart, load with ',1'

`-autostartwithcolon`
: On autostart, use the 'RUN' command with a colon, i.e., 'RUN:'

`+autostartwithcolon`
: On autostart, do not use the 'RUN' command with a colon; i.e., 'RUN'

`-autostart-handle-tde`
`+autostart-handle-tde`
: Handle/Do not handle True Drive Emulation on autostart

`+autostart-warp`
: Enable/Disable warp mode during autostart

`-autostartprgmode`
: Set autostart mode for PRG files

`-autostartprgdiskimage`
: Set disk image for autostart of PRG files

`-1 NAME` Attach '`NAME`' as a tape image file.

`-8 NAME`
`-9 NAME`
`-10 NAME`
`-11 NAME` Attach '`NAME`' as a disk image to device 8, 9, 10 or 11.

`-attach8ro`
`-attach9ro`
`-attach10ro`
`-attach11ro`
: Attach disk image for drive #8-11 read only

`-attach8rw`
`-attach9rw`
`-attach10rw`
`-attach11rw`
Attach disk image for drive #8-11 read write (if possible)

3.2 Autostarting programs from the command-line

It is possible to let the emulator *autostart* a disk or tape image file, by simply specifying its name as the *last* argument on the command line, for example

```
x64 lovelygame.x64.gz
```

will start the C64 emulator, attaching '`lovelygame.x64.gz`' as a disk image and running the first program on it. You can also specify the name of the program on the fisk image by appending a colon ('`:`') the name itself to the argument; for example

```
x64 "lovelygame.x64.gz:run me"
```

will run the program named '`run me`' on '`lovelygame.x64.gz`' instead of the first one.

Using the command-line option `-autostart` is equivalent; so the same result can be obtained with

```
x64 -autostart "lovelygame.x64.gz:run me"
```

If you specify a raw CBM or P00 file, the emulator will setup the file system based drive emulation so that it is enabled and accesses the directory containing the file first. This is a very convenient way to start multi-file programs stored in file system directories and not requiring "true" drive emulation.

See Section 5.5 [Disk and tape images], page 23. for more information about images and autostart.

4 System files

In order to work properly, the emulators need to load a few system files:

- the *system ROMs*, raw binary files containing copies of the original ROMs of the machine you are emulating;
- the *keyboard maps*, text files describing the keyboard layout;
- the *palette files*, text files describing the colors of the machine you are emulating.
- the *romset files*, text files describing the different ROMs to load.

The place where they will be searched for depends on the value of the `Directory` resource, which is a colon (`:`)-separated search path list, like the UNIX `PATH` environment variable. The default value is

```
PREFIX/lib/vice/EMU:$HOME/.vice/EMU:BOOTPATH/EMU
```

Where `PREFIX` is the installation prefix (usually '`/usr/local`'), `EMU` is the name of the emulated machine (`C64`, `C128`, `PET`, `CBM-II` or `VIC20`) and `BOOTPATH` is the directory where the executable resides. The disk drive ROMs are looked for in a directory with `EMU` set to `DRIVES`. `$HOME` is the user's home directory.

For example, if you have the C64 emulator installed in

```
/usr/local/bin/x64
```

then the value will be

```
/usr/local/lib/vice/C64:$HOME/.vice/C64:/usr/local/bin/C64
```

And system files will be searched for under the following directories, in the specified order:

1. `/usr/local/lib/VICE/C64`
2. `$HOME/.vice/C64`
3. `/usr/local/bin/C64`

System files can still be installed in a different directory if you specify a complete path instead of just a file name. For example, if you specify '`./kernal`' as the kernal image name, the kernal image will be loaded from the current directory. This can be done by using command-line options or by modifying resource values (see Section 6.1 [Resource files], page 26).

4.1 ROM files

Every emulator requires its own ROM set. For the VIC20 and the C64, the ROM set consists of the following files:

- '`kernal`', the Kernal ROM (8 KBytes)
- '`basic`', the Basic ROM (8 KBytes)
- '`chargen`', the character generator ROM (4 Kbytes)

The C128 needs the following files:

- '`kernal`', the Kernal ROM (8 Kbytes)
- '`basic`', the Basic + Editor ROM (32 Kbytes)
- '`chargen`', the character generator ROM (4 Kbytes)

The C128, VIC20 and C64 emulators also need the following DOS ROMs for the hardware-level emulation of the 1541, 1571, 1581, 2000 and 4000 disk drives:

- '`dos1541`', the 1541 drive ROM (16 Kbytes)
- '`dos1541II`', the 1541-II drive ROM (16 Kbytes)
- '`dos1571`', the 1571 drive ROM (32 Kbytes)
- '`dos1581`', the 1581 drive ROM (32 Kbytes)
- '`dos2000`', the 2000 drive ROM (32 Kbytes)
- '`dos4000`', the 4000 drive ROM (32 Kbytes)

In addition to those all emulators can handle a parallel IEEE488 interface (the C64 and C128 via `$df**` extension, the VIC20 via VIC1112 emulation) so they also need the DOS ROM for the IEEE disk drives:

- '`dos2031`', the 2031 drive ROM (16 Kbytes) (DOS 2.6, Commodore ROM images 901484-03 and 901484-05)
- '`dos2040`', the 2040 drive ROM (8 Kbytes) (DOS 1, Commodore ROM images 901468-06, 901468-07)

- '`dos3040`', the 3040 drive ROM (12 Kbytes) (DOS 2, Commodore ROM images 901468-11, 901468-12 and 901468-13)
- '`dos4040`', the 4040 drive ROM (12 Kbytes) (DOS 2, Commodore ROM images 901468-14, 901468-15 and 901468-16)
- '`dos1001`', the 1001/8050/8250 drive ROM (16 Kbytes) (DOS 2.7, Commodore ROM images 901887-01 and 901888-01)

Note that there are other DOS images on the internet. The DOS 2.5 images might be used with the 8050, but it cannot handle the double sided drives of the 1001 and 8250 and it is not supported by VICE.

The PET emulator uses an expanded setup, because there are three major versions of the Basic and the Kernal, and many versions of the Editor ROM. In addition there are cartridge ROM sockets.

The Kernal files contain the memory from range $F000-$FFFF, the Basic ROMs either the range $C000-$DFFF or $B000-$DFFF. To handle the different screen sizes and keyboards, different so-called "editor-ROMs" for the memory range $E000-$E800 are provided. The PET ROMs have the following names:

- '`kernal1`', the PET2001 Kernal ROM (4 KBytes) (Commodore ROM images 901447-06 and 901447-07)
- '`kernal2`', the PET3032 Kernal ROM (4 KBytes) (Commodore ROM image 901465-03)
- '`kernal4`', the PET4032/8032 Kernal ROM (4 KBytes) (Commodore ROM image 901465-22)
- '`basic1`', the PET2001 Basic 1 ROM (8 KBytes) (Commodore ROM images 901447-09, 901447-02, 901447-03, 901447-04.bin. The -09 ROM is the revised -01 ROM)
- '`basic2`', the PET3032 Basic 2 ROM (8 KBytes) (Commodore ROM images 901465-01 and 901465-01)
- '`basic4`', the PET4032/8032 Basic 4 ROM (12 KBytes) (Commodore ROM images 901465-23, 901465-20 and 901465-21. The -23 ROM is a revised -19 ROM)
- '`edit1g`', the PET2001 editor for graphics keyboards (2 KBytes) (Commodore ROM image 901447-05)
- '`edit2b`', the PET3032 editor for business keyboards (2 KBytes) (Commodore ROM image 901474-01)
- '`edit2g`', the PET3032 editor for graphics keyboards (2 KBytes) (Commodore ROM image 901447-24)
- '`edit4g40`', the PET4032 editor for graphics keyboards (2 KBytes) (Commodore ROM image 901498-01)
- '`edit4b40`', the PET4032 editor for business keyboards (2 KBytes) (Commodore ROM image 901474-02)
- '`edit4b80`', the PET8032 editor for business keyboards (2 KBytes) (Commodore ROM image 901474-04-?)
-
- '`chargen`', the character generator ROM (2k). It has two sets with 128 chars each. The second (inverted) half of each set is computed from the first half by inverting it. This is a PET hardware feature. (Commodore ROM image 901447-10)

- '`chargen.de`', the character generator ROM (2k). This version is a patched German charset, with the characters [, \ and] replaced by umlauts. It has been provided by U. Guettich and he reports that it is supported by some programs.

The PETs also have sockets for extension ROMs for the addresses $9000-$9FFF, $A000-$AFFF and $B000-$BFFF (the last one for PET2001 and PET3032 only). You can specify ROM image files for those extensions command line options `-petrom9`, `-petromA` and `-petromB` resp.

An alternative would be to specify a long kernal ROM with the `-kernal` option that includes the extension ROM areas.

Also, you can specify replacements for the basic ROM at $B000-$DFFF with the `-petromBasic` option and for the editor ROM at $E000-$E7FF with the `-petromEditor` option.

The CBM-II emulator again uses another setup. For those models the kernal used is the same for all. However, for different amounts of memory exist different versions of the BASIC ROMs. The 128k RAM version (C610, C710, B128) uses one bank of 64k for the BASIC text and another one for all the variables. The 256k RAM version uses one bank for text, one for variables, one for arrays and one for strings.

Also the character generator ROMs have a format different from the above. The other character ROMs have 8 bytes of pixel data per character. Those ROMs have 16 bytes per character instead. The C6x0 only uses the first 8 of it, but the C7x0 uses 14 lines per character and needs those increased ROMs. Both ROMs hold, like the PET, two character sets with 128 characters each. Again the second half of the full (256 char) character set is computed by inverting.

- '`kernal`', the KERNAL (8k) for the business machines (6xx/7xx)
- '`kernal.500`', the KERNAL (8k) for the personal machine (510) (901234-02)
- '`basic.128`', the CBM-II 128k BASIC (16k)
- '`basic.256`', CBM-II 256k BASIC (16k)
- '`basic.500`', C510 BASIC (16k) (901236-02 + 901235-02)
- '`chargen.500`', character generator ROM for the C5x0 (4k) (901225-01)
- '`chargen.600`', character generator ROM for the C6x0 (4k)
- '`chargen.700`', character generator ROM for the C7x0 (4k)

4.2 Keymap files

Keymap files are used to define the keyboard layout, defining which key (or combination of keys) must be mapped to each keysym.

In other words, the keyboard emulation works like this: whenever the user presses or releases a key while the emulation window has the input focus, the emulator receives an X-Window event with a value that identifies that key. That value is called a *keysym* and is unique to that key. The emulator then looks up that keysym in an internal table that tells it which key(s) to press or release on the emulated keyboard.

This table is described by the keymap file, which is made up of lines like the following:

```
KEYSYM ROW COLUMN SHIFTFLAG
```

Where:

- KEYSYM is a string identifying the keysym: you can use the xev utility (shipped with the X Window system) to see what keysym is bound to any key;
- ROW and COLUMN identify the key on the emulated keyboard;
- SHIFTFLAG can have one of the following values:
  - 0: the key is never shifted;
  - 1: the key is shifted;
  - 2: the key is the left shift;
  - 4: the key is the right shift;
  - 8: the key can be (optionally) shifted by the user.

The SHIFTFLAG is useful if you want certain keys to be "artificially" shifted by the emulator, and not by the user. For example, ⟨F2⟩ is shifted on the C64 keyboard, but you might want it to be mapped to the unshifted ⟨F2⟩ key on the PC keyboard. To do so, you just have to use a line like the following:

```
F2 0 4 1
```

where 0 and 4 identify the key (row 0, column 4 on the keyboard matrix), and 1 specifies that every time the user presses ⟨F2⟩ the shift key on the C64 keyboard must be pressed.

There are also some special commands you can put into the keyboard file; they are recognized because they start with an exclamation mark:

- !CLEAR clears the currently loaded keyboard map; it is necessary to put this at the beginning of the file if you want the keymap file to override all of the current internal settings;
- !LSHIFT, !RSHIFT, followed by a row and a column value, specify where the left and right shift keys are located on the emulated keyboard; for example, C64 default keymaps will specify

```
!LSHIFT 1 7
!RSHIFT 6 4
```

Any line starting with the # sign, instead, is completely ignored. This is useful for adding comments within the keymap file.

VICE keymap files have the '.vkm' default extension, and every emulator comes with a default positional mapping and a default symbolic mapping.

4.3 Palette files

Palette files are used to specify the colors used in the emulators. They are made up of lines like the following:

```
RED GREEN BLUE DITHER
```

where RED, GREEN and BLUE are hexadecimal values ranging from 0 to FF and specifying the amount of red, green and blue you want for each color and DITHER is a 4-bit hexadecimal number specifying the pattern you want when rendering on a B/W display.

You have to include as many lines as the number of colors the emulated machine has, and the order of the lines must respect the one used in the machine (so the N'th line must contain the specifications for color N - 1 in the emulated machine).

Lines starting with the # sign are completely ignored. This is useful for adding comments (such as color names) within the palette file.

For example, the default PET palette file (which has only two colors, 0 for background and 1 for foreground), looks like the following:

```
#
# VICE Palette file
#
# Syntax:
# Red Green Blue Dither
#

# Background
00 00 00 0

# Foreground
00 FF 00 F
```

4.4 Romset files

The Romset files are not used by default on all emulators. You might have recognized that the names of the ROM images are saved in resources. Loading a Romset file now just means a 'shortcut' to changing all the resources with ROM image names and reloading the ROMs.

The PET and CBM-II emulators use this feature to change between the different ROM versions available for those machines. E.g. the Romset file for the PET 2001 is

```
KernalName="pet2001"
EditorName=
ChargenName="chargen"
RomModule9Name=
RomModuleAName=
RomModuleBName=
```

As you can see, the file even uses the same syntax as the resource file, it is just a bit stripped down.

5 Basic operation

This section describes the basic things you can do once the emulator has been fired up.

5.1 The emulation window

When the emulator is run, the screen of the emulated machine is displayed in a standard X Window which we will call the *emulation window*. This window will be updated in real time, displaying the same contents that a real monitor or TV set would.

Below the emulation window there is an area which is used to display information about the state of the emulator; we will call this area the *status bar*.

On the extreme left of the status bar, there is a *performance meter*. This displays the current relative speed of the emulator (as a percentage) and the update frequency (in frames

per second). All the machines emulated are PAL, so the update frequency will be 50 frames per second if your system is fast enough to allow emulation at the speed of the real machine.

On the extreme right of the status bar, there is a *drive status indicator*. This is only visible if the hardware-level ("True") 1541 emulation is turned on. In that case, the drive status indicator will contain a rectangle emulating the drive LED and will display the current track position of the drive's read/write head.

5.2 Using the menus

It is possible to execute some commands and change emulation parameters while the emulator is running: when the pointer is over the emulation window, two menus are available by pressing either the left or right mouse buttons. The left mouse button will open the *command menu* from which several emulation-related commands can be executed; the right mouse button will open the *settings menu* from which emulation parameters can be changed. The basic difference between the command and the settings menu is that, while commands have only effect on the current session, settings can be saved and later used with the "Save settings" and "Load settings" right-button menu items, respectively. "Restore default settings" restores the factory defaults. See Chapter 6 [Settings and resources], page 25. for more information about how settings work in VICE.

Sometimes commands can be reached via *shortcuts* or *hotkeys*, i.e., it is possible to execute them by pressing a sequence of keys instead of going through the menu with the mouse. Where shortcuts exist, they are displayed in parentheses at the right edge of the menu item. In VICE, all shortcuts must begin with the ⟨Meta⟩ or ⟨Alt⟩ key. So, for example, to attach a disk image to drive #8 (the corresponding menu item displays "M-8"), you have to press the ⟨Meta⟩ (or ⟨Alt⟩) and then ⟨8⟩.

Note that no other key presses are passed on to the emulated machine while either ⟨Meta⟩ or ⟨Alt⟩ are held down.

5.3 Getting help

At any time, if you get stuck or do not remember how to perform a certain action, you can use the "Browse manuals" command (left button menu). This will popup a browser and open the HTML version of this documentation. Notice that this requires VICE to be properly (and fully) installed with a '`make install`'.

The browser can be specified via the `HTMLBrowserCommand` string resource (see Chapter 6 [Settings and resources], page 25 for information about resources). Every '`%s`' in the string will be replaced with a URL to the VICE HTML pages.

5.4 Using the file selector

In those situations where it is necessary to specify a file name, all of the VICE emulators will pop up a file selector window allowing you to select or specify a file interactively.

To the left of the file selector, there is a list of ancestor directories: by clicking on them, you can ascend the directory tree. To the right, there is a list of the files in the current directory; files can be selected by clicking on them. If you click on a directory, that directory becomes the current one; if you click on an ordinary file, it becomes the active selection.

At the top, there is a *directory box*, with the complete path of the current directory, and a *file name box*, with the name of the currently selected file. At the bottom there are two buttons: "OK" confirms the selected file and "Cancel" abandons the file selector without cancelling the operation.

It is also possible to specify what files you want to show in the file selector by writing an appropriate shell-like pattern in the directory box; e.g., '`~/*.[dx]64`' will only show files in the home directory whose name ends with either '`.d64`' or with '`.x64`'.

5.5 Using disk and tape images

The emulator is able to emulate disk drives and (read-only) tape recorders if provided with suitable *disk images* or *tape images*. An *image* is a raw dump of the contents of the media, and must be *attached* before the emulator can use it. "Attaching" a disk or tape image is like "virtually" inserting a diskette or a cassette into the disk drive or the tape recorder: once an image is attached, the emulator is able to use it as a storage media.

There are five commands (in the left button menu) that deal with disk and tape images:

- Attach Disk Image
- Detach Disk Image
- Attach Tape Image
- Detach Tape Image
- Smart-attach a file

The first four commands are used to insert and remove the virtual disks and cassettes from the respective units. On the other hand, the last commands tries to guess the type of the image you are attaching from its name and size, and attaches it to the most reasonable device.

Supported formats are `D64` and `X64` for disk images (devices 8, 9 and 10) and `T64` for tape images. Notice that `T64` support is *read-only*, and that the cassette is automatically rewound when you reach its end.

Another important feature is that raw Commodore BASIC binary files and .P00 files can be attached as tapes. As you can autostart a tape image when it is attached (see Section 5.5.2 [Autostart], page 24), this allows you to autostart these particular files as well.

You can attach a disk for which you do not have write permissions: when this happens, the 1541 emulator will emulate a write-protected disk. This is also useful if you want to prevent certain disk images from being written to; in the latter case, just remove the write permission for that file, e.g., by doing a `chmod a-w`.

5.5.1 Previewing the image contents

It is possible to examine the directory of a disk or tape image before attaching it. Just press the "Contents" button in the file selector window and a new window will pop up with the contents of the selected image.

Notice that this function automatically translates the directory from PETSCII to ASCII; but, due to differences in the two encodings, it is not always possible to translate all the characters, so you might get funny results when "weird" characters such as the semi-graphical ones are being used.

5.5.2 "Autostarting" an image

If you want to reset the machine and run the first program on a certain image without typing any commands at the Commodore BASIC prompt, you can use the "Autostart" button in the file selector window after selecting a proper disk or tape image file.

Notice that, if true drive emulation is turned on, it will be turned off before running the program and then turned on again after it has been loaded. This way, you get the maximum possible speed while loading the file, but you do not lose compatibility once the program itself is running.

This method is not completely safe, because some autostarting methods might cause the true drive emulation not to be turned on again. In such cases, the best thing to do is to disable kernal traps (which will cause true drive emulation to be always kept turned on), or to manually load the program with true drive emulation turned on.

5.5.3 Using compressed files

It is also possible to attach disk or tape images that have been compressed through various algorithms; compression formats are identified from the file extension. The following formats are supported (the expected file name extension is in parenthesis):

- GNU Zip (`.gz` or `.z`);
- BZip version 2 (`.bz2`);
- PkZip (`.zip`);
- GNU Zipped TAR archives (`.tar.gz`, `.tgz`);
- Zoo (`.zoo`).

PkZip, `tar.gz`, `lha` and `zoo` support is *read-only* and always uses the *first* `T64` or `D64` file in the archive. So archives containing multiple files will always be handled as if they contain only a single file.

Windows and MSDOS don't contain the needful programs to handle compressed archives. Get gzip and unzip for Windows at `ftp://ftp.freesoftware.com/pub/infozip/WIN32` and for MSDOS at `ftp://ftp.freesoftware.com/pub/infozip/MSDOS`. Don't use pkunzip for MSDOS, it doesn't work. The programs to use BZip2 archives may be found at `http://sourceware.cygnus.com/bzip2`. Just put the programs (unzip.exe, gzip.exe, bzip2.exe) into a directory of your search path (e.g. C:\DOS or C:\WINDOWS\COMMAND; have a look at the PATH variable).

5.5.4 Using Zipcode and Lynx images

Since version 0.15, the VICE emulators have been able to attach disks packed with Zipcode or Lynx directly, removing the need to manually convert them into `D64` or `X64` files with `c1541`. This is achieved by automatically invoking `c1541`, letting it decode the file into a temporary image and attaching the resulting temporary image read-only. For this to work, the directory containing `c1541` must be in your `PATH`.

This uses the `-unlynx` and `-zcreate` options of `c1541` (see Section 12.3 [c1541 commands and options], page 113); these commands are not very reliable yet, and could fail with certain kinds of Lynx and Zipcode images (for example, they cannot deal with `DEL` files properly). So please use them with caution.

Lynx files usually come as '`.lnx`' files which are unpacked into single disk images. On the other hand, Zipcode files do not have a particular extension (although '`.z64`' is sometimes used), and represent a disk by means of component files, named as follows:

- '`1!NAME`'
- '`2!NAME`'
- '`3!NAME`'
- '`4!NAME`'

If you attach as a disk image (or smart-attach) any one of these files, the emulator will simply pick up the other three (by examining the name) and then build a disk image using all four.

5.6 Resetting the machine

You can reset the emulated machine at any time by using the "Reset" command from the command menu. There are two types of reset:

- *soft reset*, which simply resets the CPU and all the other chips;
- *hard reset*, which also clears up the contents of RAM.

A *soft reset* is the same as a hardware reset achieved by pulling the RESET line down; a *hard reset* is more like a power on/power off sequence in that it makes sure the whole RAM is cleared.

It is possible that a soft reset may not be enough to take the machine to the OS initialization sequence: in such cases, you will have to do a hard reset instead.

This is especially the case for the CBM-II emulators. Those machines examine a memory location and if they find a certain "magic" value they only do what you know from the C64 as `Run/Stop-Restore`. Therefore, to really reset a CBM-II use hard reset.

6 Settings and resources

In the VICE emulators, all the settings are stored in entities known as called *resources*. Each resource has a name and a value which may be either an integer or a string. Integer values are often used as boolean values with the usual convention of using zero for "false" and any other value for "true".

Resource values can be changed via the right-button menu (the *settings* menu), via command-line options or via the *resource file*.

The *resource file* is a human-readable file containing resource values: it is called '`vicerc`' and is stored in the directory '`.vice/`' in the user's home directory. It is possible to dump the current values of the resources into that file or load the values stored into that file as the current values, at any time. This is achieved with the "Save settings" and "Load settings" right menu items. A third menu item, "Restore Default Settings", can be used to reset all the values to the factory defaults.

A special resource, `SaveResourcesOnExit`, if set to a non zero value, causes the emulator to ask you if you want to save the current (changed) settings before exiting, and can be toggled with the "Save settings on exit" command from the right-button menu.

Notice that not all the resources can be changed from the menus; some of them can only be changed by manually modifying the resource file or by using command-line options.

6.1 Format of resource files

A resource file is made up of several sections; sections have the purpose of separating the resources of a certain emulator from the ones of the other emulators. A section starts with the name of an emulator in brackets (e.g., '`[C64]`') and ends when another section starts or when the file ends.

Every line in a section has the following format:

```
RESOURCE=VALUE
```

where `RESOURCE` is the name of a resource and `VALUE` is its assigned value. Resource names are case-sensitive and resource values are either strings or integers. Strings must start and end with a double quote character (`"`), while integers must be given in decimal notation.

Here is an example of a stripped-down '`.vice/vicerc`' file:

```
[VIC20]
HTMLBrowserCommand="netscape %s"
SaveResourcesOnExit=0
FileSystemDevice8=1
FSDevice8ConvertP00=1
FSDevice8Dir="/home/ettore/cbm/stuff/vic20p00"
FSDevice8SaveP00=1
FSDevice8HideCBMFiles=1
[C64]
HTMLBrowserCommand="netscape %s"
SaveResourcesOnExit=1
FileSystemDevice8=1
FSDevice8ConvertP00=1
FSDevice8Dir="/home/ettore/cbm/stuff/c64p00"
FSDevice8SaveP00=1
FSDevice8HideCBMFiles=1
```

Notice that, when resource values are saved with "Save settings", the emulator only modifies its own section, leaving the others unchanged.

6.2 Using command-line options to change resources

Resources can also be changed via command-line options.

Command-line options always override the defaults from `.vice/vicerc`, and their assignments last for the whole session. So, if you specify a certain command-line option that changes a certain resource from its default value and then use "Save Settings", the value specified with the command-line option will be saved back to the resource file.

Command-line options can begin with with a minus sign ('`-`') or with a plus sign ('`+`'). Options beginning with a minus sign may require an additional parameter, while the ones beginning with the plus sign never require one.

Moreover, options beginning with a plus sign always have a counterpart with the same name, but with a minus sign; in that case, the option beginning with a minus sign is used to *enable* a certain feature, while the one beginning with a plus sign is used to *disable* the same feature (this is an X11 convention). For example, `-mitshm` enables support of MITSHM, while `+mitshm` disables it.

6.3 Performance settings

It is possible to control the emulation speed by using the "Maximum speed" menu item in the right-button menu. The default setting is `100`, which causes the emulation to never run faster than the real machine. A higher value allows the emulator to run faster, a lower one may force it to run slower. The setting "No limit" means to run as fast as possible, without limiting speed.

It is also possible to control the emulator's rate of frame update using the "Refresh rate" setting; the value ranges from "1/1" (update 1/1 of the frames of the real machine, that is 50 frames per second) to "1/10" (update 1 every 10 frames) and can be changed via the "Refresh Rate" submenu. The "Auto" setting means to dynamically adapt the refresh rate to the current speed of the host machine, making sure the maximum speed specified by the via "Maxium speed" is always reached if possible. In any case, the refresh rate will never be worse than 1/10 if this option is specified.

Note that you cannot simultaneously specify "Auto" as the refresh rate and "No limit" as the maximum speed..

Moreover, a special *warp speed* mode is provided and can be toggled with the "Enable Warp Mode" menu item. If this mode is enabled, it will cause the emulator to disable any speed limit, turn sound emulation off and use a 1/10 refresh rate, so that it will run at the maximum possible speed.

6.3.1 Performance resources

`Speed` Integer specifying the maximum relative speed, as a percentage. `0` stands for "no limit".

`RefreshRate`
Integer specifying the refresh rate; a value of `n` specifies a refresh rate of 1/`n`. A value of `0` enables automatic frame skipping.

`WarpMode` Booolean specifying whether "warp mode" is turned on or not.

6.3.2 Performance command-line options

`-speed VALUE`
Specifies the maximum speed as a percentage. `0` stands for "no limit". (Same as setting the `Speed` resource.)

`-refresh VALUE`
Specifies refresh rate; a value of `n` specifies a refresh rate of 1/`n`. A value of `0` enables automatic frame skipping. (Same as setting the `RefreshRate` resource.)

`-warp`
`+warp` Enables/disables warp mode (`WarpMode=1`, `WarpMode=0`).

6.4 Video settings

The following right-button menu items control the video output. On emulators that include two video chips (like `x128`) all options but XSync exist twice, once for each chip. XSync is shared between the video chips.

- "Video Cache" enables a video cache that can speed up the emulation when little graphics activity is going on; it is especially useful when you run the emulator on a networked X terminal as it can reduce the network bandwidth required. However, this setting can actually make the emulator slower when there is little graphics activity and the amount of work needed to maintain the cache is greater than the amount of work that would be wasted by not using it (if any).
- "Double Size" toggles *double-size mode*, which makes the emulation window twice as big. When emulating a 80-column PET, only the height is doubled, so that the aspect ratio is closer to that of the real thing.
- "Double Scan" toggles *double-scan mode*, which causes the emulator to draw only odd lines when running in double-size mode (this saves some CPU time and also makes the emulation window look more like an old monitor).
- 'Use XSync()" causes the emulator to call the X11 function `XSync()` before updating the emulation window: this might be necessary on low-end systems to prevent it from consuming so many system resources that it becomes impossible for the user to interact with it.

6.4.1 Video resources

The following resources affect the screen emulation. The prefix of some of the resources and commandline options denote the video chip the values apply to.

`UseXSync` Boolean specifying whether `XSync()` is called after updating the emulation window.

`MITSHM` Integer specifying whether VICE should try to use the shared memory extensions (MITSHM) when starting up. The shared memory extensions make things a lot faster but might not be available on your system. You will not be able to use these extensions if you are sitting at an X terminal while running the emulator on a remote machine across a network. Valid values are: 0 = do not use MITSHM, 1 = do use MITSHM, -1 = try to autodetect availability on startup (default). The last is a simple test if the emulator runs across a network and if so disables MITSHM (If you have problems with this test please report it).

`PrivateColormap`
Boolean specifying whether VICE should install a private colormap at startup. This makes sense for 8-bit displays that could run out of colors if other color-hungry applications are running at the same time.

`DisplayDepth`
Integer specifying the depth of the host display. The value '`0`' (the default) causes the emulator to autodetect it.

6.4.2 Video command-line options

`-xsync`
`+xsync` Enable/disable usage of `XSync()` when updating the emulation window (`UseXSync=1`, `UseXSync=0`).

`-mitshm`
`+mitshm` Enable/disable usage of the MITSHM extensions (`MITSHM=1`, `MITSHM=0`).

`-install`
`+install` Enable/disable installation of a private colormap (`PrivateColormap=1`, `PrivateColormap=0`).

`-displaydepth DEPTH`
Specify the display depth (`DisplayDepth`).

`-fullscreen`
`+fullscreen`
Enable/disable fullscreen mode

6.5 Keyboard settings

It is possible to specify whether the "positional" or "symbolic" keyboard mapping should be used with the "Keyboard mapping type" submenu (see Section 2.7 [Keyboard emulation], page 9 for an explanation of positional and symbolic mappings).

The keyboard settings submenu also allows you to:

- Load custom-made positional and symbolic keymap files ("Set symbolic keymap file" and "Set positional keymap file").
- Dump the current keymap to a user-defined keymap file ("Dump to keymap file").

6.5.1 Keyboard resources

`KeymapIndex`
Integer identifying which keymap is being used; `0` indicates symbolic mapping, `1` positional mapping. For the PET the even values represent symbolic mapping, odd positional. Then add `0` for UK business keyboard or `2` for graphics keyboard.

`KeymapSymFile`
String specifying the name of the keymap file for the symbolic mapping (see Section 2.7 [Keyboard emulation], page 9, all but PET and CBM-II).

`KeymapPosFile`
String specifying the name of the keymap file for the positional mapping (see Section 2.7 [Keyboard emulation], page 9, all but PET and CBM-II).

`KeymapBusinessUKSymFile`
`KeymapBusinessUKPosFile`
String specifying the name of the keymap file for the symbolic and positional mapping for the UK business keyboard (see Section 2.7 [Keyboard emulation], page 9, PET and CBM-II).

`KeymapGraphicsSymFile`
`KeymapGraphicsPosFile`
: String specifying the name of the keymap file for the symbolic and positional mapping for the graphics keyboard (see Section 2.7 [Keyboard emulation], page 9, PET only).

`KeymapBusinessDESymFile`
`KeymapBusinessDEPosFile`
: String specifying the name of the keymap file for the symbolic and positional mapping for the German business keyboard. (see Section 2.7 [Keyboard emulation], page 9, PET only).

6.5.2 Keyboard command-line options

`-keymap N` Specifies which keymap is being used; `0` indicates symbolic mapping, `1` positional mapping (as for the `KeymapIndex` resource).

`-symkeymap NAME`
: Specify '`NAME`' as the symbolic keymap file (`KeymapSymFile`).

`-poskeymap NAME`
: Specify '`NAME`' as the positional keymap file (`KeymapPosFile`).

`-symdekeymap NAME`
: Specify '`NAME`' as the symbolic keymap file (`KeymapSymDeFile`).

`-buksymkeymap NAME`
`-bukposkeymap NAME`
: Specify '`NAME`' as the symbolic/positional keymap file for the UK business keyboard (`KeymapBusinessUKSymFile`, `KeymapBusinessUKPosFile`, PET and CBM-II).

`-grsymkeymap NAME`
`-grposkeymap NAME`
: Specify '`NAME`' as the symbolic/positional keymap file for the graphics keyboard (`KeymapGraphicsSymFile`, `KeymapGraphicsPosFile`, PET only).

`-bdesymkeymap NAME`
`-bdeposkeymap NAME`
: Specify '`NAME`' as the symbolic/positional keymap file for the German business keyboard (`KeymapBusinessDESymFile`, `KeymapBusinessDEPosFile`, PET only).

6.6 Joystick settings

6.6.1 Joystick command-line options

`-joydev1 <0-8>`
`-joydev2 <0-8>`
: Set the device for joystick emulation of port 1 and 2, respectively (`JoyDevice1`, `JoyDevice2`).

`-extrajoydev1 <0-8>`
`-extrajoydev2 <0-8>`
: Set device for extra joystick port 1 and 2.

`-mouse`
`+mouse`
: Enable/Disable mouse grab

`-mousetype <value>`
: Select the mouse type (0 = 1351, 1 = NEOS, 2 = Amiga, 3 = Paddles)

`-mouseport <value>`
: Select the joystick port the mouse is attached to

`-lightpen`
`+lightpen`
: Enable/Disable lightpen emulation

`-lightpentype <type>`
: Set lightpen type

6.7 Sound settings

The following menu items control sound output:

- "Enable sound playback" turns sound emulation on and off.
- "Sound synchronization" specifies the method for syncronizing the sound playback. Possible settings are:
  - "Flexible", i.e., the audio renderer flexibly adds/removes samples to the output to smoothly adapt the playback to slight changes in the speed of the emulator.
  - "Adjusting" works like "flexible", but supports bigger differences in speed. For example, if the emulation speed drops down from from 100% to 50%, audio slows down by the same amount too.
  - "Exact", instead, makes the audio renderer output always the same sounds you would hear from the real thing, without trying to adapt the ratio; to compensate the tolerances in speed, some extra frames will be skipped or added.
- "Sample rate" specifies the sampling frequency, ranging from 8000 to 48000 Hz (not all the sound cards and/or sound drivers can support all the frequencies, so actually the nearest candidate will be chosen).
- "Buffer size" specifies the size of the audio buffer; the bigger the buffer, the longer the delay with which sounds are played. You should pick the smallest value your machine can handle without problems.
- "Sound suspend time", will cause the audio playback to pause for the specified number of seconds whenever some clicking happens. If "Keep going" is selected, no pausing is done.
- "Oversample" specifies an oversampling factor, from 1 to 8 times (warning: this eats CPU cycles!).

6.7.1 Sound resources

`Sound` Boolean specifying whether audio emulation is turned on.

`SoundSpeedAdjustment`
: Integer specifying what speed adjustment method the audio renderer should use. Possible values are:
  - `0`: "flexible"
  - `1`: "adjusting"
  - `2`: "exact"

`SoundSampleRate`
: Integer specifying the sampling frequency, ranging from 8000 to 48000 Hz (not all the sound cards and/or sound drivers can support all the frequencies, so actually the nearest candidate will be chosen).

`SoundBufferSize`
: Integer specifying the size of the audio buffer, in milliseconds.

`SoundSuspendTime`
: Integer specifying the pause interval when audio underflows ("clicks") happen. `0` means no pause is done.

`SoundDeviceName`
: String specifying the audio driver.

  Implemented drivers are:
  - `aix`, for the IBM AIX sound driver.
  - `uss`, for the Linux/FreeBSD Universal Sound System driver (`SoundDeviceArg` specifies the audio device, '`/dev/dsp`' by default);
  - `sgi`, for the Silicon Graphics audio device (`SoundDeviceArg` specifies the audio device, '`/dev/audio`' by default);
  - `sun`, for the Solaris audio device (unfinished; `SoundDeviceArg` specifies the audio device, '`/dev/audio`' by default).
  - `hpux`, for the HP-UX audio device (unfinished; `SoundDeviceArg` specifies the audio device, '`/dev/audio`' by default).
  - `sdl`, for the Simple DirectMedia Layer audio driver.
  - `esd`, for EsounD, the Enlightened Sound Daemon; `SoundDeviceArg` specifies the ESD server ('`host:port`') to connect, empty by default.
  - `dummy`, fully emulating the SID, but not actually playing samples.
  - `dump`, writing all the write accesses to the registers to a file (specified by `SoundDeviceArg`, default value is `vicesnd.sid`);
  - `speed`, like `dummy` but also calculating samples (mainly used to evaluate the speed of the sample generator);
  - `fs`, writing samples to a file (specified by `SoundDeviceArg`; default is '`vicesnd.raw`');

  These drivers will actually be present only if the VICE configuration script detected the corresponding devices at the time of compilation.

`SoundDeviceArg`
: String specifying an additional parameter for the audio driver (see `SoundDeviceName`).

6.7.2 Sound command-line options

`-sound`
`+sound` Turns sound emulation on (`Sound=1`) and off (`Sound=0`).

`-soundsync N`
Specify `N` as the sound speed adjustment method (`SoundSpeedAdjustment`).

`-soundrate RATE`
Specifies the sound playback sample rate (`SoundSampleRate`).

`-soundbufsize SIZE`
Specifies the size of the audio buffer in milliseconds (`SoundBufferSize`).

`-soundfragsize <value>`
Set sound fragment size (0 = small, 1 = medium, 2 = large)

`-sounddev NAME`
Specifies the name of the audio device (`SoundDeviceName`).

`-soundarg ARG`
Specifies an additional parameter for the audio device (`SoundDeviceArg`).

`-soundrecdev <name>`
Specify recording sound driver

`-soundrecarg <args>`
Specify initialization parameters for recording sound driver

6.8 Tape settings

These settings are used to control the hardware-level emulation of the Tape drive.

6.8.1 Tape command-line options

`-dsresetwithcpu`
`+dsresetwithcpu`
Enable/Disable automatic Datasette-Reset

`-dszerogapdelay <value>`
Set delay in cycles for a zero in the tap

`-dsspeedtuning <value>`
Set number of cycles added to each gap in the tap

6.9 Drive settings

These settings are used to control the hardware-level emulation of the Disk drives. When hardware-level emulation is turned on, only drives 8 and 9 are being emulated.

The following settings affect both drives:

- "Enable true drive emulation" enables the (slow) hardware-level emulation of the drives for maximum compatibility. This must be turned on for any of the following settings to have effect.

- "Drive sync factor" specifies the speed of the drive's CPU. This can be used to help loading certain programs that have trouble with the default PAL setting (for example, programs designed for NTSC machines). The ratio is calculated as follows:

  ```
  sync_factor = 65536 * clk_drive / clk_machine
  ```

  where `clk_drive` and `clk_machine` are clock speeds in MHz. The menu lets you choose between the PAL and NTSC values, and also lets you specify whatever value you want. Be careful when changing it, though, because a wrong value can break things and even corrupt disk images.

The following settings, instead, are specific of each drive:

- "Drive model" specifies the model of the drive being emulated. **Warning:** This will reset the drive.
- "Enable parallel cable" enables emulation of a SpeedDOS parallel cable; if you switch this option on and replace the original Commodore ROMs with SpeedDOS-compatible ones, you can speed up loading/saving times.
- "Idle method" specifies which method the drive emulation should use to save CPU cycles in the host CPU. There are three methods:
  - *Skip cycles*: Each time the serial line is accessed by the C64, the drive executes all the cycles since the last time it ran. If the number of elapsed cycles is larger than a certain value, the drive discards part of them.
  - *Trap idle*: The disk drive is still emulated upon serial line accesses as with the previous option, but it is also always emulated at the end of each screen frame. If the drive gets into the DOS idle loop, only pending interrupts are emulated to save time.
  - *No traps*: Like "Trap idle", but without any traps at all. So basically the drive works exactly as with the real thing, and nothing is done to reduce the power needs of the drive emulation.

  The first option ("Skip cycles") is usually best for performance, as the drive is emulated as little as possible; on the other hand, you may notice sudden slowdowns (when the drive executes several cycles at once) and the LED status is never updated (because it would not be possible to do correctly so). Moreover, if the drive tries to get in sync with the computer in some weird way and the computer does not access the serial line for a long time, it is possible that some cycles are discarded and the sync is lost. Notice that this hack will have no effect on performance if a program continuously reads from the IEC port, as the drive will have to be fully emulated in any case (some stupid programs do this, even when they don't actually need to use the drive).

  The second option ("Trap idle") is usually a bit slower, as at least interrupts are always emulated, but ensures the LED state is always updated correctly and always keeps the drive and the computer in sync. On the other hand, if a program installs a non-standard idle loop in the drive, the drive CPU has to be emulated even when not necessary and the global emulation speed is then *much* slower.
- "40-track image support" specifies how 40-track ("extended") disk images should be supported. There are three possible ways:
  - "Never extend" never extends disk images at all (so if a program tries to write tracks beyond the 35th, it is not allowed to do so);

- "Ask on extend" prompts the user as soon as a program tries to write tracks beyond the 35th, and the user can then choose whether he wants the disk image to be extended or not;
- "Extend on access" simply extends the disk image as soon the program needs it, without prompting the user.

6.9.1 Drive resources

`DriveTrueEmulation`
: Boolean controlling whether the "true" drive emulation is turned on.

`Drive8Type`
`Drive9Type`
: Integers specifying the model number for drives 8 and 9. Possible values are `1541`, `1571`, `1581`, `2000`, `4000` and `2031`.

`Drive8ParallelCable`
`Drive9ParallelCable`
: Booleans controlling whether the SpeedDOS-compatible cable is emulated or not for drives 8 and 9.

`Drive8ExtendImagePolicy`
`Drive9ExtendImagePolicy`
: Integer specifying the policy for 40-track support for drives 8 and 9. Possible values are `0` (never extend), `1` (ask on extend), `2` (extend on access).

`Drive8IdleMethod`
`Drive9IdleMethod`
: Integers specifying the idling method for the drive CPU. Possible values are `0` (none), `1` (skip cycles), `2` (trap idle). See Section 6.9 [Drive settings], page 33.

`DosName1541`
`DosName1571`
`DosName1581`
`DosName2000`
`DosName4000`
`DosName2031`
: Strings specifying the names of the ROM images for the drive emulation.

6.9.2 Drive command-line options

`-truedrive`
`+truedrive`
: Turns true drive emulation on (`DriveTrueEmulation=1`) and off (`DriveTrueEmulation=0`), respectively.

`-drivesound`
`+drivesound`
: Turns drive sound emulation on (`DriveSoundEmulation=1`) and off (`DriveSoundEmulation=0`), respectively.

`-drive8type TYPE`
`-drive9type TYPE`
`-drive10type TYPE`
`-drive11type TYPE`
Specifies the drive types for drives 8-11, respectively. Possible values for `TYPE` are `1541`, `1542` (meaning 1541-II), `1551`, `1570`, `1571`, `1573` (meaning 1571cr), `1581`, `2000`, `4000`, `2031`, `2040`, `3040`, `4040`, `1001`, `8050` and `8250`.

`-parallel8 <type>`
`-parallel9 <type>`
`-parallel10 <type>`
`-parallel11 <type>`
Set parallel cable type (0: none, 1: standard, 2: Dolphin DOS)

`-drive8idle NUM`
`-drive9idle NUM`
`-drive10idle NUM`
`-drive11idle NUM`
Specifies `NUM` as the idling method for drives 8-11 (0: no traps, 1: skip cycles, 2: trap idle), respectively (`Drive8IdleMethod`, `Drive9IdleMethod`), `Drive10IdleMethod`), `Drive11IdleMethod`).

`-drive8extend NUM`
`-drive9extend NUM`
`-drive10extend NUM`
`-drive11extend NUM`
Specifies `NUM` as the track 40 extend policy in drives 8 and 9, respectively (`Drive8ExtendImagePolicy`, `Drive9ExtendImagePolicy`).

`-dos1541 <name>`
`-dos1541II <name>`
`-dos1551 <name>`
`-dos1570 <name>`
`-dos1571 <name>`
`-dos1571cr <name>`
`-dos1581 <name>`
`-dos2000 <name>`
`-dos4000 <name>`
`-dos2031 <name>`
`-dos2040 <name>`
`-dos3040 <name>`
`-dos4040 <name>`
`-dos1001 <name>`
Specify the ROM names for the 1541, 1541II, 1551, 1570, 1571, 1571cr, 1581, 2000, 4000, 2031, 2040, 3040, 4040 and 1001 emulation respectively.

`-drive8ram2000, +drive8ram2000`
`-drive9ram2000, +drive9ram2000`
`-drive10ram2000, +drive10ram2000`
`-drive11ram2000, +drive11ram2000`
Enable/Disable 8KB RAM expansion at $2000-$3FFF

`-drive8ram4000, +drive8ram4000`
`-drive9ram4000, +drive9ram4000`
`-drive10ram4000, +drive10ram4000`
`-drive11ram4000, +drive11ram4000`
Enable/Disable 8KB RAM expansion at $4000-$5FFF

`-drive8ram6000, +drive8ram6000`
`-drive9ram6000, +drive9ram6000`
`-drive10ram6000, +drive10ram6000`
`-drive11ram6000, +drive11ram6000`
Enable/Disable 8KB RAM expansion at $6000-$7FFF

`-drive8ram8000, +drive8ram8000`
`-drive9ram8000, +drive9ram8000`
`-drive10ram8000, +drive10ram8000`
`-drive11ram8000, +drive11ram8000`
Enable/Disable 8KB RAM expansion at $8000-$9FFF

`-drive8rama000, +drive8rama000`
`-drive9rama000, +drive9rama000`
`-drive10rama000, +drive10rama000`
`-drive11rama000, +drive11rama000`
Enable/Disable 8KB RAM expansion at $A000-$BFFF

`-drive8profdos, +drive8profdos`
`-drive9profdos, +drive9profdos`
`-drive10profdos, +drive10profdos`
`-drive11profdos, +drive11profdos`
Enable/Disable Professional DOS

`-profdos1571 <name>`
`Specify name of Professional DOS 1571 ROM image`

6.10 Peripheral settings

VICE is able to support some special peripherals:

- *file system devices*, pseudo-drives accessing the Unix file system;
- printers.

These features depend on some *kernal traps* that replace the existing routines in the original Commodore operating system with custom-made C routines.

6.10.1 Settings for file system devices

These settings deal with the drive-like peripherals connected to the bus of the emulated machine. The first setting relates to the parallel IEEE488 interface. With this interface a

special engine is used to listen to the bus lines to translates them to the filesystem code. Thus the PET will always detect a drive for example, but it can also use drives 10 and 11 even together with true disk drive emulation.

- "Enable virtual devices", enables the peripheral access via the fast disk emulation (either kernal traps or IEEE488 interface). Both, filesystem and disk image access via fast drive emulation, are affected.

Four peripherals, numbered from 8 to 11, are accessible; each of them provides the following settings:

- "File system access", if enabled, allows the device to emulate a drive accessing a file system directory; note that when a disk image is attached to the same drive, the directory is no longer visible and the attached disk is used instead.
- "File system directory" specifies the directory to be accessed by the drive.
- "Convert P00 file names", if enabled, allows access to P00 files using their built-in name instead of the Unix one.
- "Create P00 files on save", if enabled, creates P00 files (instead of raw CBM files) whenever a program creates a file.

Note that, by default, all drives except 11 create P00 files on save, while drive 11 creates raw CBM files. Those files come without any header, but also with the filename restrictions given by the operating system VICE runs on.

6.10.1.1 Resources for file system devices

`FSDevice8ConvertP00`
`FSDevice9ConvertP00`
`FSDevice10ConvertP00`
`FSDevice11ConvertP00`

Booleans specifying whether on-read support for P00 files is enabled on drives 8, 9, 10 and 11 respectively (on by default).

`FSDevice8SaveP00`
`FSDevice9SaveP00`
`FSDevice10SaveP00`
`FSDevice11SaveP00`

Booleans specifying whether the drives should create P00 files instead of plain CBM ones (on by default for drives 8-10, off for 11).

`FSDevice8HideCBMFiles`
`FSDevice9HideCBMFiles`
`FSDevice10HideCBMFiles`
`FSDevice11HideCBMFiles`

Booleans specifying whether non-P00 files should be invisible to programs running in the emulator (do not hide by default).

`FSDevice8Dir`
`FSDevice9Dir`
`FSDevice10Dir`
`FSDevice11Dir`

Strings specifying the directories to which drives 8, 9, 10 and 11 have access.

6.10.1.2 Command-line options for file system devices

`-iecdevice8`
`+iecdevice8`
Enable/Disable IEC device emulation for device #8

`-iecdevice9`
`+iecdevice9`
Enable/Disable IEC device emulation for device #9

`-iecdevice10`
`+iecdevice10`
Enable/Disable IEC device emulation for device #10

`-iecdevice11`
`+iecdevice11`
Enable/Disable IEC device emulation for device #11

`-device8 <type>`
`-device9 <type>`
`-device10 <type>`
`-device11 <type>`
Set device type for device 8-11 (0: NONE, 1: FILESYSTEM, 2: OPENCBM, 3: BLOCK DEVICE)

`-fs8 PATH`
`-fs9 PATH`
`-fs10 PATH`
`-fs11 PATH`
Specify the paths for the file system access on drives 8, 9, 10 and 11, respectively (`FSDevice8Dir`, `FSDevice9Dir`, `FSDevice10Dir` and `FSDevice11Dir`).

`-flipname <name>`
Specify name of the flip list file image

6.10.2 Printer settings

The VICE emulators can emulate printers connected to either the IEC buffer or the user port. Emulation can be achieved by redirecting the printer output to a file or by piping it through an external process. This is defined by so-called *printer device file names*; a printer device file name can be either a simple path, or a command name preceeded by a pipe symbol '`|`'.

For example, printer device '`filename`' will cause the output to be appended to the file '`filename`', while printer device '`|lpr`' will cause the `lpr` command to be executed and be fed the printer output. The printer output will not be converted but saved as printed by the emulated machine.

Up to three printer devices may be specified through the following resources:

- device 1, whose default value is `print.dump`;
- device 2, whose default value is `|lpr`.
- device 3, whose default value is `|petlp -F PS|lpr`;

So, basically, by default printer device 1 will dump printer output to '`print.dump`'; printer device 2 will print it via `lpr` directly to the printer and device 3 will print it via `petlp` (a not-yet-complete utility that will produce Postscript output from the Commodore printer code) and then to the printer via `lpr`.

6.10.2.1 Printer resources

`PrinterTextDevice1`
`PrinterTextDevice2`
`PrinterTextDevice3`
: Strings specifying the printer devices (see Section 6.10.2 [Printer settings], page 39).

`Printer4TextDevice`
`Printer5TextDevice`
: Integer (ranging from 0 to 2, for device 1-3) specifying what printer device (see Section 6.10.2 [Printer settings], page 39) the IEC printer is using.

`Printer4`

`Printer5` Integer specifying how the IEC printer (device 4-5) is being emulated. (0: NONE, 1: FS, 2: REAL)

`Printer4Driver`
`Printer5Driver`
: String (ascii, mps803, nl10) specifying the IEC printer output driver.

`Printer4Output`
`Printer5Output`
: String specifying the IEC printer output device.

`PrinterUserport`
: Boolean specifying if the user-port printer is being emulated.

`PrinterUserportTextDevice`
: Integer (ranging from 0 to 2, for device 1-3) specifying what printer device the user-port printer is using.

`PrinterUserportDriver`
: String specifying the user-port printer output driver.

`PrinterUserportOutput`
: String specifying the user-port printer output device.

6.10.2.2 Printer command-line options

`-iecdevice4`
`+iecdevice4`
: Enable/Disable IEC device emulation for device #4

`-iecdevice5`
`+iecdevice5`
: Enable/Disable IEC device emulation for device #5

`-device4 <type>`
`-device5 <type>`
Set device type for device 4-5 (0: NONE, 1: FS, 2: REAL)

`-prtxtdev1 <name>`
`-prtxtdev2 <name>`
`-prtxtdev3 <name>`
Specify name of printer text device or dump file

`-pr4txtdev <0-2>`
`-pr5txtdev <0-2>`
Specify printer text output device for IEC printer #4-5

`-pr4output <name>`
`-pr5output <name>`
Specify name of output device for device #4-5 Specify name of output device for device #5-5

`-pr4drv <name>`
`-pr5drv <name>`
Specify name of printer driver for device #4-5 Specify name of printer driver for device #5-5

`-pruser`
`+pruser` Enable/disable emulation of the userport printer emulation (`PrUser=1`, `PrUser=0`).

`-prusertxtdev <0-2>`
Specify printer text output device for userport printer

`-pruseroutput <name>`
Specify name of output device for the userport printer

`-pruserdrv <name>`
Specify name of printer driver for the userport printer

6.10.3 Disabling kernal traps

If you have compatibility problems, you can completely disable Kernal traps with the "Disable kernal traps" option. This will of course disable all the features that depend on it, such as the fast 1541 emulation (so you will have to turn true 1541 emulation on if you want to be able to read or write disk images) and tape support.

6.10.3.1 Resources to control Kernal traps

`VirtualDevices`
Boolean specifying whether all the mechanisms for virtual device emulation should be enabled. Serial IEC devices use kernal traps, parallel IEEE488 devices use an own IEEE488 engine. Both are switched on and off with this resource.

6.10.3.2 Command-line options to control Kernal traps

`-virtualdev`
`+virtualdev`
Enable (`VirtualDevices=1`) or disable (`VirtualDevices=0`) virtual devices.

6.11 RS232 settings

The VICE emulators can emulate the RS232 device most of the machines have. The C64, C128 and VIC20 emulators emulate the userport RS232 interface at 300 and 1200 baud. The C64 and C128 can also use the 9600 baud interface by Daniel Dallmann, using the shift registers of the two CIA 6526 chips. The PET can have a 6551 ACIA RS232 interface when running as a SuperPET, and the CBM-II has such an ACIA by default. The C64 and C128 emulators can emulate an ACIA 6551 (also known as Datapump for example) as extension at `$de**`.

Emulation can be achieved by either:

- connecting a real UNIX serial device;
- dumping to a file;
- piping through a process.

It is possible to define up to four UNIX serial devices, and then decide which interface should be connected to which device. This is done by so-called *rs232 device file names*; an rs232 device file name can be either a simple path, or a command name preceeded by a pipe symbol '`|`'. If the path specifies a special device (e.g. '`/dev/ttyS0`') it is recognized by VICE and the emulator can set the baudrate.

For example, rs232 device '`filename`' will cause the output to be written (not appended) to the file '`filename`', while printer device '`|lpr`' will cause the `lpr` command to be executed and be fed the rs232 output. The rs232 output will not be converted but saved as sent by the emulated machine. The same holds true for the rs232 input. If the command writes data to the standard output it will be caught by VICE and sent back to the emulator. Also the data sent by the pseudo device will be sent back to VICE.

For example you can setup a null-modem cable between two serial ports of your PC, setup one port for login and use the other in VICE. Then you can login from your emulator via the RS232 emulation and the null-modem cable to your machine again.

You can not simply run a shell from VICE, as the shell will notice that it does not run on its own pseudo terminal and will thus buffer its output. You need to write some program that opens an own pseudo terminal and runs the shell from there (not yet finished).

Up to four RS232 devices may be specified through the following resources:

- device 1, whose default value is `/dev/ttyS0`;
- device 2, whose default value is `/dev/ttyS1`;
- device 3, whose default value is `rs232.dump`;
- device 4, whose default value is `|lpr`.

For the first two devices you can change the baudrate the tty device is set to by specifying it on the commandline or in the menu. This baudrate is 9600 by default for the latter two, but can be changed only by resources (The baudrate is independent from the baudrate the emulator actually expects).

6.11.1 RS232 resources

`RsDevice1`
`RsDevice2`
`RsDevice3`
`RsDevice4`
: Strings specifying the RS232 devices (see Section 6.11 [RS232 settings], page 42).

`RsDevice1Baud`
`RsDevice2Baud`
`RsDevice3Baud`
`RsDevice4Baud`
: Integer specifying the RS232 baudrate devices if the device file points to a special device (like '`/dev/ttyS0`'; see Section 6.11 [RS232 settings], page 42).

`Acia1Dev` : Integer (ranging from 0 to 3, for device 1-4) specifying what RS232 device (see Section 6.11 [RS232 settings], page 42) the ACIA is using (all except VIC20).

`Acia1Irq` : Integer specifying which interrupt to use. 0 = none, 1 = IRQ, 2 = NMI (C64 and C128 only)

`RsUserEnable`
: Boolean specifying if the user-port RS232 interface is being emulated (C64, C128 and VIC20).

`RsUserBaud`
: Integer specifying the baudrate of the user-port RS232 interface (C64, C128 and VIC20).

`RsUserDev`
: Integer (ranging from 0 to 3, for device 1-4) specifying what RS232 device the user-port interface is using (C64, C128 and VIC20).

6.11.2 RS232 command-line options

`-rsdev1 NAME`
`-rsdev2 NAME`
`-rsdev3 NAME`
`-rsdev4 NAME`
: Specify `NAME` as RS232 devices 1, 2, 3 and 4, respectively (`RsDevice1`, `RsDevice2` `RsDevice3` and `RsDevice4`).

`-rsdev1 BAUDRATE`
`-rsdev2 BAUDRATE`
`-rsdev3 BAUDRATE`
`-rsdev4 BAUDRATE`
: Specify `BAUDRATE` as baudrate for the RS232 devices if the device name specifies a special device (like '`/dev/ttyS0`' for example, see Section 6.11 [RS232 settings], page 42; `RsDevice1Baud`, `RsDevice2Baud` `RsDevice3Baud` and `RsDevice4Baud`).

`-myaciadev <0-3>`
: Specify RS232 device the ACIA should work on

`-rsuser`
`+rsuser`
: Enable or disable emulation of the userport RS232 emulation (`RsUser`; C64, C128 and VIC20)

`-rsuserbaud <baud>`
: Set the baud rate of the RS232 userport emulation.

`-rsuserdev <0-3>`
: Specify device for the userport RS232 emulation (`RsUserDev`; C64, C128 and VIC20).

6.11.3 RS232 usage example

Here we give you a simple example how to set up an emulated C64 using the modem connected to your PC. The following list shows each step.

`Attach your modem to your PC at a serial port.`
: Normally you should set it up to use the modem as "/dev/modem".

`start VICE`
`Setup VICE to use your modem as "serial device 1"`
: Go to the RS232 settings menu and change "Serial 1 device" to "/dev/modem" (or the device where you attached your modem to) Then go to the RS232 settings menu and change "Serial 1 baudrate" to the baudrate your modem should run at. Watch out, e.g. on Linux there is an additional multiplier to multiply with the baudrate (so e.g. 19200 gives 115200 or so baud) See the "setserial" manpage on Linux for example. However, most modems should be able to autodetect the speed to the computer as well.

`Select the RS232 emulation your programs use`
: If you want to use the Userport emulation, go to the RS232 settings and change "Userport RS232 Device" to "Serial 1". If you want ACIA emulation (swiftlink or what's it called?) then change "ACIA $DE** device" to "Serial 1".

`Enable the emulation`
: Go to the RS232 settings and select either "ACIA $DE** emulation" or Userport 300/1200 baud or CIA 9600 baud emulation.

`Load your program and start it.`
: If it is able to detect an RS232 cartridge like swiftlink or so, try to detect the ACIA emulation if enabled. Otherwise just set the baudrate to either 300, 1200 or 9600 according to what you enabled in the VICE menu for the userport.

6.12 Monitor settings

This section lists command-line options specific to the built-in monitor.

6.12.1 Monitor command-line options

`-moncommands FILENAME`
: Execute the commands from the file `FILENAME` in the monitor after starting up. This command line switch is mainly thought to load labels and to set breakpoints. Not all other commands are useful to be executed in this way, some may even lead to strange effects.

`-initbreak <address>`
: Set an initial breakpoint for the monitor. Addresses with prefix "0x" are hexadecimal.

`-remotemonitor`
`+remotemonitor`
: Enable/Disable remote monitor

`-remotemonitoraddress <name>`
: The local address the remote monitor should bind to

6.13 Machine settings

6.13.1 Machine command-line options

`-pal` Use PAL sync factor

`-ntsc` Use NTSC sync factor

`-ntscold` Use old NTSC sync factor

`-paln` Use PAL-N sync factor

6.14 Memory settings

6.14.1 Memory command-line options

`-raminitstartvalue <value>`
: Set the value for the very first RAM address after powerup

`-raminitvalueinvert <num of bytes>`
: Length of memory block initialized with the same value

`-raminitpatterninvert <num of bytes>`
: Length of memory block initialized with the same pattern

6.15 Miscellaneous settings

This section lists generic resources that do not fit in the other categories.

6.15.1 Miscellaneous resources

`Directory`
: String specifying the search path for system files. It is defined as a sequence of directory names, separated by colons (‘`:`’), just like the `PATH` variable in the shell. The special string ‘`$$`’ stands for the default search path, which is initialized at startup to the following value:

```
LIBDIR/EMUID:$HOME/.vice/EMUID:BOOTPATH/EMUID:LIBDIR/DRIVES:$HOME/.vice/DRIV
```

where:

- `LIBDIR` is the VICE installation directory (usually '`/usr/local/lib/vice`', '`/usr/lib/vice`' or '`/opt/vice/lib`');
- `EMUID` is the emulation identification string (`C64`, `C128`, `VIC20` or `PET`);
- `BOOTPATH` is the directory where the binary lies (usually '`/usr/local/bin`', '`/usr/bin`' or `/opt/vice/bin`).
- `DRIVES` is the directory called "DRIVES", where the disk drive ROMs are. (The disk drive ROMs are used by all emulators, so there is an extra directory for them.)

Notice that the middle entry points to a default location in the user's home directory. Here private ROM versions (e.g. speeddos or JiffyDos) can be stored for example.

See Chapter 4 [System files], page 16. for a description of the method used to load the emulator's system files.

`HTMLBrowserCommand`
: String specifying the command to run the help browser. The help browser can be any HTML browser, and every '`%s`' in the string is replaced with the name of the toplevel file of the VICE documentation. For example, the default value '`netscape %s`' runs Netscape Navigator.

`SaveResourcesOnExit`
: Boolean specifying whether the emulator should save changed settings before exiting. If this is enabled, the user will be always prompted first, in case the settings have changed.

`DoCoreDump`
: Boolean specifying whether the emulator should dump core when it gets a signal.

`JoyDevice1`
`JoyDevice2`
: Integer specifying which joystick device the emulator should use for joystick emulation for ports 1 and 2, respectively (0=None, 1=Numpad, 2=Custom keys, 3=Analog joystick 1, 4=Analog joystick 2, 5=Digital joystick 1, 6=Digital joystick 2 on Unix) The available joysticks might differ depending on operating system and joystick support in the OS (Linux joystick module must be available for example).

6.15.2 Miscellaneous command-line options

`-directory SEARCHPATH`
: Specify the system file search path (`Directory`).

`-htmlbrowser COMMAND`
: Specify the command to run the HTML browser for the on-line help (`HTMLBrowserCommand`).

`-saveres`
`+saveres` Enable/disable automatic saving of settings on exit (`SaveResourcesOnExit=1`, `SaveResourcesOnExit=0`).

`-confirmexit`
Confirm quiting VICE

`+confirmexit`
Never confirm quiting VICE

`-core`
`+core` Enable/disable generation of core dumps (`DoCoreDump=1`, `DoCoreDump=0`).

7 Machine-specific features

7.1 C64/128-specific commands and settings

This section lists the settings and commands that are C64/128 specific and thus are not present in the other emulators.

7.1.1 Using cartridges

The cartridge system is organized in "Slots" to allow more than one cartridge connected at a time, like it can be done using an expansion port expander on a real C64 (see below).

Generally a cartridge can be enabled by attaching its respective cartridge image, or using the respective menu option for cartridges that do not require an image.

x64, x64sc and x128 allow you to attach the following kinds of images:

- '`.crt`' images, as used by the CCS64 emulator by Per Hkan Sundell
- raw '`.bin`' images, with or without load address

Cartridge images are like disk images, but contain the contents of cartridge ROM and/or RAM images instead of disk images.

To attach cartridges, use the "Attach a cartridge image" submenu. When using '`.crt`' images, this will work for every cartridge which is supported. For raw '`.bin`' images you might have to use command line options.

When you have successfully attached a cartridge image, you should then reset the machine to make sure the cartridge initializes itself. (Or enable the "reset on cartridge change" option).

Of course, it is also possible to detach a currently attached cartridge image ("Detach cartridge image").

If you are using a freezer cart like an Action Replay cartridge, you can emulate the cartridge's freeze button with the "Cartridge freeze" command.

The imaginary expansion port expander is organized in 4 slots, the cartridges are associated with them like this:

7.1.1.1 Slot 0

All carts that have a passthrough connector go here. Once a "Slot 0" cartridge is enabled all further cartridges are connected to its respective passthrough port.

Only one cartridge of this type can be active at a time.

"Slot 0" carts have individual "enable" switches, enabling means enabling permanently.

The following cartridges are emulated in this slot:

- IEEE-488 Interface (`http://www.funet.fi/pub/cbm/schematics/cartridges/c64/ieee-488/eprom.bin`)
- Magic Voice
- MMC64

7.1.1.2 Slot 1

Mostly RAM based cartridges which for one reason or the other might make sense to be enabled together with one of the "Main Slot" cartridges go here.

Only one cartridge of this type can be active at a time.

"Slot 1" carts have individual "enable" switches, enabling means enabling permanently

The following cartridges are emulated in this slot:

- Double Quick Brown Box (DQBB)
- Expert Cartridge
- ISEPIC
- RamCart

7.1.1.3 Main Slot

All other cartridges which are not pure i/o extensions go here.

Only one cartridge of this type can be active at a time.

Cartridges in the "Main Slot" must be explicitly set as default to enable them permanently.

The following cartridges are emulated in this slot:

- generic 4K, 8K and 16K game- and ultimax cartridges
- Action Replay V5
- Action Replay MK2
- Action Replay MK3
- Action Replay MK4
- Atomic Power
- C64 Games System
- Capture
- Comal 80
- Dela EP64
- Dela EP7x8
- Dela EP256

- Diashow-Maker
- Dinamic
- EasyFlash
- Epyx FastLoad
- EXOS
- The Final Cartridge
- The Final Cartridge III
- Final Cartridge Plus
- Freeze Frame
- Freeze Machine
- Fun Play
- Game Killer
- IDE64 (`http://www.ide64.org/`)
- KCS Power Cartridge
- MACH 5
- Magic Desk
- Magic Formel
- Mikro Assembler
- MMC Replay
- Ocean
- Prophet64
- REX 256k EPROM Cart
- REX Utility
- Retro Replay
- ROSS
- Simons' BASIC
- Snapshot 64
- Stardos
- Structured BASIC
- Super Explode V5.0
- Super Games
- Super Snapshot V4
- Super Snapshot V5
- Warp Speed
- Westermann Learning
- Zaxxon

7.1.1.4 I/O Slot

All carts that are pure I/O extensions go here.

Any number of "I/O Slot" Carts may be active at a time.

"I/O Slot" carts have individual "enable" switches, enabling means enabling permanently.

The following cartridges are emulated in this slot:

- ACIA (Swiftlink, Turbo232)
- DigiMAX
- Ethernet (The Final Ethernet, RR-Net)
- GEO-RAM
- MIDI (Passport/Syntech, Datel/Siel/JMS/C-Lab, Maplin, Namesoft, Sequential)
- RAM Expansion Module (REU)
- SFX Sound Expander
- SFX Sound Sampler

7.1.1.5 Expected behaviour

When the emulator is run without arguments, all settings from the config file should be applied and arguments override settings from the config file.

When saving the settings to the config file it is expected that on the next run of the emulator all settings will be in the same state as they were when saved.

There is an exception to this rule: the cartridge in the "Main Slot" must be explicitly set as default before it gets saved to the config file.

`+cart` should disable ALL cartridges, including eventually activated REU, Swithlink and all similar expansionport devices.

`-cartXYZ` options should generally attach AND activate a cart of type XYZ. As a consequence, attaching carts this way which are NOT in the "Main Slot" will also enable the cart permanently.

7.1.1.6 Common problems

If attaching a cartridge does not work as expected, this may be because of various reasons:

- Not seldomly the CRT type is incorrectly set in `.crt` files found "in the wild". Make sure this is not the case (if in doubt use `cartconv` to verify and/or fix).
- You may have unintentionally enabled more than one cartridge at once, for example by saving the settings with REU enabled, and then later attaching a game cartridge from the command-line. The cartridge system will allow certain combinations, but (as on the real thing) not all do (can) actually work. To make sure this is not the case, either detach all cartridges from the menus, or use `+cart` on the command-line.
- The cartridge image might be broken. Try one from a different source. If you are sure the dump is ok (for example because you dumped it yourself) then make sure it is in proper linear order (on some cartridges, for example "capture", address and/or data lines at the eprom are shuffled around so a dump made with an eprom burner can not be used as is).

- Last not least you might have encountered a bug in the emulation. If you suspect this is the case, and you can still reproduce the bug after checking the things above, please file a bug report including the following information:
  - attach your vicerc and a reference to the cartridge binaries
  - if you can, comment in the respective DEBUGXYZ macros prominently defined at the top of these files: src/c64/cart/c64cart.c src/c64/cart/c64cartmem.c src/c64/c64io.c src/c64/c64export.c and then recompile. this will add debug output that might make it much easier to locate certain problems.

7.1.1.7 IEEE-488 interface

To be able to use an IEEE drive, you need to enable IEEE emulation for the emulator. To do this, follow the following steps:

Download the IEEE 488 ROM image from the CBM archives (formerly known as FU-NET) Attach that image with File/Attach cartridge image/IEEE488 interface image.

Make sure you have a one-drive system only (that is, go to Settings/Peripheral Setting, uncheck "use IEC device" for all devices, go to Settings/Drive Settings and select "Floppy type" as "none" for all drives other than drive 8.

After this, all drives can be selected in x64 and x128.

7.1.1.8 The Final Cartridge 3

The Final Cartridge 3 detects wether a mouse is connected when it starts and disables mouse support if it doesnt detect one. So to make mouse emulation work you must either enable it on the command line, or reset the cartridge after enabling it from the user interface.

7.1.2 C64 cartridge settings

7.1.2.1 C64 cartridge resources

`CartridgeReset`
`CartridgeType`
`CartridgeFile`
`DQBB` Boolean specifying whether the Double Quick Brown Box should be emulated or not.

`DQBBfilename`
`DQBBImageWrite`
`ExpertCartridgeEnabled`
Boolean specifying whether the Expert Cartridge should be emulated or not.

`Expertfilename`
`ExpertImageWrite`
`ExpertCartridgeMode`
`IDE64Config`
String encoded content of IDE64 DS1302 RAM, used to store IDEDOS setup parameters. Not meant to be directly manipulated as content depends on the version of IDEDOS used.

`IDE64version4`
: Boolean specifying whether the emulated card version is V4.1 or V3.4. This is automatically detected most of the time for .crt cartridge images.

`IDE64RTCOffset`
: Integer in seconds which gives the difference between the local time and the time of the emulated DS1302 RTC.

`IDE64Image1`
`IDE64Image2`
`IDE64Image3`
`IDE64Image4`
: String specifying the full path to the four harddisk images. If a file is non-existing the drive is not emulated. Some older IDEDOS versions only support the first two harddisks.

`IDE64Cylinders1`
`IDE64Cylinders2`
`IDE64Cylinders3`
`IDE64Cylinders4`
: Number of cylinders for the four harddisk images. Valid range is 1–65535.

`IDE64Heads1`
`IDE64Heads2`
`IDE64Heads3`
`IDE64Heads4`
: Number of heads for the four harddisk images. Valid range is 1–16.

`IDE64Sectors1`
`IDE64Sectors2`
`IDE64Sectors3`
`IDE64Sectors4`
: Number of sectors for the four harddisk images. Valid range is 1–63.

`IDE64AutodetectSize1`
`IDE64AutodetectSize2`
`IDE64AutodetectSize3`
`IDE64AutodetectSize4`
: Boolean specifying whether the disk geometry should be auto detected based on the disk image for the respective harddisk, or the cylinder/head/sector resources above should be used.

`IEEE488` Boolean specifying whether the IEEE488 interface should be emulated or not.

`IEEE488Image`
`IsepicCartridgeEnabled`
: Boolean specifying whether ISEPIC should be emulated or not.

`Isepicfilename`
`IsepicSwitch`
`IsepicImageWrite`
`MagicVoiceCartridgeEnabled`
: Boolean specifying whether the Magic Voice should be emulated or not.

`MagicVoiceImage`
`MMC64` Boolean specifying whether the MMC64 should be emulated or not.

`MMC64BIOSfilename`
`MMC64_bios_write`
`MMC64_flashjumper`
`MMC64_revision`
`MMC64imagefilename`
`MMC64_RO`

`MMC64_sd_type`
`MMCRCardImage`
`MMCREEPROMImage`
`MMCRRescueMode`
`MMCRImageWrite`
`MMCRCardRW`
`MMCRSDType`
`MMCREEPROMRW`
`RAMCART` Boolean specifying whether the RAMCart should be emulated or not.

`RAMCARTfilename`
`RAMCARTImageWrite`
`RAMCART_RO`
`RAMCARTsize`
`RRFlashJumper`
`RRBankJumper`
`RRBiosWrite`

7.1.2.2 C64 cartridge command-line options

`+cart` Disable all cartridges (which would eventually be enabled in the config file).

`-cartreset`
`+cartreset`
Reset/Do not reset machine if a cartridge is attached or detached

`-cart8 <name>`
Attach generic 8KB cartridge image

`-cart16 <name>`
Attach generic 16KB cartridge image

`-cartultimax <name>`
Attach generic 16kB Ultimax cartridge image

`-cartcrt <name>`
Attach CRT cartridge image

`-cartap <name>`
Attach raw 32KB Atomic Power cartridge image

`-cartar2 <name>`
Attach raw 16kB Action Replay MK2 cartridge image

`-cartar3 <name>`
Attach raw 16KB Action Replay MK3 cartridge image

`-cartar4 <name>`
Attach raw 32KB Action Replay MK4 cartridge image

`-cartar5 <name>`
Attach raw 32KB Action Replay cartridge image

`-cartcap <name>`
Attach raw 8kB Capture cartridge image

`-cartcomal <name>`
Attach raw 64kB Comal 80 cartridge image

`-cartdep256 <name>`
Attach raw Dela EP256 cartridge image

`-cartdep64 <name>`
Attach raw Dela EP64 cartridge image

`-cartdep7x8 <name>`
Attach raw Dela EP7x8 cartridge image

`-cartdin <name>`
Attach raw 128kB Dinamic cartridge image

`-cartdsm <name>`
Attach raw 8kB Diashow-Maker cartridge image

`-cartdqbb <name>`
Attach raw 16kB Double Quick Brown Box cartridge image

`-dqbb`
`+dqbb` Enable/Disable Double Quick Brown Box

`-dqbbimage <name>`
Specify Double Quick Brown Box filename

`-dqbbimagerw`
`+dqbbimagerw`
Allow/Disallow writing to DQBB image

`-carteasy <name>`
Attach raw EasyFlash cartridge image

`-easyflashjumper`
`+easyflashjumper`
Enable/Disable EasyFlash jumper

`-easyflashcrtwrite`
`+easyflashcrtwrite`
Allow/Disallow writing to EasyFlash .crt image

`-cartepyx <name>`
Attach raw 8KB Epyx FastLoad cartridge image

`-cartexos <name>`
Attach raw 8kB EXOS cartridge image

`-cartexpert <name>`
Attach raw 8kB Expert Cartridge image

`-expert`
`+expert` Enable/Disable the Expert Cartridge

`-expertimagename <name>`
Set Expert Cartridge image name

`-expertimagerw`
`+expertimagerw`
Allow/Disallow writing to Expert Cartridge image

`-cartfc1 <name>`
Attach raw 16kB Final Cartridge image

`-cartfc3 <name>`
Attach raw 64kB Final Cartridge III image

`-cartfcplus <name>`
Attach raw 32kB Final Cartridge Plus image

`-cartff <name>`
Attach raw 8kB Freeze Frame image

`-cartfm <name>`
Attach raw 32kB Freeze Machine image

`-cartfp <name>`
Attach raw 128kB Fun Play/Power Play cartridge image

`-cartgk <name>`
Attach raw 8KB Game Killer cartridge image

`-cartgs <name>`
Attach raw 512kB Game System cartridge image

`-cartide64 <name>`
Attach raw 64KB or 128KB IDE64 cartridge image

`-IDE64image1 <name>`
`-IDE64image2 <name>`
`-IDE64image3 <name>`
`-IDE64image4 <name>`
Specify path to the image files for IDE64 harddisks

`-IDE64cyl1 <value>`
`-IDE64cyl2 <value>`
`-IDE64cyl3 <value>`
`-IDE64cyl4 <value>`
Set number of cylinders for the IDE64 harddisk emulation (1-65535)

`-IDE64hds1 <value>`
`-IDE64hds2 <value>`
`-IDE64hds3 <value>`
`-IDE64hds4 <value>`
Set number of heads for the IDE64 harddisk emulation (1-16)

`-IDE64sec1 <value>`
`-IDE64sec2 <value>`
`-IDE64sec3 <value>`
`-IDE64sec4 <value>`
Set number of sectors for the IDE64 harddisk emulation (1-63)

`-IDE64autosize1`
`+IDE64autosize1`
`-IDE64autosize2`
`+IDE64autosize2`
`-IDE64autosize3`
`+IDE64autosize3`
`-IDE64autosize4`
`+IDE64autosize4`
Autodetect geometry of formatted images or do not autodetect and use specified geometry

`-IDE64version4`
`+IDE64version4`
Emulate version 4 hardware/Emulate pre version 4 hardware

`-cartieee <name>`
Attach CBM IEEE-488 cartridge image

`-ieee488`
`+ieee488` Enable (`IEEE488=1`) or disable (`IEEE488=0`) emulation of the IEEE488 interface.

`-ieee488image <name>`
Set IEEE488 interface image name

`-isepic`
`+isepic` Enable/Disable the ISEPIC cart

`-cartisepic <name>`
Attach raw 2kB ISEPIC cartridge image

`-isepicimagename <name>`
Set ISEPIC image name

`-isepicimagerw`
`+isepicimagerw`
Allow/Disallow writing to ISEPIC image

`-cartkcs <name>`
Attach raw 16kB KCS Power cartridge image

`-cartks <name>`
Attach raw 24kB Kingsoft cartridge image

`-cartmach5 <name>`
Attach raw 8kB MACH 5 cartridge image

`-cartmd <name>`
Attach raw 32/64/128kB Magic Desk cartridge image

`-cartmf <name>`
Attach raw Magic Formel cartridge image

`-cartmikro <name>`
Attach raw 8kB Mikro Assembler cartridge image

`-mmc64`
`+mmc64` Enable/Disable the MMC64 expansion

`-cartmmc64 <name>`
Attach raw 8kB MMC64 cartridge image

`-mmc64bios <name>`
Specify name of MMC64 BIOS image

`-mmc64image <name>`
Specify name of MMC64 image

`-mmc64readonly`
Set the MMC64 card to read-only

`-mmc64readwrite`
Set the MMC64 card to read/write

`-mmc64bioswrite`
Save the MMC64 bios when changed

`-cartmmcr <name>`
Attach raw 512kB MMC Replay cartridge image

`-mmcrrescue`
`+mmcrrescue`
Enable/Disable MMC Replay rescue mode

`-mmcrimagerw`
`+mmcrimagerw`
Allow/Disallow writing to MMC Replay image

`-mmcrcardimage <filename>`
Specify MMC Replay card image filename

`-mmcrcardrw`
`+mmcrcardrw`
Allow/Disallow writes to MMC Replay card image

`-mmcreepromimage`
Specify MMC Replay EEPROM image filename

`-mmcreepromrw`
`+mmcreepromrw`
Allow/Disallow writes to MMC Replay EEPROM image

`-cartmv <name>`
Attach raw 16kB Magic Voice cartridge image

`-cartocean <name>`
Attach raw Ocean cartridge image

`-cartp64 <name>`
Attach raw 256KB Prophet 64 cartridge image

`-cartpf <name>`
Attach raw 64kb Pagefox cartridge image

`-cartramcart <name>`
Attach raw RamCart cartridge image

`-ramcart`
`+ramcart` Enable/Disable the RAMCART expansion

`-ramcartsize <size in KB>`
Size of the RAMCART expansion

`-ramcartimage <name>`
Specify name of RAMCART image

`-ramcartimagerw`
`+ramcartimagerw`
Allow/Disallow writing to RAMCart image

`-cartrep256 <name>`
Attach raw REX EP256 cartridge image

`-cartross <name>`
Attach raw 16/32kB ROSS cartridge image

`-cartrr <name>`
Attach raw 64KB Retro Replay cartridge image

`-rrbioswrite`
`+rrbioswrite`
Enable/Disable saving of the RR ROM at exit

`-rrbankjumper`
`+rrbankjumper`
Set/Unset RR Bank Jumper

`-rrflashjumper`
`+rrflashjumper`
Set/Unset RR Flash Jumper

`-cartru <name>`
Attach raw 8kB REX Utility cartridge image

`-carts64 <name>`
Attach raw 4kB Snapshot 64 cartridge image

`-cartsb <name>`
Attach raw Structured Basic cartridge image

`-cartse5 <name>`
Attach raw 16kB Super Explode V5 cartridge image

`-cartsg <name>`
Attach raw 64kB Super Games cartridge image

`-cartsimon <name>`
Attach raw 16kB Simons Basic cartridge image

`-cartss4 <name>`
Attach raw 32KB Super Snapshot V4 cartridge image

`-cartss5 <name>`
Attach raw 64KB Super Snapshot V5 cartridge image

`-cartstar <name>`
Attach raw 16KB Stardos cartridge image

`-cartwl <name>`
Attach raw 16KB Westermann Learning cartridge image

`-cartws <name>`
Attach raw 8kB Warp Speed cartridge image

`-cartzaxxon <name>`
Attach raw 16kB Zaxxon cartridge image

7.1.3 CIA settings

7.1.3.1 CIA command-line options

`-ciamodel <model>`
Set both CIA models (0 = old 6526, 1 = new 6526A)

`-cia1model <model>`
Set CIA 1 model (0 = old 6526, 1 = new 6526A)

`-cia2model <model>`
Set CIA 2 model (0 = old 6526, 1 = new 6526A)

7.1.4 VIC-II settings

These settings control the emulation of the VIC-II (MOS6569) video chip used in both the C64 and the C128.

- "Sprite-sprite collisions" and "Sprite-background collisions", if enabled, cause the hardware detection of sprite-to-sprite and sprite-to-background collisions of the VIC-II to be emulated. This feature is used by many games, and disabling either of the two detection systems can sometimes make you invincible (although there is also a chance that also enemies become invincible then).

- "Color set" can be used to dynamically change the palette file being used by choosing one of the available predefined color sets:
  - '`default.vpl`' ("default"), the default VICE palette;
  - '`c64s.vpl`' ("C64S"), palette taken from the shareware C64S emulator by Miha Peternel.
  - '`ccs64.vpl`' ("CCS64"), palette taken from the shareware CCS64 emulator by Per Hkan Sundell.
  - '`frodo.vpl`' ("Frodo"), palette taken from the free Frodo emulator by Christian Bauer (`http://www.uni-mainz.de/~bauec002/FRMain.html`).
  - '`pc64.vpl`' ("PC64"), palette taken from the free PC64 emulator by Wolfgang Lorenz.
  - '`godot.vpl`' ("GoDot"), palette as suggested by the authors of the C64 graphics package GoDot (`http://users.aol.com/howtogodot/welcome.htm`).

7.1.4.1 VIC-II resources

`VICIICheckSsColl`
: Boolean specifying whether the sprite-sprite hardware collision detection must be emulated.

`VICIICheckSbColl`
: Boolean specifying whether the sprite-background hardware collision detection must be emulated.

`VICIIVideoCache`
: Boolean specifying whether the video cache is turned on.

`VICIIDoubleSize`
: Boolean specifying whether double-size mode is turned on.

`VICIIDoubleScan`
: Boolean specifying whether double-scan mode is turned on.

`VICIIPaletteFile`
: String specifying the name of the palette file being used. The '`.vpl`' extension is optional.

7.1.4.2 VIC-II command-line options

`-VICIIcheckss`
`+VICIIcheckss`
: Enable (`VICIICheckSsColl=1`) and disable (`VICIICheckSsColl=0`) emulation of hardware sprite-sprite collision detection, respectively.

`-VICIIchecksb`
`+VICIIchecksb`
: Enable (`VICIICheckSbColl=1`) and disable (`VICIICheckSbColl=0`) emulation of hardware sprite-background collision detection, respectively.

`-VICIIvcache`
`+VICIIvcache`
: Enable/disable the video cache (`VICIIVideoCache=1`, `VICIIVideoCache=0`).

`-VICIIdsize`
`+VICIIdsize`
: Enable/disable the double size mode (`VICIIDoubleSize=1`, `VICIIDoubleSize=0`).

`-VICIIdscan`
`+VICIIdscan`
: Enable/disable the double scan mode (`VICIIDoubleScan=1`, `VICIIDoubleScan=0`).

`-VICIIhwscale`
`+VICIIhwscale`
: Enable/Disable hardware scaling

`-VICIIscale2x`
`+VICIIscale2x`
: Enable/Disable Scale2x

`-VICIIintpal`
: Use an internal calculated palette

`-VICIIextpal`
: Use an external palette (file)

`-VICIIpalette NAME`
: Specify `NAME` as the palette file (`VICIIPaletteFile`).

`-VICIIfulldevice <device>`
: Select fullscreen device

`-VICIIXRANDRfullmode <mode>`
: Select fullscreen mode

`-VICIIVidmodefullmode <mode>`
: Select fullscreen mode

`-VICIIborders <mode>`
: Set VIC-II border display mode (0: normal, 1: full, 2: debug)

`-VICIImodel <model>`
: Set VIC-II model (6569/6569r1/8565/6567/8562/6567r56a). This setting is only available in `x64sc`.

`-newluminance`
`+newluminance`
: Enable/Disable new luminances.

`-saturation <0-2000>`
: Set saturation of internal calculated palette [1000]

`-contrast <0-2000>`
: Set contrast of internal calculated palette [1000]

`-brightness <0-2000>`
: Set brightness of internal calculated palette [1000]

`-gamma <0-4000>`
Set gamma of internal calculated palette [2200]

`-tint <0-2000>`
Set tint of internal calculated palette [1000]

`-oddlinesphase <0-2000>`
Set phase for color carrier in odd lines [1250]

`-oddlinesoffset <0-2000>`
Set phase offset for color carrier in odd lines [750]

`-crtblur <0-1000>`
Amount of horizontal blur for the CRT emulation. [500]

`-crtscanlineshade <0-1000>`
Amount of scan line shading for the CRT emulation [667]

7.1.5 SID settings

These settings control the emulation of the SID (MOS6581 or MOS8580) audio chip.

- "Second SID" maps a second SID chip into the address space for stereo sound. This emulates e.g. the "SID Symphony Stereo Cartridge" from Dr. Evil Laboratories. The second SID can be used with software such as "Stereo SID Player" by Mark Dickenson or "The Enhanced Sidplayer" by Craig Chamberlain.
- "Second SID base address" sets the start address for the second SID chip. Software normally uses $DE00 or $DF00, since $DE00-$DEFF and $DF00-$DFFF can be mapped through the cartridge port of the C64. The default start address is $DE00.
- "Emulate filters" causes the built-in programmable filters of the SID chip to be emulated. A lot of C64 music requires them to be emulated properly, but their emulation requires some additional processor power.
- "ChipModel" specifies the model of the SID chip being emulated: there are two slightly different generations of SID chips: MOS6581 ones and MOS8580 ones.
- "Use reSID emulation" specifies whether the more accurate (and resource hungry) reSID emulation is turned on or off.
- "reSID sampling method" selects the method for conversion of the SID output signal to a sampling rate appropriate for playback by standard digital sound equipment. Possible settings are:
  - "Fast" simply clocks the SID chip at the output sampling frequency, picking the nearest sample. This yields acceptable sound quality, but sampling noise is noticeable in some cases, especially with SID combined waveforms. The sound emulation is still cycle exact.
  - "Interpolating" clocks the SID chip each cycle, and calculates each sample with linear interpolation. The sampling noise is now strongly attenuated by the SID external filter (as long as "Emulate filters" is selected), and the linear interpolation further improves the sound quality.
  - "Resampling" clocks the SID chip each cycle, and uses the theoretically correct method for sample generation. This delivers CD quality sound, but is extremely

CPU intensive, and is thus most useful for non-interactive sound generation. Unless you have a very fast machine, that is.

- "reSID resampling passband" specifies the percentage of the total bandwidth allocated to the resampling filter passband. The work rate of the resampling filter is inversely proportional to the remaining transition band percentage. This implies that e.g. with the transition band starting at ~ 20kHz, it is faster to generate 48kHz than 44.1kHz samples. For CD quality sound generation at 44.1kHz the passband percentage should be set to 90 (i.e. the transition band starting at almost 20kHz).

7.1.5.1 SID resources

`SidStereo`
: Boolean selecting emulation of a second SID.

`SidStereoAddressStart`
: Integer specifying the start address for the second SID.

`SidFilters`
: Boolean specifying whether the built-in SID filters must be emulated.

`SidModel` Integer specifying what model of the SID must be emulated (`0`: MOS6581, `1`: MOS8580).

`SidEngine`
`SidResidSampling`
: Integer specifying the sampling method (`0`: Fast, `1`: Interpolation, `2`: Resampling)

`SidResidPassband`
: Integer specifying the resampling filter passband in percentage of the total bandwidth (`0 - 90`).

7.1.5.2 SID command-line options

`-sidstereo`
: Emulates a second SID chip for stereo sound (`SidStereo`).

`-sidstereoaddress ADDRESS`
: Specifies the start address for the second SID chip (`SidStereoAddressStart`).

`-sidenginemodel <engine and model>`
: Specify `SID engine` and `MODEL` for the emulated SID chip (0: FastSID 6581, 1: FastSID 8580, 256: ReSID 6581, 257: ReSID 8580, 258: ReSID 8580 + digiboost, 1024: ParSID in par port 1, 1280: ParSID in par port 2, 1536: ParSID in par port 3, 1800: ReSID-FP 6581R3 4885, 1801: ReSID-FP 6581R3 0486S, 1802: ReSID-FP 6581R3 3984, 1803: ReSID-FP 6581R4 AR 3789, 1804: ReSID-FP 6581R3 4485, 1805: ReSID-FP 6581R4 1986S, 1808: ReSID-FP 8580R5 3691, 1809: ReSID-FP 8580R5 3691 + digiboost, 1810: ReSID-FP 8580R5 1489, 1811: ReSID-FP 8580R5 1489D).

`-sidfilters`
`+sidfilters`
: Enable (`SidFilters=1`) or disable (`SidFilters=0`) emulation of the built-in SID filters.

`-residsamp METHOD`
: Specifies the sampling method; fast (`SidResidSampling=0`), interpolating (`SidResidSampling=1`), resampling (`SidResidSampling=2`), fast resampling (`SidResidSampling=3`).

`-residpass PERCENTAGE`
: Specifies the resampling filter passband in percentage of the total bandwidth (`SidResidPassband=0-90`).

`-residgain PERCENTAGE`
: Specifies reSID gain in percent (90 - 100).

`-residfilterbias <number>`
: reSID filter bias setting, which can be used to adjust DAC bias in millivolts.

7.1.6 C64 I/O extension settings

I/O extensions are (usually) cartridges which do not map into ROM space, but use only the I/O space at address range $DE00 . . . $DEFF and/or $DF00 . . . $DFFF.

Please use these extensions only when needed, as they might cause compatibility problems.

The following I/O extensions are available:

- ACIA (Swiftlink, Turbo232)
- DigiMAX
- Ethernet (The Final Ethernet, RR-Net)
- GEO-RAM
- MIDI (Passport, Datel, Maplin, Namesoft, Sequential)
- REU - The “RAM Expansion Module” extension emulates a standard Commodore RAM Expansion Unit; this can be used with GEOS and other programs that are designed to take advantage of it. This currently works only in the C64 emulator.
- SFX Sound Expander
- SFX Sound Sampler

7.1.6.1 C64 I/O extension resources

TODO

`Acia1Enable`
: Boolean specifying whether the ACIA (Swiftlink, Turbo232) cartridge should be emulated or not.

`DIGIMAX` Boolean specifying whether the DigiMAX cartridge should be emulated or not.

`DIGIMAXbase`
`ETHERNET_INTERFACE`
`ETHERNET_DISABLED`
`ETHERNET_ACTIVE`
`ETHERNET_AS_RR`
`GEORAM` Boolean specifying whether the GEO-RAM cartridge should be emulated or not.

`GEORAMfilename`
`GEORAMImageWrite`
`GEORAMsize`
`MIDIEnable`
: Boolean specifying whether the MIDI cartridge should be emulated or not.

`MIDIMode`

`REU`
: Boolean specifying whether the RAM Expansion Module should be emulated or not.

`REUfilename`
`REUImageWrite`
`REUsize`

`SFXSoundExpander`
: Boolean specifying whether the SFX Sound Expander should be emulated or not.

`SFXSoundExpanderChip`
`SFXSoundSampler`
: Boolean specifying whether the SFX Sound Sampler should be emulated or not.

7.1.6.2 C64 I/O extension command-line options

`-acia1`
`+acia1`
: Enable/Disable the $DE** ACIA RS232 interface emulation

`-digimax`
`+digimax`
: Enable/Disable the DigiMAX cartridge

`-digimaxbase <base address>`
: Base address of the DigiMAX cartridge

`-miditype <0-4>`
: MIDI interface type (0: Sequential, 1: Passport, 2: DATEL, 3: Namesoft, 4: Maplin)

`-midi`
`+midi`
: Enable/Disable MIDI emulation

`-midiin <name>`
: Specify MIDI-In device

`-midiout <name>`
: Specify MIDI-Out device

`-mididrv <driver>`
: Specify MIDI driver (0 = OSS, 1 = ALSA)

`-georam`
`+georam`
: Enable/Disable the GEORAM expansion unit

`-cartgeoram <name>`
: Attach raw GEO-RAM cartridge image

`-georamimage <name>`
Specify name of GEORAM image

`-georamimagerw`
`+georamimagerw`
Allow/Disallow writing to GEORAM image

`-georamsize <size in KB>`
Size of the GEORAM expansion unit

`-reu`
`+reu` Enable (`REU=1`) or disable (`REU=0`) emulation of the RAM Expansion Module.

`-cartreu <name>`
Attach raw REU cartridge image

`-reuimage <name>`
Specify name of REU image

`-reuimagerw`
`+reuimagerw`
Allow/Disallow writing to REU image

`-reusize <size in KB>`
Size of the RAM expansion unit

`-sfxse`
`+sfxse` Enable/Disable the SFX soundexpander cartridge

`-sfxsetype <type>`
Set YM chip type (3526 / 3812)

`-sfxss`
`+sfxss` Enable/Disable the SFX Sound Sampler cartridge

`-tfe`
`+tfe` Enable/Disable the TFE ("The Final Ethernet") unit

`-tfeif <name>`
Set the system ethernet interface for TFE emulation

`-tferrnet`
`+tferrnet`
Enable/Disable RRNet mode of TFE emulation

`-burstmod <value>`
Enable/Disable burst modification. If it's 1 the cable is connected to CIA1, if 2 then to CIA2 and 0 disables it. This is emulates the fast serial bus connection as described at http://www.cs.tut.fi/~albert/Dev/burst/, with the wire to the tape port cut.

7.1.7 C64/128 system ROM settings

These settings can be used to control what system ROMs are loaded in the C64/128 emulators at startup. They cannot be changed from the menus.

7.1.7.1 C64/128 system ROM resources

`KernalName`
: String specifying the name of the Kernal ROM (default '`kernal`').

`BasicName`
: String specifying the name of the Basic ROM (default '`basic`'). In the C128 emulator, the ROM image must actually include the editor ROM too.

`ChargenName`
: String specifying the name of the character generator ROM (default '`chargen`').

`KernalRev`
: String specifying the Kernal revision. This resource can be used to control what revision of the C64 kernal is being used; it cannot be changed at runtime. VICE is able to automatically convert one ROM revision into another, by manually patching the loaded image. This way, it is possible to use any of the ROM revisions without changing the ROM set. Valid values are:

  `0` Kernal revision 0;

  `3` Kernal revision 3;

  `sx`
  `67` Commodore SX-64 ROM;

  `100`
  `4064` Commodore 4064 (also known as "PET64" or "Educator 64") ROM.

7.1.7.2 C64/128 system ROM command-line options

`-kernal NAME`
: Specify '`NAME`' as the Kernal ROM file (`KernalName`).

`-basic NAME`
: Specify '`NAME`' as the Basic ROM file (`BasicName`).

`-chargen NAME`
: Specify '`NAME`' as the character generator ROM file (`ChargenName`).

`-kernalrev REVISION`
: Specify Kernal revision (`KernalRev`).

7.1.8 C64 settings

7.1.8.1 C64 command-line options

`-gluelogictype <type>`
: Set glue logic type (0 = discrete, 1 = 252535-01)

`-plus60k`
`+plus60k` Enable/Disable the PLUS60K RAM expansion

`-plus60kimage <name>`
: Specify name of PLUS60K image

`-plus60kbase <base address>`
Base address of the PLUS60K expansion

`-plus256k`
`+plus256k`
Enable/Disable the PLUS256K RAM expansion

`-plus256kimage <name>`
Specify name of PLUS256K image

`-256k`
`+256k` Enable/Disable the 256K RAM expansion

`-256kimage <name>`
Specify name of 256K image

`-256kbase <base address>`
Base address of the 256K expansion

7.2 C128-specific commands and settings

7.2.1 VDC settings

7.2.1.1 VDC command-line options

`-VDCvcache`
`+VDCvcache`
Enable/Disable the video cache

`-VDCdsize`
`+VDCdsize`
Enable/Disable double size

`-VDCdscan`
`+VDCdscan`
Enable/Disable double scan

`-VDChwscale`
`+VDChwscale`
Enable/Disable hardware scaling

`-VDCintpal`
Use an internal calculated palette

`-VDCextpal`
Use an external palette (file)

`-VDCpalette <name>`
Specify name of file of external palette

`-VDCfulldevice <device>`
Select fullscreen device

`-VDCXRANDRfullmode <mode>`
Select fullscreen mode

`-VDCVidmodefullmode <mode>`
Select fullscreen mode

`-VDC16KB` Set the VDC memory size to 16KB

`-VDC64KB` Set the VDC memory size to 64KB

`-VDCRevision <number>`
Set VDC revision (0..2)

`-saturation <0-2000>`
Set saturation of internal calculated palette [1000]

`-contrast <0-2000>`
Set contrast of internal calculated palette [1000]

`-brightness <0-2000>`
Set brightness of internal calculated palette [1000]

`-gamma <0-4000>`
Set gamma of internal calculated palette [2200]

`-tint <0-2000>`
Set tint of internal calculated palette [1000]

`-oddlinesphase <0-2000>`
Set phase for color carrier in odd lines [1250]

`-oddlinesoffset <0-2000>`
Set phase offset for color carrier in odd lines [750]

`-crtblur <0-1000>`
Amount of horizontal blur for the CRT emulation. [500]

`-crtscanlineshade <0-1000>`
Amount of scan line shading for the CRT emulation [667]

7.2.2 C128 system ROM settings

7.2.2.1 C128 system ROM command-line options

`-basic64 <name>`
Specify name of C64 mode BASIC ROM image

`-kernal64 <name>`
Specify name of C64 mode Kernal ROM image

`-basiclo <name>`
Specify name of BASIC ROM image (lower part)

`-basichi <name>`
Specify name of BASIC ROM image (higher part)

`-kernal <name>`
Specify name of international Kernal ROM image

`-kernalde <name>`
Specify name of German Kernal ROM image

`-kernalfi <name>`
Specify name of Finnish Kernal ROM image

`-kernalfr <name>`
Specify name of French Kernal ROM image

`-kernalit <name>`
Specify name of Italian Kernal ROM image

`-kernalno <name>`
Specify name of Norwegian Kernal ROM image

`-kernalse <name>`
Specify name of Swedish Kernal ROM image

`-chargen <name>`
Specify name of international character generator ROM image

`-chargde <name>`
Specify name of German character generator ROM image

`-chargfr <name>`
Specify name of French character generator ROM image

`-chargse <name>`
Specify name of Swedish character generator ROM image

`-intfunc`
`+intfunc` Enable/Disable the internal Function ROM

`-intfrom <name>`
Specify name of internal Function ROM image

`-extfunc`
`+extfunc` Enable/Disable the external Function ROM

`-extfrom <name>`
Specify name of external Function ROM image

7.2.3 C128 settings

7.2.3.1 C128 command-line options

`-40col` Activate 40 column mode

`-80col` Activate 80 column mode

`-go64` Always switch to C64 mode on reset

`+go64` Always switch to C128 mode on reset

7.3 C64DTV-specific commands and settings

This section lists the settings and commands that are C64DTV specific and thus are not present in the other emulators.

7.3.1 C64DTV ROM image

The DTV has a 2MB Flash chip which contains the kernal, basic and character set ROMs along with other data, such as games in the case of the original C64DTV ROM.

The image file is a dump of the flash chip. It is exactly 2MB (2097152 bytes).

If you do not have a suitable image file, an image using the C64 kernal, basic and charset is automatically created.

If writing to the C64DTV ROM is enabled, the image file is rewritten with the current data when exiting x64dtv.

Note that x64dtv tries to load the image file from the C64DTV directory first, and if it isn't found there, x64dtv tries to load it from the current directory. If you do not have '`dtvrom.bin`' in your C64DTV directory and writing to DTV ROM is enabled, the '`dtvrom.bin`' file is created to the current directory.

NOTE: The original C64DTV ROM has somewhat distorted colors, normally you should use a patched rom.

`-c64dtvromimage NAME`
: Specify '`NAME`' as the C64DTV ROM image

`-c64dtvromrw`
`+c64dtvromrw`
: Enable or disable writing to C64DTV ROM image

The trueflashfs option is analogous to True drive emulation. If disabled, any file access to the flash filesystem (device 1) will go to the local file system instead.

`-trueflashfs`
`+trueflashfs`
: Enable or disable true hardware flash file system

`-fsflash NAME`
: Specify '`NAME`' as directory for flash file system device

7.3.2 DTV revision

The DTV revision 2 has a bug in the Blitter. Using revision 3 is recommended. Emulation of DTV revision 2 including Blitter bug is intended for testing DTV software.

`-dtvrev REVISION`
: Specify DTV '`REVISION`' (2 or 3)

7.3.3 LumaFix

The PAL C64DTVs have wrong resistors in the video output circuit, which causes incorrect luminances. Several hardware solutions ("LumaFixes") have been developed to fix this flaw.

The fixed video output is emulated by selecting "New Luminances". The unmodified C64DTV video output can be emulated with "Old Luminances".

The default setting is "New Luminances".

7.3.4 Userport

The C64DTV userport emulation currently supports three devices: Hummer ADC, userport joystick and PS/2 mouse.

The joystick that controls either the Hummer ADC or userport joystick can be selected using the same parameter or menu option.

While using the Hummer ADC, joystick UP and DOWN are mapped to the Hummer buttons A and B respectively. LEFT and RIGHT set the ADCs output to 0 and 255. Centering the joystick results in the ADC value of 128.

Currently the Hummer ADC and userport joystick are mutually exclusive. This means that enabling one disables the other. PS/2 mouse emulation can be used simultaneously with either Hummer ADC or userport joystick.

`-hummeradc`
`+hummeradc`
: Enable/Disable Hummer ADC

`-ps2mouse`
`+ps2mouse`
: Enable or disable PS/2 mouse on userport

7.3.5 Debug

Debugging information on Blitter, DMA and Flash can be enabled with command line parameters. This can be useful for DTV software development.

`-dtvblitterlog`
`+dtvblitterlog`
: Enable or disable DTV Blitter log

`-dtvdmalog`
`+dtvdmalog`
: Enable or disable DTV DMA log

`-dtvflashlog`
`+dtvflashlog`
: Enable or disable DTV Flash log

7.3.6 Monitor DTV features

Currently the registers A, Y and X are registers R0, R1 and R2 regardless of the mapping, which can be seen and modified via the registers ACM and XYM.

The monitor can access all 2MB of RAM and 2MB of Flash, but only 64 kB at a time. The 64kB bank can be selected with "bank ram00".."ram1f" for RAM and "bank rom00".."rom1f" for Flash.

The "load" command can load large files (>64kB) correctly if the bank is set to "ramXX", where XX is the starting bank (usually "bank00").

7.4 VIC20-specific commands and settings

This section lists the settings and commands that are VIC20-specific and thus are not present in the other emulators.

7.4.1 Using cartridge images

As with the C64 (see Section 7.1.1 [C64 cartridges], page 47), it is possible to attach several types of cartridge images:

- 4 or 8 Kbyte cartridges located at $2000;
- 4 or 8 Kbyte cartridges located at $4000;
- 4 or 8 Kbyte cartridges located at $6000;
- 4 or 8 Kbyte cartridges located at $A000;
- 4 Kbyte cartridges located at $B000.

This can all be done via the "Attach cartridge image..." command in the left-button menu. It is also possible to let xvic "guess" the type of cartridge using "Smart-attach cartridge image...".

Notice that several cartridges are actually made up of two pieces (and two files), that need to be loaded separately at different addresses. In that case, you have to know the addresses (which are usually specified in the file name) and use the "attach" command twice.

A special kind of cartridge file is where the two files mentioned above are concatenated (with removing the two byte load address of the second image) into one 16k image. There are only few of those images, though. Normally the second part is located at $A000. Vice can now attach such concatenated files at the start address $2000, $4000, and $6000. The second half of such an image is moved to $A000. If you encounter 16k images that have the second half not at $A000 you can split the image into two halfs (i.e. one 8194 byte and one 8192 byte, because the first has the load address) and attach both files separately.

One cartridge that is currently only partially supported here is the VIC1112 IEEE488 interface. You have to load the ROM as a cartridge, but you also have to enable the IEEE488 hardware by menu.

7.4.2 VIC20 cartridge settings

7.4.2.1 VIC20 cartridge command-line options

`-cart2 <name>`
: Specify 4/8/16K extension ROM name at $2000

`-cart4 <name>`
: Specify 4/8/16K extension ROM name at $4000

`-cart6 <name>`
: Specify 4/8/16K extension ROM name at $6000

`-cartA <name>`
: Specify 4/8K extension ROM name at $A000

`-cartB <name>`
: Specify 4K extension ROM name at $B000

`-cartgeneric <name>`
: Specify generic extension ROM name

`-cartmega <name>`
: Specify Mega-Cart extension ROM name

`-mcnvramfile <name>`
: Set Mega-Cart NvRAM filename

`-mcnvramwriteback`
`+mcnvramwriteback`
: Enable/Disable Mega-Cart NvRAM writeback

`-cartfe <name>`
: Specify Final Expansion extension ROM name

`-fewriteback`
`+fewriteback`
: Enable/Disable Final Expansion write back to ROM file

`-cartfp <name>`
: Specify Vic Flash Plugin extension ROM name

`-fpwriteback`
`+fpwriteback`
: Enable/Disable Vic Flash Plugin write back to ROM file

`-ieee488`
`+ieee488` Enable/Disable VIC-1112 IEEE488 interface

`-sidcart`

`+sidcart` Enable/Disable SID Cartridge

7.4.3 VIC settings

7.4.3.1 VIC resources

`VICVideoCache`
: Boolean specifying whether the video cache is turned on.

`VICDoubleSize`
: Boolean specifying whether double-size mode is turned on.

`VICDoubleScan`
: Boolean specifying whether double-scan mode is turned on.

`VICPaletteFile`
: String specifying the name of the palette file being used. The '`.vpl`' extension is optional.

7.4.3.2 VIC command-line options

`-VICvcache`
`+VICvcache`
: Enable/disable the video cache (`VICVideoCache=1`, `VICVideoCache=0`).

`-VICdsize`
`+VICdsize`
: Enable/disable the double size mode (`VICDoubleSize=1`, `VICDoubleSize=0`).

`-VICdscan`
`+VICdscan`
: Enable/disable the double scan mode (`VICDoubleScan=1`, `VICDoubleScan=0`).

`-VIChwscale`
`+VIChwscale`
: Enable/Disable hardware scaling

`-VICscale2x`
`+VICscale2x`
: Enable/Disable Scale2x

`-VICpalette NAME`
: Specify `NAME` as the palette file (`VICPaletteFile`).

`-VICintpal`
: Use an internal calculated palette

`-VICextpal`
: Use an external palette (file)

`-VICfulldevice <device>`
: Select fullscreen device

`-VICXRANDRfullmode <mode>`
: Select fullscreen mode

`-VICVidmodefullmode <mode>`
: Select fullscreen mode

`-saturation <0-2000>`
: Set saturation of internal calculated palette [1000]

`-contrast <0-2000>`
: Set contrast of internal calculated palette [1000]

`-brightness <0-2000>`
: Set brightness of internal calculated palette [1000]

`-gamma <0-4000>`
: Set gamma of internal calculated palette [2200]

`-tint <0-2000>`
: Set tint of internal calculated palette [1000]

`-oddlinesphase <0-2000>`
: Set phase for color carrier in odd lines [1250]

`-oddlinesoffset <0-2000>`
: Set phase offset for color carrier in odd lines [750]

`-crtblur <0-1000>`
: Amount of horizontal blur for the CRT emulation. [500]

`-crtscanlineshade <0-1000>`
: Amount of scan line shading for the CRT emulation [667]

7.4.4 Changing memory configuration

It is possible to change the VIC20 memory configuration in two ways: by enabling and/or disabling certain individual memory blocks, or by choosing one among a few typical memory configurations. The former can be done by modifying resource values directly or from the right-button menu; the latter can only be done from the menu.

There are 5 RAM expansion blocks in the VIC20, numbered 0, 1, 2, 3 and 5:

- block 0 (3 Kbytes at $0400-$0FFF);
- block 1 (8 Kbytes at $2000-$3FFF);
- block 2 (8 Kbytes at $4000-$5FFF);
- block 3 (8 Kbytes at $6000-$7FFF);
- block 5 (8 Kbytes at $A000-$BFFF).

These blocks are called *expansion blocks* because they are not present a stock ("unexpanded") machine. Each of them is associated to a boolean `RamBlockX` resource (where `X` is the block number) that specifies whether the block is enabled or not.

There are also some common memory configurations you can pick from the right-button menu:

- no RAM expansion blocks at all;
- all RAM expansion blocks enabled;
- 3K expansion (only block 0 is enabled);
- 8K expansion (only block 1 is enabled);
- 16K expansion (only blocks 1 and 2 are enabled);
- 24K expansion (only blocks 1, 2 and 3 are enabled).

7.4.4.1 VIC20 memory configuration resources

`RAMBlock0`
`RAMBlock1`
`RAMBlock2`
`RAMBlock3`
`RAMBlock5`

Booleans specifying whether RAM blocks 0, 1, 2, 3 and 5 must be enabled.

7.4.4.2 VIC20 memory configuration command-line options

`-memory CONFIG`

Specify memory configuration. It must be a comma-separated list of options, each of which can be one the following:

- `none` (no extension);
- `all` (all blocks);
- `3k` (3k space in block 0);
- `8k` (first 8k extension block);
- `16k` (first and second 8k extension blocks);
- `24k` (first, second and 3rd extension blocks);

- `0`, `1`, `2`, `3`, `5` (memory in respective blocks);
- `04`, `20`, `40`, `60`, `A0` (memory at respective address.

For example,

```
xvic -memory none
```

gives an unexpanded VIC20. While

```
xvic -memory 60,a0
```

or

```
xvic -memory 3,5
```

enables memory in blocks 3 and 5, which is the usual configuration for 16k ROM modules.

7.4.5 VIC20 system ROM settings

These settings can be used to control what system ROMs are loaded in the VIC20 emulator at startup. They cannot be changed from the menus.

7.4.5.1 VIC20 system ROM resources

`KernalName`
: String specifying the name of the Kernal ROM (default '`kernal`').

`BasicName`
: String specifying the name of the Basic ROM (default '`basic`').

`ChargenName`
: String specifying the name of the character generator ROM (default '`chargen`').

`GenericCartridgeFile2000`
`GenericCartridgeFile4000`
`GenericCartridgeFile6000`
`GenericCartridgeFileA000`
`GenericCartridgeFileB000`
: String specifying the name of the respective cartridge ROM images.

7.4.5.2 VIC20 system ROM command-line options

`-kernal NAME`
: Specify '`NAME`' as the Kernal ROM file (`KernalName`).

`-basic NAME`
: Specify '`NAME`' as the Basic ROM file (`BasicName`).

`-chargen NAME`
: Specify '`NAME`' as the character generator ROM file (`ChargenName`).

`-cart2 NAME`
`-cart4 NAME`
`-cart6 NAME`
`-cartA NAME`
`-cartB NAME`
: Specify '`NAME`' as the cartridge image to attach. (`CartridgeFile2000`,...,`CartridgeFileB000`).

7.4.6 VIC20 settings

7.4.6.1 VIC20 command-line options

`-OEMjoy`

`+OEMjoy` Enable/Disable the OEM userport joystick adapter

7.5 PLUS4-specific commands and settings

7.5.1 TED settings

7.5.1.1 TED command-line options

`-TEDvcache`
`+TEDvcache`
: Enable/Disable the video cache

`-TEDdsize`
`+TEDdsize`
: Enable/Disable double size

`-TEDdscan`
`+TEDdscan`
: Enable/Disable double scan

`-TEDscale2x`
`+TEDscale2x`
: Enable/Disable Scale2x filter

`-TEDhwscale`
`+TEDhwscale`
: Enable/Disable hardware scaling

`-TEDintpal`
: Use an internal calculated palette

`-TEDextpal`
: Use an external palette (file)

`-TEDpalette <name>`
: Specify name of file of external palette

`-TEDfulldevice <device>`
: Select fullscreen device

`-TEDXRANDRfullmode <mode>`
: Select fullscreen mode

`-TEDVidmodefullmode <mode>`
: Select fullscreen mode

`-saturation <0-2000>`
: Set saturation of internal calculated palette [1000]

`-contrast <0-2000>`
Set contrast of internal calculated palette [1000]

`-brightness <0-2000>`
Set brightness of internal calculated palette [1000]

`-gamma <0-4000>`
Set gamma of internal calculated palette [2200]

`-tint <0-2000>`
Set tint of internal calculated palette [1000]

`-oddlinesphase <0-2000>`
Set phase for color carrier in odd lines [1250]

`-oddlinesoffset <0-2000>`
Set phase offset for color carrier in odd lines [750]

`-crtblur <0-1000>`
Amount of horizontal blur for the CRT emulation. [500]

`-crtscanlineshade <0-1000>`
Amount of scan line shading for the CRT emulation [667]

7.5.2 PLUS4 I/O extension settings

7.5.2.1 PLUS4 I/O extension command-line options

`-digiblaster`
`+digiblaster`
Enable/Disable the digiblaster add-on

`-sidcart`

`+sidcart` Enable/Disable SID Cartridge

`-sidcartjoy`
`+sidcartjoy`
Enable/Disable SID cartridge joystick

`-speech`
`+speech` Enable/Disable the v364 speech add-on

`-speechrom <name>`
Attach Speech ROM image

7.5.3 PLUS4 system ROM settings

7.5.3.1 PLUS4 system ROM command-line options

`-functionlo <name>`
Specify name of Function low ROM image

`-functionhi <name>`
Specify name of Function high ROM image

`-c1lo <name>`
: Specify name of Cartridge 1 low ROM image

`-c1hi <name>`
: Specify name of Cartridge 1 high ROM image

`-c2lo <name>`
: Specify name of Cartridge 2 low ROM image

`-c2hi <name>`
: Specify name of Cartridge 2 high ROM image

7.5.4 PLUS4 settings

7.5.4.1 PLUS4 command-line options

`-ramsize <ramsize>`
: Specify size of RAM installed in kb (16/32/64)

`-h256k` Enable the HANNES 256K RAM expansion

`-h1024k` Enable the HANNES 1024K RAM expansion

`-h4096k` Enable the HANNES 4096K RAM expansion

`-cs256k` Enable the CSORY 256K RAM expansion

7.6 PET-specific commands and settings

This section lists the settings and commands that are PET-specific and thus are not present in the other emulators.

7.6.1 Changing PET model settings

With `xpet`, it is possible to change at runtime the characteristics of the emulated PET so that it matches (or not) the ones of a certain PET model, and it is also possible to select from a common set of PET models so that all the features are selected accordingly.

The former is done by changing the following resources (via resource file, command line options or right-menu items):

`RamSize` Size of memory in kByte. 96k denotes a 8096, 128k a 8296.

`IOSize` Size of I/O area in Byte. Either 2048 or 256 for 8296.

`VideoSize`
: The number of columns on the screen (40 or 80). A 0 auto-detects this from the ROM.

`Ram9` The 8296 can map RAM into the address range $9***

`RamA` The 8296 can map RAM into the address range $A***

`SuperPET` This resource enables the SuperPET (MicroMainFrame 9000) I/O and disables the 8x96 mappings.

`Basic1` If (by checksum) a version 1 kernal is detected, then the kernal ROM is patched to make the IEEE488 interface work.

`Basic1Chars`
: Exchanges some character in the character ROM that have changed between the first PET 2001 and all newer versions.

`EoiBlank`
: This resource enables the "blank screen on EOI" feature of the oldest PET 2001.

`DiagPin`
: Set the diagnositc pin on the PET userport (see below).

`ChargenName`
: Specify '`NAME`' as the character generator ROM file

`KernalName`
: Specify '`NAME`' as the kernal ROM file. This file contains the complete BASIC, EDITOR and KERNAL ROMs and is either 16k (BASIC 1 and 2) or 20k (BASIC 4) in size.

`EditorName`
: Specify '`NAME`' as the editor ROM file. This file contains an overlay for the editor ROM at $E000-$E7FF if necessary.

`RomModule9Name`
: Specify '`NAME`' as the $9*** Expansion ROM file. This file contains an expansion ROM image of 4k.

`RomModuleAName`
: Specify '`NAME`' as the $A*** Expansion ROM file. This file contains an expansion ROM image of 4k.

`RomModuleBName`
: Specify '`NAME`' as the $B*** Expansion ROM file. This file contains an expansion ROM image of 4k. This file overlays the lowest 4k of a BASIC 4 ROM.

Choosing a common PET model is done from the right-button menu instead, by choosing an item from the "Model defaults" submenu. Available models are:

- PET 2001-8N
- PET 3008
- PET 3016
- PET 3032
- PET 3032B
- PET 4016
- PET 4032
- PET 4032B
- PET 8032
- PET 8096
- PET 8296
- SuperPET

Notice that this will **reset the emulated machine**.

It is also possible to select the PET model at startup, with the `-model` command-line option: for example, '`xpet -model 3032`' will emulate a PET 3032 while '`xpet -model 8296`' will emulate a PET 8296.

7.6.2 CRTC Settings

7.6.2.1 CRTC resources

`Crtc` Enables CRTC 6545 emulation (all models from 40xx and above)

`CrtcVideoCache`
: Boolean specifying whether the video cache is turned on.

`CrtcDoubleSize`
: Boolean specifying whether double-size mode is turned on.

`CrtcDoubleScan`
: Boolean specifying whether double-scan mode is turned on.

`CrtcPaletteFile`
: String specifying the name of the palette file being used. The '`.vpl`' extension is optional.

7.6.2.2 CRTC command-line options

`-Crtcvcache`
`+Crtcvcache`
: Enable/Disable the video cache

`-Crtcdsize`
`+Crtcdsize`
: Enable/Disable double size

`-Crtcdscan`
`+Crtcdscan`
: Enable/Disable double scan

`-Crtcscale2x`
`+Crtcscale2x`
: Enable/Disable Scale2x filter

`-Crtchwscale`
`+Crtchwscale`
: Enable/Disable hardware scaling

`-Crtcintpal`
: Use an internal calculated palette

`-Crtcextpal`
: Use an external palette (file)

`-Crtcpalette NAME`
: Specify `NAME` as the palette file (`CrtcPaletteFile`).

`-Crtcfulldevice <device>`
: Select fullscreen device

`-CrtcXRANDRfullmode <mode>`
Select fullscreen mode

`-CrtcVidmodefullmode <mode>`
Select fullscreen mode

`-saturation <0-2000>`
Set saturation of internal calculated palette [1000]

`-contrast <0-2000>`
Set contrast of internal calculated palette [1000]

`-brightness <0-2000>`
Set brightness of internal calculated palette [1000]

`-gamma <0-4000>`
Set gamma of internal calculated palette [2200]

`-tint <0-2000>`
Set tint of internal calculated palette [1000]

`-oddlinesphase <0-2000>`
Set phase for color carrier in odd lines [1250]

`-oddlinesoffset <0-2000>`
Set phase offset for color carrier in odd lines [750]

`-crtblur <0-1000>`
Amount of horizontal blur for the CRT emulation. [500]

`-crtscanlineshade <0-1000>`
Amount of scan line shading for the CRT emulation [667]

7.6.3 The PET diagnostic pin

It is possible to enable or disable emulation of the PET diagnostic pin via the `DiagPin` resource, or the "PET userport diagnostic pin" item in the right-button menu.

When the diagnostic pin is set, the Kernal does not try to initialize the BASIC, but directly jumps into the builtin machine monitor.

7.6.4 PET command line options

These are the commandline options specific for the CBM-II models.

`-model MODEL`
Specify the PET model you want to emulate.

`-kernal NAME`
Specify '`NAME`' as the Kernal/BASIC ROM file (`KernalName`).

`-editor NAME`
Specify '`NAME`' as the editor ROM file (`EditorName`).

`-chargen NAME`
Specify '`NAME`' as the character generator ROM file (`ChargenName`).

`-rom9 NAME, -romA NAME, -romB NAME`
: Specify '`NAME`' as the ROM image file for the respective cartridge areas (`RomModule9Name`, `RomModuleAName`, `RomModuleBName`).

`-petram9, +petram9`
: Switch on RAM mapping on addresses $9000-$9fff (`Ram9`).

`-petramA, +petramA`
: Switch on RAM mapping on addresses $a000-$afff (`RamA`).

`-superpet, +superpet`
: Enable/Disable SuperPET I/O emulation (`SuperPET`).

`-basic1, +basic1`
: Enable/Disable patching the IEEE488 section of the PET2001 ROM when detected (`Basic1`).

`-basic1char, +basic1char`
: Enable/Disable PET 2001 character generator (`Basic1Chars`).

`-eoiblank, +eoiblank`
: Enable/Disable EOI blanking the screen (`EoiBlank`).

`-diagpin`
`+diagpin`
: Enable (`DiagPin=1`) or disable (`DiagPin=0`) the diagnostic pin at the PET userport.

`-petreu`
`+petreu`
: Enable or disable the PET Memory Expansion Unit.

`-petreuimage <name>`
: Specify name of PET Ram and Expansion Unit image

`-petreuramsize <size in KB>`
: Size of the PET Ram and Expansion Unit

`-userportdac, +userportdac`
: Enable or disable the userport DAC.

`-petdww`
`+petdww`
: Enable/Disable the PET DWW hi-res board

`-petdwwimage <name>`
: Specify name of PET DWW image

`-sidcart`
`+sidcart`
: Enable/Disable SID Cartridge

7.6.5 Changing screen colors

It is also possible to choose what color set is used for the emulation window. This is done by specifying a palette file name (see Section 4.3 [Palette files], page 20) in the `PaletteName` resource. The menu provides the following values:

- `green.vpl` (default, "green")), the good old green-on-black feeling;
- `amber.vpl` ("amber"), an amber phosphor lookalike;
- `white.vpl` ("white"), simple white-on-black palette.

7.7 CBM-II-specific commands and settings

This section lists the settings and commands that are CBM-II-specific and thus are not present in the other emulators.

7.7.1 Changing CBM-II model

With `xcbm2`, it is possible to change at runtime the characteristics of the emulated CBM so that it matches (or not) the ones of a certain CBM model, and it is also possible to select from a common set of CBM models so that all the features are selected accordingly.

The former is done by changing the following resources (via resource file, command line options or right-menu items):

`UseVicII` Whether to use VIC-II for video output (value 1) or the CRTC for the other machines (value 0)

`RamSize` Size of memory in kByte. Possible values are 128, 256, 512 and 1024

`Ram08, Ram1, Ram2, Ram4, Ram6, RamC`
Expanded CBM-II models could map RAM to the expansion ROM areas at $0800-$0fff, $1000-$1fff, $2000-$3FFF, $4000-$5FFF, $6000-$7FFF and $c000-$cfff respectively.

`Cart2Name, Cart4Name, Cart6Name`
Specify '`NAME`' as the $2000-$3FFF, $4000-$5FFF or $6000-$6FFF Expansion ROM file. This file contains an 8k ROM dump.

`ModelLine`
The CBM-II business models have two hardcoded lines at one of the I/O ports. From those lines the kernal determines how it should init the CRTC video chip for either 50Hz (Europe) or 60Hz (North America), and either for 8 (C6x0) or 14 (C7x0) scanlines per character. 0 = CBM 7x0 (50Hz), 1 = 60Hz C6x0, 2 = 50Hz C6x0).

Choosing a common CBM-II model is done from the right-button menu instead, by choosing an item from the "Model defaults" submenu. Available models are:

- C510 (128k RAM)
- C610 (128k RAM)
- C620 (256k RAM)
- C620+ (1024k RAM, expanded)
- C710 (128k RAM)
- C720 (256k RAM)
- C720+ (1024k RAM, expanded)

Notice that this will **reset the emulated machine**.

Warning: At this time switching between 510 and other machines during runtime is not supported and will not work.

It is also possible to select the CBM model at startup, with the `-model` command-line option: for example, '`xcbm2 -model 610`' will emulate a CBM 610 while '`xcbm2 -model 620`' will emulate a CBM 620. Notably this is the only way to start a C510 emulation, with `-model 510`.

7.7.2 CBM-II command line options

These are the commandline options specific for the CBM-II models.

`-ramsize <ramsize>`
: Specify size of RAM (64/128/256/512/1024 kByte)

`-usevicii`
`+usevicii`
: Specify whether to use (-usevicii) or not to use (+usevicii) the VIC-II emulation.

`-kernal NAME`
: Specify '`NAME`' as the Kernal ROM file (`KernalName`).

`-basic NAME`
: Specify '`NAME`' as the Basic ROM file (`BasicName`).

`-chargen NAME`
: Specify '`NAME`' as the character generator ROM file (`ChargenName`).

`-cart1 <name>`
: Specify '`NAME`' as the ROM image file for the cartridge area $1000-$1FFF (`Cart1Name`).

`-cart2 <name>`
: Specify '`NAME`' as the ROM image file for the cartridge area $2000-$3fff (`Cart2Name`).

`-cart4 <name>`
: Specify '`NAME`' as the ROM image file for the cartridge area $4000-$5fff (`Cart4Name`).

`-cart6 <name>`
: Specify '`NAME`' as the ROM image file for the cartridge area $6000-$7fff (`Cart6Name`).

`-ram08`
`+ram08`
: Enable/Disable RAM mapping in bank 15 on addresses $0800-$0FFF resp (`Ram08`).

`-ram1`
`+ram1`
: Enable/Disable RAM mapping in bank 15 on addresses $1000-$1FFF resp (`Ram1`).

`-ram2`
`+ram2`
: Enable/Disable RAM mapping in bank 15 on addresses $2000-$3FFF resp (`Ram2`).

`-ram4`
`+ram4`
: Enable/Disable RAM mapping in bank 15 on addresses $4000-$5FFF resp (`Ram4`).

`-ram6`
`+ram6`
: Enable/Disable RAM mapping in bank 15 on addresses $6000-$7FFF resp (`Ram6`).

`-ramC`
`+ramC` Enable/Disable RAM mapping in bank 15 on addresses $C000-$CFFF resp (`RamC`).

`-modelline`
Define the hardcoded model switch in the CBM-II models.

7.7.3 Changing screen colors

It is also possible to choose what color set is used for the emulation window. This is done by specifying a palette file name (see Section 4.3 [Palette files], page 20) in the `PaletteName` resource. The menu provides the following values:

- `green.vpl` (default, "green"), the good old green-on-black feeling;
- `amber.vpl` ("amber"), an amber phosphor lookalike;
- `white.vpl` ("white"), simple white-on-black palette.

7.8 VSID-specific commands and settings

7.8.1 VSID settings

7.8.1.1 VSID command-line options

`-keepenv` Override PSID settings for Video standard and SID model

`-tune <number>`
Specify PSID tune <number>

8 Snapshots

Every VICE emulator has a built-in snapshot feature, that saves the complete emulator state into one file for later use. You can therefore save the emulator state - including the state of the game you are playing for example - in a single file.

8.1 Snapshot usage

A snapshot is one file containining the complete emulator state. A snapshot file can be generated by selecting the "Save snapshot" command at any time. This will pop up a requester from which you can specify whether the snapshot should also contain the disk and ROM status.

A snapshot file can be used to restore the emulator state by selecting the `load snapshot` menu entry at any time. Unfortunately attached ROM images/cartridges are only supported in the VIC20, the PET and the CBM-II emulators at this time.

The memory configuration of the emulator is saved in the snapshot file as well. This configuration is restored when the snapshot is loaded.

A quick snapshot can now be made by pressing the `M-F11` key and reloaded by pressing the `M-F10` key.

8.2 Snapshot format

A snapshot file consists of several modules of mostly different types. Each module has a name and saves the state of an entity like a CIA, the CPU, or the memory.

8.2.1 Emulator modules

This section lists the modules that are contained in each of the emulators snapshot files.

8.2.1.1 x64 modules

The modules in the x64 emulator are:

Name	Type	Description
MAINCPU	6502	The Main CPU - although it is a 6510, only the 6502 core is saved here
C64MEM	Memory	Holds the RAM contents of the C64. Also the CPU I/O register contents are saved here.
C64ROM	ROM images	Dump of the system ROMs
VIC-II	656*	The VIC-II of the C64/128
CIA1	6526	The CIA for the interrupts and the keyboard
CIA2	6526	The CIA for the userport, IEC-bus and RS232.
SID	6581	The SID sound chip of the C64/C128
REU*		The RAM Extension Unit state (optional)
ACIA1	6551	An ACIA (RS232 interface) at $DE00 (optional)
TPI	6525	A TPI at $DF00 for a parallel IEEE488 interface (optional)
*	Drive modules	The emulated drive(s) have their own modules see Section 8.2.1.6 [Drive modules], page 90

Some of the modules are optional and are only saved if the specific feature is enabled at save-time. If the module is found when restoring the state the optional features are enabled, and disabled otherwise.

8.2.1.2 x128 modules

The modules in the x128 emulator are:

Name	Type	Description
MAINCPU	6502	The Main CPU - although it is a 6510, only the 6502 core is saved here
C128MEM	Memory	Holds the RAM contents of the C64. Also the CPU I/O register contents are saved here.
C128ROM	ROM images	Dump of the system ROMs
VIC-II	656*	The VIC-II of the C64/128

CIA1	6526	The CIA for the interrupts and the keyboard
CIA2	6526	The CIA for the userport, IEC-bus and RS232.
SID	6581	The SID sound chip of the C64/C128
ACIA1	6551	An ACIA at $DE00 (optional)
TPI	6525	A TPI at $DF00 for a parallel IEEE488 interface (optional)
*	Drive modules	The emulated drive(s) have their own modules see Section 8.2.1.6 [Drive modules], page 90

Some of the modules are optional and are only saved if the specific feature is enabled at save-time. If the module is found when restoring the state the optional features are enabled, and disabled otherwise.

Not yet supported are the 80 column video chip, cartridges and RAM expansion unit.

8.2.1.3 xvic modules

The modules in the xvic emulator are:

Name	Type	Description
MAINCPU	6502	The Main CPU
VIC20MEM	Memory	Holds the RAM contents of the VIC20.
VIC20ROM	ROM images	Holds the ROM images of the VIC20, including possibly attached cartridges
VIC-I	656*	The VIC-I of the VIC20
VIA1	6522	The VIA for the interrupts and the keyboard
VIA2	6522	The VIA for the userport, IEC-bus and RS232.
*	Drive modules	The emulated drive(s) have their own modules see Section 8.2.1.6 [Drive modules], page 90

8.2.1.4 xpet modules

The modules in the xpet emulator are:

Name	Type	Description
MAINCPU	6502	The Main CPU
PETMEM	Memory	Holds the RAM contents of the PET.
PETROM	ROM images	Holds the ROM images of the PET, including possibly attached cartridges
CRTC	6545	The CRTC of the PET. This is also included if it is a dump of a PET without CRTC, because the video state is saved here anyway.
PIA1	6520	The PIA for the interrupts, tape and the keyboard

PIA2	6520	The PIA for the IEEE488-bus
VIA	6522	The VIA for IEEE488, userport, sound
ACIA1	6551	The ACIA for the SuperPET. This module is optional.
*	Drive modules	The emulated drive(s) have their own modules see Section 8.2.1.6 [Drive modules], page 90

8.2.1.5 xcbm2 modules

The modules in the xcbm2 emulator are:

Name	Type	Description
MAINCPU	6502	The Main CPU - although it is a 6509, only the 6502 core is saved here
CBM2MEM	Memory	Holds the RAM contents of the CBM-II models. Also holds the exec-bank and indirection bank registers
C500DATA		Holds additional state information necessary for the C500 (e.g. cycles till the next IRQ)
CBM2ROM	Memory	optional. Holds the ROM images.
CRTC	6545	The video chip for the C6*0 and C7*0 models (only those models).
VIC-II	656?	The video chip for the C5*0 models (only the C5*0 models).
CIA1	6526	The CIA for IEEE 488 and userport.
TPI1	6525	TPI 1 for IEEE488
TPI2	6525	TPI 2 for interrupts and keyboard.
ACIA1	6551	The RS232 interface
SID	6581	The CBM2s SID sound chip
*	Drive modules	The emulated drive(s) have their own modules see Section 8.2.1.6 [Drive modules], page 90

The snapshot either contains CRTC or VIC-II snapshot modules, but not both. Currently switching between the two video emulations is not possible at runtime, so only snapshots that fit the current UseVicII resource are accepted.

8.2.1.6 Drive modules

The modules for the real disk drive emulation are included in the emulator when the emulation is enabled during the writing of the snapshot.

Name	Type	Description
*CPU	6502	The Drive 0 CPU
*	*	*

8.2.2 Module formats

This section shows the basic module framework and the contents of the different types of modules.

The single chip modules contain the **chip** state, not the state of the emulator. We tried to make the format as implementation-independent as possible, to allow reuse of snapshots in later versions of this emulator, or even in other emulators.

8.2.2.1 Terminology

In this section we use certain abbreviations to define the types of the data saved in the snapshot.

`BYTE` 8 bit integer.

`WORD` 16 bit integer. Saved with low-byte first, high-byte last.

`DWORD` 32 bit integer. Saved with low-word first, then high-word. Each word saved with its low-byte first.

`ARRAY` Array of BYTE values. Length depends on the description.

The tables for the single modules state the type, name and description of the data saved in the modules. The data is saved in the order it is in the tables, so no offset is given.

8.2.2.2 Module framework

The VICE snapshot file starts with the magic string and includes the fileformat version number.

Type	Name	Description
19 BYTE	MAGIC	"VICE Snapshot File\032", padded with 0
BYTE	VMAJOR	fileformat major version number
BYTE	VMINOR	fileformat minor version number
16 BYTE	MACHINENAME	Name of emulated machine, like "PET", "CBM-II", "VIC20", "C64" or "C128". zerobyte-padded.

The file header is followed by a number of different snapshot modules.

Each module has a header with the information given in the table below. The header includes two version numbers, VMAJOR and VMINOR. Modules with the same VMAJOR should be able to be exchanged. I.e. higher VMINOR numbers only append to the data for lower VMINOR. This additional data is ignored by older restore routines. The other way around newer restore routines must accept the fewer info from lower VMINOR dumps. Changes in VMAJOR might introduce any incompatibility you like, but that's what VMAJOR is for after all :-)

Type	Name	Description
16 BYTE	MODULENAME	The name of the module in ASCII, padded with 0 to 16 byte.
BYTE	VMAJOR	major version number
BYTE	VMINOR	minor version number
DWORD	SIZE	size of the module, including this header

8.2.2.3 CPU module

This module saves the core 6502 state. You will find a clock value there. All other modules save their own clock values relative to this value. However, the drive modules save their clocks relative to their appropriate CPUs of course.

Warning: This module is still under construction and saves some information that is not sure to be VICE-independent. If in doubt, read the source.

Type	Name	Description
DWORD	CLK	the current CPU clock value. All other clock values are relative to this.
BYTE	AC	Accumulator
BYTE	XR	X index register
BYTE	YR	Y index register
BYTE	SP	Stack Pointer
WORD	PC	Programm Counter
BYTE	ST	Status Registers
DWORD	LASTOPCODE	?
DWORD	IRQCLK	absolute CLK when the IRQ line came active
DOWRD	NMICLK	absolute CLK when the NMI line came active
DWORD	?	?
DWORD	?	?

8.2.2.4 CIA module

The CIA 6526 is an I/O port chip with 2 8-bit I/O ports, a shift register, two timers, a Time of Day clock and interrupts.

Version numbers: Major 1, Minor 1.

Type	Name	Description
BYTE	ORA	Output register A
BYTE	ORB	Output register B
BYTE	DDRA	Data direction register A
BYTE	DDRB	Data direction register B
WORD	TAC	Timer A counter value
WORD	TBC	Timer B counter value
BYTE	TOD_TEN	Time of Day - current tenth of second
BYTE	TOD_SEC	Time of Day - current seconds
BYTE	TOD_MIN	Time of Day - current minutes
BYTE	TOD_HR	Time of Day - current hours
BYTE	SDR	contents of shift register
BYTE	IER	mask of enabled interrupt masks
BYTE	CRA	Control register A
BYTE	CRB	Control register B
WORD	TAL	Timer A latch value
WORD	TBL	Timer B latch value
BYTE	IFR	mask of currently active interrupts

BYTE	PBSTATE	Bit 6/7 reflect the PB6/7 toggle bit state. Bit 2/3 reflect the corresponding port bit state.
BYTE	SRHBITS	number of half-bits to still shift in/out SDR
BYTE	ALARM_TEN	Time of Day - alarm tenth of second
BYTE	ALARM_SEC	Time of Day - alarm seconds
BYTE	ALARM_MIN	Time of Day - alarm minutes
BYTE	ALARM_HR	Time of Day - alarm hours
BYTE	READICR	current clock minus the clock when ICR was read last plus 128.
BYTE	TODLATCHED	Bit 0: 1= latched for reading, Bit 1: 2=stopped for writing
BYTE	TODL_TEN	Time of Day - latched tenth of second
BYTE	TODL_SEC	Time of Day - latched seconds
BYTE	TODL_MIN	Time of Day - latched minutes
BYTE	TODL_HR	Time of Day - latched hours
DWORD	TOD_TICKS	clk ticks till next tenth of second
–	–	The next items have been added in V1.1
WORD	TASTATE	The state bits of the CIA timer A, according to ciatimer.h
WORD	TBSTATE	The state bits of the CIA timer B, according to ciatimer.h

The last two items have been added in CIA snapshot version 1.1 due to the improved CIA emulation in the newer VICE versions. Some state bits correspond to the CIA state as described in the "A Software Model of the CIA 6526" document by Wolfgang Lorenz, some are delayed versions. For more read the source file `ciatimer.h`.

8.2.2.5 VIA module

The VIA 6522 is the predecessor of the CIA and also an I/O port chip with 2 8-bit I/O ports, a shift register, two timers and interrupts.

Version numbers: Major 1, Minor 0.

Type	Name	Description
BYTE	ORA	Output register A
BYTE	DDRA	Data direction register A
BYTE	ORB	Output register B
BYTE	DDRB	Data direction register B
WORD	T1L	Timer 1 Latch value
WORD	T1C	Timer 1 counter value
BYTE	T2L	Timer 2 latch (8 bit as only lower byte is used)
WORD	T2C	Timer 2 counter value
BYTE	RUNFL	bit 7: timer 1 will generate IRQ on underflow; bit 6: timer 2 will generate IRQ on underflow
BYTE	SR	Shift register value
BYTE	ACR	Auxiliary control register

BYTE	PCR	Peripheral control register
BYTE	IFR	active interrupts
BYTE	IER	interrupt mask
BYTE	PB7	bit 7 = pb7 state
BYTE	SRHBITS	number of half-bits to shift out on SR
BYTE	CABSTATE	bit 7: state of CA2 pin, bit 6: state of CB2 pin
BYTE	ILA	Port A Input Latch (see ACR bit 0)
BYTE	ILB	Port B Input Latch (see ACR bit 1)

8.2.2.6 PIA module

The PIA 6520 is a chip with two I/O ports (Parallel Interface Adapter) and four additional handshake lines. The chip is pretty the same for Port A and B, only that Port A implements handshake on read operation and port B on write operation.

Version numbers: Major 1, Minor 0.

Type	Name	Description
UBYTE	ORA	Output register A
UBYTE	DDRA	Data Direction Register A
UBYTE	CTRLA	Control Register A
UBYTE	ORB	Output register B
UBYTE	DDRB	Data Direction Register B
UBYTE	CTRLB	Control Register B
UBYTE	CABSTATE	Bit 7 = state of CA2, Bit 6 = state of CB2

8.2.2.7 TPI module

The TPI 6525 is a chip with three I/O ports (Tri-Port-Interface). One of the ports can double as an interrupt prioritizer. Therefore we also have to save the states of the interrupt stack etc.

Version numbers: Major 1, Minor 0.

Type	Name	Description
BYTE	PRA	Port A output register
BYTE	PRB	Port B output register
BYTE	PRC	Port C output register (doubles as IRQ latch register)
BYTE	DDRA	Port A data direction register
BYTE	DDRB	Port B data direction register
BYTE	DDRC	Port C data direction register (doubles as IRQ mask register)
BYTE	CR	Control Register
BYTE	AIR	Active interrupt register
BYTE	STACK	Interrupt stack - the interrupt bits that are not (yet) served.
BYTE	CABSTATE	State of CA/CB pins. Bit 7 = state of CA, Bit 6 = state of CB

8.2.2.8 RIOT module

The RIOT 6532 is a chip with two I/O ports, some RAM and a Timer. The chip contains 128 byte RAM, but the RAM is not saved in the RIOT snapshot, but in the memory section.

Warning: This module is still under construction

Version numbers: Major 0, Minor 0.

Type	Name	Description
BYTE	ORA	Port A output register
BYTE	DDRA	Port A data direction register
BYTE	ORB	Port B output register
BYTE	DDRB	Port B data direction register
BYTE	EDGECTRL	Bit 0/1: A0/A1 address bits written to edgecontrol registers
BYTE	IRQFL	Bit 6/7: A6/A7 IRQ flag register. Bit 0: state of the IRQ line (0=inactive, 1=active)
BYTE	N	timer value
WORD	DIVIDER	Pre-scale divider value (1, 8, 64, or 1024)
WORD	REST	cycles since the last counter change
BYTE	IRQEN	Bit 0: 0= timer IRQ disabled, 1= timer IRQ enabled

8.2.2.9 SID module

Warning: This module is still under construction.

8.2.2.10 ACIA module

The ACIA 6551 is an RS232 interface chip. VICE emulates RS232 connections via `/dev/ttyS*` (Unix) or `COM:` (DOS/WIN - not yet?). When saving a snapshot, those connections are of course lost. The state of the ACIA however is restored if possible. I.e. if a connection is already open when restoring the snapshot, this connection is used instead. If no connection is open, a carrier/DTR drop is emulated.

Version numbers: Major 1, Minor 0.

Type	Name	Description
BYTE	TDR	Transmit Data Register
BYTE	RDR	Receiver Data Register
BYTE	SR	Status Register
BYTE	CMD	Command Register
BYTE	CTRL	Ctrl Register
BYTE	INTX	0 = no data to tx; 1 = Data is being transmitted; 2 = Data is being transmitted while data in TDR waiting to be put to internal transmit register
DWORD	TICKS	Clock ticks till the next TDR empty interrupt

8.2.2.11 VIC-I module

Warning: This module is still under construction.

8.2.2.12 VIC-II module

Warning: This module is still under construction.

8.2.2.13 CRTC module

Warning: After VICE version 1.0 the CRTC emulation has improved considerably. Especially it is now cycle exact. Therefore a lot more variables must be saved. The snapshot module version jumped from 0.0 to 1.0. Newer versions of VICE can read the old snapshots, but older versions (1.0 and below) cannot read the new snapshots.

Warning: This module is still under construction. Especially the RASTERY and RASTERLINE values might be bogus.

Version numbers: Major 1, Minor 1.

Type	Name	Description
		Hardware options
WORD	VADDR_MASK	Mask of the address bits valid when accessing the video memory
WORD	VADDR_CHARSWITCH	If one bit in the video address is used to switch the character generator, it is masked here.
WORD	VADDR_CHAROFFSET	The offset in characters in the character generator that CHARSWITCH switches.
WORD	VADDR_REVSWITCH	If one bit in the video address inverts the screen, it is masked here.
WORD	CHARGEN_MASK	size of character generator in byte - 1
WORD	CHARGEN_OFFSET	offset given by external circuitry
BYTE	HW_CURSOR	external hardware cursor circuitry enabled
BYTE	HW_COLS	number of displayed columns during one character clock cycle
BYTE	HW_BLANK	set if the hardware blank feature is available
		CRTC register
20 BYTE	REGISTERS	register DUMP of the CRTC registers 0-19.
		CRTC internal registers
BYTE	REGNO	The current index in the CRTC register file
BYTE	CHAR	The current cycle within the current rasterline
BYTE	CHARLINE	The current character line
BYTE	YCOUNTER	The current rasterline in the character
BYTE	CRSRCNT	Framecounter for the blinking cursor
BYTE	CRSRSTATE	if set the hardware cursor is visible
BYTE	CRSRLINES	set if ycounter is within the active cursor rasterlines for a char
WORD	CHARGEN_REL	relative base of currently used character generator in ROM (in byte)
WORD	SCREEN_REL	screen address to load the counter at the beginning of the next rasterline

WORD	VSYNC	number of rasterlines left within vsync; 0 = not in vsync
BYTE	VENABLE	vertical enable flipflop; 1= display, 0= blank.
		(VICE-dependent?) variables
WORD	SCREEN_WIDTH	width of the current display window
WORD	SCREEN_HEIGHT	height of the current display window
WORD	SCREEN_XOFFSET	x position where the first character in a line starts in the window...
WORD	HJITTER	...but only after adding this jitter
WORD	SCREEN_YOFFSET	x position where the first character in a line starts in the window...
WORD	FRAMELINES	expected number of rasterlines for the current frame
WORD	CURRENT_LINE	current rasterline as seen from the CRTC This value has been added in module version V1.1
BYTE	FLAG	Bit 0: If 1 then bit in VADDR_REVSWITCH must be set for reverse; if 0 then bit must be cleared for reverse.

Here is the reference for the previous CRTC snapshot module. It is outdated and will not be read by this and later versions of VICE.

Version numbers: Major 0, Minor 0.

Type	Name	Description
BYTE	RASTERY	The number of clock cycles from rasterlines start
WORD	RASTERLINE	The current rasterline
WORD	ADDRMASK	The address mask valid for the CRTC. All memory accesses are masked with this value
BYTE	HWFLAG	Bit 0: 1= hardware cursor available. Bit 1: 1= number of columns is doubled by external hardware
20 BYTE	REGISTERS	register DUMP of the CRTC registers 0-19.
BYTE	CRSRSTATE	Hardware cursor: Bits 0-3: frame counter till next crsr line toggle. Bit 7: 1= cursor line active

8.2.2.14 C64 memory module

The C64 memory module actually consists of two modules. The "C64MEM" module is mandatory and contains the RAM dump. The "C64ROM" module is optional and contains a dump of the ROM images.

The size of the C64 memory modules differs with each different memory configuration. The RAM configuration is saved in the snapshot, and restored when the snapshot is loaded. The attached cartridges are **not yet(!)** saved and not yet restored upon load.

Version numbers: Major 0, Minor 0

The C64MEM module

Type	Name	Description
BYTE	CPUDATA	CPU port data byte
BYTE	CPUDIR	CPU port direction byte
BYTE	EXROM	state of the EXROM line (?)
BYTE	GAME	state of the GAME line (?)
ARRAY	RAM	64k RAM dump

The C64ROM module

Type	Name	Description
ARRAY	KERNAL	8k dump of the kernal ROM
ARRAY	BASIC	8k dump of the basic ROM
ARRAY	CHARGEN	4k dump of the chargen ROM

8.2.2.15 C128 memory module

The C128 memory module actually consists of two modules. The "C128MEM" module is mandatory and contains the RAM dump. The "C128ROM" module is optional and contains a dump of the ROM images.

The size of the C128 memory modules differs with each different memory configuration. The RAM configuration is saved in the snapshot, and restored when the snapshot is loaded. The attached cartridges are also restored upon load if they have been saved in the snapshot.

Version numbers: Major 0, Minor 0

The C128MEM module

Type	Name	Description
12 BYTE	MMU	dump of the 12 MMU registers
ARRAY	RAM	128k RAM dump banks 0 and 1

The C128ROM module

Type	Name	Description
ARRAY	KERNAL	8k dump of the kernal ROM
ARRAY	BASIC	32k dump of the basic ROM
ARRAY	EDITOR	4k dump of the editor ROM
ARRAY	4k CHARGEN	dump of the chargen ROM

8.2.2.16 VIC20 memory module

The VIC20 memory module actually consists of two modules. The "VIC20MEM" module is mandatory and contains the RAM dump. The "VIC20ROM" module is optional and contains a dump of the ROM images.

The size of the VIC20 memory modules differs with each different memory configuration. The RAM configuration is saved in the snapshot, and restored when the snapshot is loaded. The attached cartridges are also restored upon load if they have been saved in the snapshot.

The VIC20MEM module

Version numbers: Major 1, Minor 0

Type	Name	Description
BYTE	CONFIG	Configuration register. Bits 0,1,2,3,5 reflect if the corresponding memory block is RAM (bit=1) or not (bit=0).
ARRAY	RAM0	1k RAM dump $0000-$03ff
ARRAY	RAM1	4k RAM dump $1000-$1fff
ARRAY	COLORRAM	2k Color RAM, $9400-$9bff
ARRAY	BLK0	if CONFIG & 1 then: 3k RAM dump $0400-$0fff
ARRAY	BLK1	if CONFIG & 2 then: 8k RAM dump $2000-$3fff
ARRAY	BLK2	if CONFIG & 4 then: 8k RAM dump $4000-$5fff
ARRAY	BLK3	if CONFIG & 8 then: 8k RAM dump $6000-$7fff
ARRAY	BLK5	if CONFIG & 32 then: 8k RAM dump $a000-$bfff

The VIC20ROM module

Version numbers: Major 1, Minor 1

Type	Name	Description
BYTE	CONFIG	Bit 0: 1= ROM block $2*** enabled. Bit 1: 1= ROM block $3*** enabled. Bit 2: 1= ROM block $4*** enabled. Bit 3: 1= ROM block $5*** enabled. Bit 4: 1= ROM block $6*** enabled. Bit 5: 1= ROM block $7*** enabled. Bit 6: 1= ROM block $A*** enabled. Bit 7: 1= ROM block $B*** enabled.
ARRAY	KERNAL	8k KERNAL ROM image $e000-$ffff
ARRAY	BASIC	16k BASIC ROM image $c000-$dfff
ARRAY	CHARGEN	4k CHARGEN ROM image
ARRAY	BLK1A	4k ROM image $2*** (if CONFIG & 1)
ARRAY	BLK1B	4k ROM image $3*** (if CONFIG & 2)
ARRAY	BLK3A	4k ROM image $6*** (if CONFIG & 16)
ARRAY	BLK3B	4k ROM image $7*** (if CONFIG & 32)
ARRAY	BLK5A	4k ROM image $A*** (if CONFIG & 64)
ARRAY	BLK5B	4k ROM image $B*** (if CONFIG & 128)
ARRAY	BLK2A	4k ROM image $4*** (if CONFIG & 4; added in V1.1)
ARRAY	BLK2B	4k ROM image $5*** (if CONFIG & 8; added in V1.1)

8.2.2.17 PET memory module

The PET memory module actually consists of two modules. The "PETMEM" module is mandatory and contains the RAM dump. The "PETROM" module is optional and contains a dump of the ROM images.

The size of the PET memory modules differs with each different memory configuration. The RAM configuration is saved in the snapshot, and restored when the snapshot is loaded.

The PETMEM module

Version numbers: Major 1, Minor 2

Type	Name	Description
BYTE	CONFIG	Configuration value. Bits 0-3: 0= 40 col PET without CRTC; 1= 40 col PET with CRTC; 2 = 80 col PET (with CRTC); 3= SuperPET; 4= 8096; 5= 8296. Bit 6: 1= RAM at $9***. Bit 7: 1= RAM at $A***.
BYTE	KEYBOARD	Keyboard type. 0= UK business; 1= Graphics; 2= German business
BYTE	MEMSIZE	memory size of low 32k in k (possible values 4, 8, 16, 32)
BYTE	CONF8X96	Value of the 8x96 configuration register
BYTE	SUPERPET	SuperPET config. Bit 0: 1= $9*** RAM enabled. Bit 1: 1= RAM write protected. Bit 2: 1= CTRL register write protected. Bit 3: 0= DIAG pin active. Bits 4-7: RAM block in use.
ARRAY	RAM	4-32k RAM (not 8296, size depends on MEMSIZE)
ARRAY	VRAM	2/4k RAM (not 8296, size depends on CONFIG)
ARRAY	EXTRAM	64k expansion RAM (SuperPET and 8096 only)
ARRAY	RAM	128k RAM (8296 only)
–	–	The following item has been added in V1.1
BYTE	POSITIONAL	bit 0=0 = symbolic keyboard mapping, bit 0=1 = positional mapping.
–	–	The following item has been added in V1.1
BYTE	EOIBLANK	bit 0=0 = EOI does not blank screen, bit 0=1 = EOI blanks screen.

The last item has been added in PETMEM snapshot version 1.1. It is ignored by earlier restore routines (V1.0) and the V1.1 restore routines do not change the current setting when reading a V1.0 snapshot.

In V1.2 the new EOIBLANK variable has been added. This implements the "blank screen on EOI" feature that was previously linked to a wrong resource.

The PETROM module

Version numbers: Major 1, Minor 0

Type	Name	Description
BYTE	CONFIG	Bit 0: 1= $9*** ROM included. Bit 1: 1= $A*** ROM included. Bit 2: 1= $B*** ROM included. Bit 3: 1= $e900-$efff ROM included
ARRAY	KERNAL	4k KERNAL ROM image $f000-$ffff
ARRAY	EDITOR	2k EDITOR ROM image $e000-$e7ff
ARRAY	CHARGEN	2k CHARGEN ROM image
ARRAY	ROM9	4k $9*** ROM image (if CONFIG & 1)
ARRAY	ROMA	4k $A*** ROM image (if CONFIG & 2)
ARRAY	ROMB	4k $B*** ROM image (if CONFIG & 4)
ARRAY	ROMC	4k $C*** ROM image
ARRAY	ROMD	4k $D*** ROM image
ARRAY	ROME9	7 blocks $e900-$efff ROM image (if CONFIG & 8)

8.2.2.18 CBM-II memory module

The CBM-II memory module actually consists of two modules. The "CBM2MEM" module is mandatory and contains the RAM dump. The "CBM2ROM" module is optional and contains a dump of the ROM images.

The size of the CBM-II memory modules differs with each different memory configuration. The RAM configuration is saved in the snapshot, and restored when the snapshot is loaded.

Version numbers: Major 1, Minor 0

The CBM2MEM module

Type	Name	Description
UBYTE	MEMSIZE	Memory size in 128k blocks (1=128k, 2=256k, 4=512k, 8=1024k)
UBYTE	CONFIG	Bit 0 = $f0800-$f0fff RAM, Bit 1 = $f1000-$f1fff RAM, Bit 2 = $f2000-$f3fff RAM, Bit 3 = $f4000-$f5fff RAM, Bit 4 = $f6000-$f7fff RAM, Bit 5 = $fc000-$fcfff RAM, Bit 6 = is a C500
UBYTE	HWCONFIG	Bit 0/1: model line configuration
UBYTE	EXECBANK	CPUs execution bank register
UBYTE	INDBANK	CPUs indirection bank register
ARRAY	SYSRAM	2k system RAM $f0000-$f07ff
ARRAY	VIDEO	2k video RAM $fd000-$fd7ff
ARRAY	RAM	RAM dump, size according to MEMSIZE
ARRAY	RAM08	if memsize < 1M and CONFIG & 1 : 2k RAM $f0800-$f0fff
ARRAY	RAM1	if memsize < 1M and CONFIG & 2 : 4k RAM $f1000-$f1fff
ARRAY	RAM2	if memsize < 1M and CONFIG & 4 : 8k RAM $f2000-$f3fff

ARRAY	RAM4	if memsize < 1M and CONFIG & 8 : 8k RAM $f4000-$f5fff
ARRAY	RAM6	if memsize < 1M and CONFIG & 16 : 8k RAM $f6000-$f7fff
ARRAY	RAMC	if memsize < 1M and CONFIG & 32 : 4k RAM $fc000-$fcfff

The RAM* arrays are only saved if the RAM itself is less than 1M. If the memory size is 1M then those areas are taken from the bank 15 area of the normal RAM.

The memory array starts at $10000 if the memory size is less than 512k, or at $00000 if 512k or more. In case of a C510, then the memory array also always starts at $00000.

The CBM2ROM module

Type	Name	Description
UBYTE	CONFIG	Bit 1: 1= $1*** ROM image included. Bit 2: 1= $2000-$3fff ROM image included. Bit 3: 1= $4000-$5fff ROM image included. Bit 4: 1= $6000-$7fff ROM image included. Bit 5: 1= chargen ROM is VIC-II chargen, 0= CRTC chargen.
ARRAY	KERNAL	8 KERNAL ROM image ($e000-$efff)
ARRAY	BASIC	BASIC ROM image ($8000-$bfff)
ARRAY	CHARGEN	4k CHARGEN ROM image
ARRAY	ROM1	4k cartridge ROM image for $1*** (if CONFIG & 2)
ARRAY	ROM2	8k cartridge ROM image for $2000-$3fff (if CONFIG & 4)
ARRAY	ROM4	8k cartridge ROM image for $4000-$5fff (if CONFIG & 8)
ARRAY	ROM6	8k cartridge ROM image for $6000-$7fff (if CONFIG & 16)

8.2.2.19 C500 data module

The C500 data module contains simple state information not already saved in the other modules.

Version numbers: Major 0, Minor 0

The C500DATA module

Type	Name	Description
DWORD	IRQCLK	CPU clock ticks till next 50Hz IRQ

9 Media images

9.1 Media images command-line options

`-doodleoversize <method>`
: Select the way the oversized input should be handled, (0: scale down, 1: crop left top, 2: crop center top, 3: crop right top, 4: crop left center, 5: crop center, 6: crop right center, 7: crop left bottom, 8: crop center bottom, 9: crop right bottom)

`-doodlemc <method>`
: Select the way the multicolor to hires should be handled, (0: b&w, 1: 2 colors, 2: 4 colors, 3: gray scale, 4: best cell colors)

`-doodletedlum <method>`
: Select the way the TED luminosity should be handled, (0: ignore, 1: dither)

`-doodlecrtctextcolor <color>`
: Select the CRTC text color (0: white, 1: amber, 2: green)

`-ffmpegaudiobitrate <value>`
: Set bitrate for audio stream in media file

`-ffmpegvideobitrate <value>`
: Set bitrate for video stream in media file

10 Event history

10.1 Event history command-line options

`-playback`
: Playback recorded events

11 Monitor

Every VICE emulator has a complete built-in monitor, which can be used to examine, disassemble and assemble machine language programs, as well as debug them through breakpoints. It can be activated by using the "Activate monitor" command (left button menu). Notice that you have to run the emulator from a terminal emulation program (such as `rxvt` or `xterm`) in order to use the monitor.

Warning: this version of the monitor is still under construction, and some of the features are not fully working yet.

11.1 Terminology

'`address_space`'
: This refers to the range of memory locations and a set of registers. This can be the addresses available to the computer's processor, the disk drive's processor or a specific memory configuration of one of the mentioned processors.

'`bankname`'
: The CPU can only see 64k of memory at any one time, due to its 16 bit address bus. The C64 and other computers have more than this amount, and this is handled by banking: a memory address can have different contents, depending on the active memory bank. A bankname names a specific bank in the current address_space.

'`register`'
: One of the following: program counter (PC), stack pointer (SP), accumulator (A), X register (X), or Y register (Y).

'`address`'
: A specific memory location in the range $0000 to $FFFF.

'`address_range`'
: Two addresses. If the second address is less than the first, the range is assumed to wraparound from $FFFF to $0000. Both addresses must be in the same address space.

'`address_opt_range`'
: An address or an address range.

'`label`'
: `label` is the name of a label. it must start with a dot (`"."`) in order for the monitor to recognize it as a label.

'`prompt`'
: The prompt has the format [x:y]. If x is -, memory reads from the monitor do not have side effects. Otherwise, x is S. The second part of the prompt, y, shows the default address space.

'`checkpoint`'
: The monitor has the ability to setup triggers that perform an action when a specified situation occurs. There are three types of checkpoints; breakpoints, tracepoints and watchpoints.

'`breakpoint`'
: A breakpoint is triggered based on the program counter. When it is triggered, the monitor is entered.

'`tracepoint`'
: Like breakpoints, a tracepoint is triggered based on the program counter. Instead of entering the monitor, the program counter is printed and execution continues.

'`watchpoint`'
: Watchpoints are triggered by a read and/or write to an address. When a watchpoint is triggered, the monitor is entered.

'`memmap`'
: The memmap keeps track of RAM/ROM/IO read/write/execute accesses. The feature must be enabled with `"–enable-memmap"` configure option, as it might decrease performance notably on slower hardware. The option also enables CPU history.

'`<...>`'
: A data type.

'`*`'
: Zero or more occurrences.

'`[...]`'
: An optional argument.

11.2 Machine state commands

`backtrace`
`bt` Print JSR call chain (most recent call first). Stack offset relative to SP+1 is printed in parentheses. This is a best guess only.

`cpuhistory [<count>]`
`chis [<count>]`
Show <count> last executed commands. (disabled by default; configure with –enable-memmap to enable)

`dump "<filename>"`
Write a snapshot of the machine into the file specified. This snapshot is compatible with a snapshot written out by the UI. Note: No ROM images are included into the dump.

`goto <address>`
`g <address>`
Change the PC to address and continue execution.

`io [<address>]`
Display i/o registers. Invoking without an address shows a dump of the entire io range, if an address is given then details for the chip at the respective (base-)address are displayed (if available).

`next [<count>]`
`n [<count>]`
Advance to the next instruction. Subroutines are treated as a single instruction.

`registers [<reg_name> = <number> [, <reg_name> = <number>]*]`
`r [<reg_name> = <number> [, <reg_name> = <number>]*]`
Assign respective registers. With no parameters, display register values.

`reset [<type>]`
Reset the machine or drive. `type`: 0 = soft, 1 = hard, 8-11 = drive.

`return`
`ret` Continues execution and returns to the monitor just after the next RTS or RTI is executed.

`step [<count>]`
`z [<count>]`
Single step through instructions. An optional count allows stepping more than a single instruction at a time.

`stopwatch [reset]`
Print the CPU cycle counter of the current device. 'reset' sets the counter to 0.

`undump "<filename>"`
Read a snapshot of the machine from the file specified.

11.3 Memory commands

`bank [<bankname>]`
: Without a bankname, display all available banks for the current address_space. With a bankname given, switch to the specified bank. If a bank is not completely filled (ROM banks for example) normally the `ram` bank is used where the bank has holes. The `cpu` bank uses the bank currently used by the CPU.

`compare <address_range> <address>`
`c <address_range> <address>`
: Compare memory from the source specified by the address range to the destination specified by the address. The regions may overlap. Any values that miscompare are displayed using the default displaytype.

`device [c:|8:|9:]`
: Set the default address space to either the computer 'c:' or the specified drive '8:' or '9:'

`fill <address_range> <data_list>`
`f <address_range> <data_list>`
: Fill memory in the specified address range with the data in <data_list>. If the size of the address range is greater than the size of the data_list, the data_list is repeated.

`hunt <address_range> <data_list>`
`h <address_range> <data_list>`
: Hunt memory in the specified address range for the data in <data_list>. If the data is found, the starting address of the match is displayed. The entire range is searched for all possible matches. The data list may have 'xx' as a wildcard.

`i <address_opt_range>`
: Display memory contents as PETSCII text.

`ii <address_opt_range>`
: Display memory contents as screen code text

`mem [<data_type>] [<address_opt_range>]`
`m [<data_type>] [<address_opt_range>]`
: Display the contents of memory. If no datatype is given, the default is used. If only one address is specified, the length of data displayed is based on the datatype. If no addresses are given, the 'dot' address is used.

`memmapshow [<mask>] [<address_opt_range>]`
`mmsh [<mask>] [<address_opt_range>]`
: Show the memmap. The mask can be specified to show only those locations with accesses of certain type(s). The mask is a number with the bits "ioRWXrwx", where RWX are for ROM and rwx for RAM. Optionally, an address range can be specified. (disabled by default; configure with –enable-memmap to enable)

`memmapzap`
`mmzap`
: Clear the memmap. (disabled by default; configure with –enable-memmap to enable)

`memmapsave "<filename>" <format>`
`mmsave "<filename>" <format>`
Save the memmap as a picture. `format`: 0 = BMP, 1 = PCX, 2 = PNG, 3 = GIF, 4 = IFF. (disabled by default; configure with –enable-memmap to enable)

`memchar [<data_type>] [<address_opt_range>]`
`mc [<data_type>] [<address_opt_range>]`
Display the contents of memory as character data. If only one address is specified, only one character is displayed. If no addresses are given, the "dot" address is used.

`memsprite [<data_type>] [<address_opt_range>]`
`ms [<data_type>] [<address_opt_range>]`
Display the contents of memory as sprite data. If only one address is specified, only one sprite is displayed. If no addresses are given, the "dot" address is used.

`move <address_range> <address>`
`t <address_range> <address>`
Move memory from the source specified by the address range to the destination specified by the address. The regions may overlap.

`screen`
`sc` Displays the contents of the screen.

`sidefx [on|off|toggle]`
`sfx [on|off|toggle]`
Control how monitor generated reads affect memory locations that have read side-effects, like CIA interrupt registers for example. If the argument is 'on' then reads may cause side-effects. If the argument is 'off' then reads don't cause side-effects. If the argument is 'toggle' then the current mode is switched. No argument displays the current state.

`> [<address>] <data_list>`
Write the specified data at `address`.

11.4 Assembly commands

`a <address> [ <instruction> [: <instruction>]* ]`
Assemble instructions to the specified address. If only one instruction is specified, enter assembly mode (enter an empty line to exit assembly mode).

`disass [<address> [<address>]]`
`d [<address> [<address>]]`
Disassemble instructions. If two addresses are specified, they are used as a start and end address. If only one is specified, it is treated as the start address and a default number of instructions are disassembled. If no addresses are specified, a default number of instructions are disassembled from the dot address.

11.5 Checkpoint commands

`break [load|store|exec] [address [address] [if <cond_expr>]]`

This command allows setting a breakpoint or listing the current breakpoints. If no address is given, the currently valid checkpoints are printed. If an address is given, a breakpoint is set for that address and the breakpoint number is printed. The "load|store|exec" parameter can be either "load", "store" or "exec" (or any combination of these) to determine on which operation the monitor breaks. If not specified, the monitor breaks on "exec". A conditional expression can also be specified for the breakpoint. For more information on conditions, see the CONDITION command.

`enable <checknum>`
`disable <checknum>`

Each checkpoint can be enabled or disabled. This command allows changing between these states.

`command <checknum> "<command>"`

When checkpoint `checknum` is hit, the specified command is executed by the monitor. Note that the `x` command is not yet supported as a command argument.

`condition <checknum> if <cond_expr>`
`cond <checknum> if <cond_expr>`

Each time the specified checkpoint is examined, the condition is evaluated. If it evalutes to true, the checkpoint is activated. Otherwise, it is ignores. If registers are specified in the expression, the values used are those at the time the checkpoint is examined, not when the condition is set.

Currently, the cond_expr is very limited. You can use registers (.A, .X, .Y, .PC, and .SP) and compare against other registers or absolute values. For example, the following are all valid conditions: .A == 0, .X == .Y, 8:.X == .X, .A != 5, .A < .X.

However, you cannot specify memory contents and compare that.

`delete <checknum>`
`del <checknum>`

Delete the specified checkpoint.

`ignore <checknum> [<count>]`

Ignore a checkpoint after a given number of crossings. If no count is given, the default value is 1.

`trace [load|store|exec] [address [address] [if <cond_expr>]]`
`tr [load|store|exec] [address [address] [if <cond_expr>]]`

This command is similar to the `break` command except that it operates on tracepoints. A tracepoint differs from a breakpoint by not stopping execution but simply printing the PC, giving the user an execution trace. The second optional address can be used to specify the end of an range of addresses to be traced. If no addresses are given, a list of all the checkpoints is printed. The "load|store|exec" parameter can be either "load", "store" or "exec" (or any

combination of these) to determine which operation the monitor traces. If not specified, the monitor traces all operations. A conditional expression can also be specified for the tracepoint. For more information on conditions, see the CONDITION command.

`until [<address>]`
`un [<address>]`
If no address is given, the currently valid breakpoints are printed. If an address is given, a temporary breakpoint is set for that address and the breakpoint number is printed. Control is returned to the emulator by this command. The breakpoint is deleted once it is hit.

`watch [load|store|exec] [address [address] [if <cond_expr>]]`
`w [load|store|exec] [address [address] [if <cond_expr>]]`
This command is similar to the `break` command except that it operates on watchpoints. A watchpoint differs from a breakpoint by stopping on a read and/or write to an address or range of addresses. If no addresses are given, a list of all the checkpoints is printed. The "load|store|exec" parameter can be either "load", "store" or "exec" (or any combination of these) to determine on which operation the monitor breaks. If not specified, the monitor breaks on "load" and "store" operations. A conditional expression can also be specified for the watchpoint. For more information on conditions, see the CONDITION command.

11.6 General commands

`cd <directory>`
Change the working directory.

`device [c:|d:]`
`dev [c:|d:]`
Set the default memory device to either the computer (`c:`) or the disk (`d:`).

`dir [<directory>]`
`ls [<directory>]`
Display the directory contents.

`pwd` Show current working directory.

`radix [H|D|O|B]`
`rad [H|D|O|B]`
Set the default radix to hex, decimal, octal, or binary. With no argument, the current radix is printed.

11.7 Disk commands

`attach <filename> <device>`
Attach file to device. (device 32 = cart)

`block_read <track> <sector> [<address>]`
`br <track> <sector> [<address>]`
Read the block at the specified track and sector. If an address is specified, the data is loaded into memory. If no address is given, the data is displayed using the default datatype.

`block_write <track> <sector> <address>`
`bw <track> <sector> <address>`
Write a block of data at `address` to the specified track and sector of disk in drive 8.

`detach <device>`
Detach file from device. (device 32 = cart)

`@<disk command>`
Perform a disk command on the currently attached disk image on drive 8. The specified disk command is sent to the drive's channel #15.

`load "<filename>" <device> [<address>]`
`l "<filename>" <device> [<address>]`
Load the specified file into memory. If no address is given, the file is loaded to the address specified by the first two bytes read from the file. If address is given, the file is loaded to the specified address and the first two bytes read from the file are skipped. If device is 0, the file is read from the file system.

`list [<directory>]`
List disk contents.

`bload "<filename>" <device> <address>`
`bl "<filename>" <device> <address>`
Load the specified file into memory at the specified address. If device is 0, the file is read from the file system.

`save "<filename>" <device> <address1> <address2>`
`s "<filename>" <device> <address1> <address2>`
Save the memory from address1 to address2 to the specified file. Write two-byte load address. If device is 0, the file is written to the file system.

`bsave "<filename>" <device> <address1> <address2>`
`bs "<filename>" <device> <address1> <address2>`
Save the memory from address1 to address2 to the specified file. If device is 0, the file is written to the file system.

11.8 Command file commands

`playback "<filename>"`
`pb "<filename>"`
Monitor commands from the specified file are read and executed. This command stops at the end of file or when a STOP command is read.

`record "<filename>"`
`rec "<filename>"`
: After this command, all commands entered are written to the specified file until the STOP command is entered.

`stop`
: Stop recording commands. See `record`.

11.9 Label commands

`add_label <address> <label>`
`al <address> <label>`
: Map a given address to a label. This label can be used when entering assembly code and is shown during disassembly. Additionally, it can be used whenever an address must be specified.

  <label> is the name of the label; it must start with a dot (".") in order for the monitor to recognize it as a label.

`delete_label [<memspace>] <label>`
`dl [<memspace>] <label>`
: Remove the specified label from the label tables. If no memory space is checked, all tables are checked.

`load_labels [<memspace>] "<filename>"`
`ll [<memspace>] "<filename>"`
: Load a file containing a mapping of labels to addresses. If no memory space is specified, the default readspace is used.

  The file must contain commands the monitor understands, e.g. add_label. The compiler cc65 can create such label files.

  Vice can also load label files created by the Acme assembler. Their syntax is e.g. "labelname = $1234 ; Maybe a comment". A dot will be added automatically to label names assigned in this way to fit to the Vice label syntax. Normally the semicolon seperates commands but after an assignment of this kind it may be used to start a comment to end of line, so unchanged Acme label files can be fed into Vice.

`save_labels [<memspace>] "<filename>"`
`sl [<memspace>] "<filename>"`
: Save labels to a file. If no memory space is specified, all of the labels are saved.

`show_labels [<memspace>]`
`shl [<memspace>]`
: Display current label mappings. If no memory space is specified, show all labels.

11.10 Miscellaneous commands

`cartfreeze`
: Use cartridge freeze.

`cpu <type>`
: Specify the type of CPU currently used (6502/z80).

`exit`
`x` Leave the monitor and return to execution.

`export`
`exp` Print out list of attached expansion port devices.

`help [<command>]`
If no argument is given, prints out a list of all available commands. If an argument is given, prints out specific help for that command.

`keybuf "<string>"`
Put the specified string into the keyboard buffer. Note that you can specify specific keycodes by using C-style escaped hexcodes ("\x0a").

`print <expression>`
`p <expression>`
Evaluate the specified expression and output the result.

`resourceget "<resource>"`
`resget "<resource>"`
Displays the value of the `resource`.

`resourceset "<resource>" "<value>"`
`resset "<resource>" "<value>"`
Sets the value of the `resource`.

`screenshot "<filename>" [<format>]`
`scrsh "<filename>" [<format>]`
Take a screenshot. `format`: default = BMP, 1 = PCX, 2 = PNG, 3 = GIF, 4 = IFF.

`tapectrl <command>`
Control the datasette. `command`: 0 = stop, 1 = start, 2 = forward, 3 = rewind, 4 = record, 5 = reset, 6 = reset counter.

`quit` Exit the emulator immediately.

`~ <number>`
Display the specified number in decimal, hex, octal and binary.

12 c1541

VICE is provided with a complete stand-alone disk image maintenance utility, called `c1541`.

You can either invoke it from the command line or from within one of the VICE emulators, using the "Run c1541" command which will open a new `xterm` window with a running `c1541` in it.

The syntax is:

```
c1541 [IMAGE1 [IMAGE2]] [COMMAND1 COMMAND2 ... COMMANDN]
```

`IMAGE1` and `IMAGE2` are disk image names that can be attached before `c1541` starts. `c1541` can handle up to two disk images at the same time by using two virtual built-in

drives, numbered 8 and 9; IMAGE1 (if present) is always attached to drive 8, while IMAGE2 is attached to drive 9.

COMMANDs specified on the command-line all begin with the minus sign (-); if present, c1541 executes them in the same order as they are on the command line and returns a zero error code if they were successful. If any of the COMMANDs fails, c1541 stops and returns a nonzero error code.

If no COMMANDs are specified at all, c1541 enters interactive mode, where you can type commands manually. Commands in interactive mode are the same as commands in batch mode, but do not require a leading -. As with the monitor, file name completion and command line editing with history are provided via GNU readline. Use the command 'quit' or press *C-d* to exit.

12.1 Specifying files in c1541

When accessing CBM DOS files (i.e. files that reside on disk images), c1541 uses a special syntax that lets you access files on both drive 8 and 9. If you prepend the file name with @8: or @9:, you will specified that file is to be found or created on drive 8 and 9, respectively.

For instance,

```
@8:somefile
```

will name file named somefile on unit 8, while

```
@9:somefile
```

will name file named somefile on unit 9.

12.2 Using quotes and backslashes

You can use quotes (") in a command to embed spaces into file names. For instance,

```
read some file
```

will read file some from the disk image and write it into the file system as file, while

```
read "some file"
```

will copy some file into the file system, with the name some file.

The backslash character (\) has a special meaning too: it lets you literally insert the following character no matter what it is. For example,

```
read some\ file
```

will copy file some file into the file system, while

```
read some\ file this\"file
```

will copy some file into the file system with name this"file (with an embedded quote).

12.3 c1541 commands and options

This is a list of the c1541 commands. They are shown in their interactive form, without the leading -. Square brackets [] indicate an optional part, and "<COMMAND>" translates to a disk command according to CBM DOS, like "i0" for example.

[<command>]
: Execute specified CBM DOS command and print the current status of the drive. If no command is specified, just print the status.

`? [<command>]`
: Explain specified command. If no command is specified, list available ones.

`attach <diskimage> [<unit>]`
: Attach `diskimage` to `unit` (default unit is 8).

`block <track> <sector> <disp> [<drive>]`
: Show specified disk block in hex form.

`copy <source1> [<source2> ... <sourceN>] <destination>`
: Copy `source1` ... `sourceN` into destination. If N > 1, `destination` must be a simple drive specifier (`@n:`).

`delete <file1> [<file2> ... <fileN>]`
: Delete the specified files.

`exit`
: Exit (same as `quit`).

`extract`
: Extract all the files to the file system.

`format <diskname,id> [<type> <imagename>] [<unit>]`
: If `unit` is specified, format the disk in unit `unit`. If `type` and `imagename` are specified, create a new image named `imagename`, attach it to unit 8 and format it. `type` is a disk image type, and must be either `x64`, `d64` (both VC1541/2031), `g64` (VC1541/2031 but in GCR coding), `d71` (VC1571), `d81` (VC1581), `d80` (CBM8050) or `d82` (CBM8250/1001). Otherwise, format the disk in the current unit, if any.

`gcrformat <diskname,id> <imagename>`
: Create and format a G64 disk image named `imagename`.

`help [<command>]`
: Explain specified command. If no command is specified, list available ones.

`info [<unit>]`
: Display information about unit `unit` (if unspecified, use the current one).

`list [<pattern>]`
: List files matching `pattern` (default is all files).

`quit`
: Exit (same as `exit`).

`read <source> [<destination>]`
: Read `source` from the disk image and copy it into `destination` in the file system. If `destination` is not specified, copy it into a file with the same name as `source`.",

`rename <oldname> <newname>`
: Rename `oldname` into `newname`. The files must be on the same drive.

`tape <t64name> [<file1> ... <fileN>]`
: Extract files from a T64 image.

`unit <number>`
: Make unit `number` the current unit.

`unlynx <lynxname> [<unit>]`
: Extract the specified Lynx image file into the specified unit (default is the current unit).

`validate [<unit>]`
: Validate the disk in unit `unit`. If `unit` is not specified, validate the disk in the current unit.

`write <source> [<destination>]`
: Write `source` from the file system into `destination` on a disk image.

`zcreate <x64name> <zipname> [<label,id>]`
: Create an X64 disk image out of a set of four Zipcoded files named `1!zipname`, `2!zipname`, `3!zipname` and `4!zipname`.

12.4 Executing shell commands

If you want to execute a shell command from withing `c1541`, just prepend it with an exclamation mark (`!`). For example,

```
!ls -la
```

will execute the command `ls -la`, which will show you all the files in the current directory.

12.5 c1541 examples

`c1541 -attach test.d64 -write test.prg testfile`
: Write `test.prg` to `test.d64` as `testfile`.

13 cartconv

The cartconv program is a cartridge conversion utility, it can convert between binary and .crt images and it can 'insert' binary and/or .crt images into the EPROM type of cartridges.

13.1 cartconv command line options

The cartconv program has the following parameters:

`-i "input name"`
: This parameter is mandatory, it should contain the name of the binary/.crt file you want to convert. For the EPROM type of cartridges this parameter can be used multiple times to insert images into the resulting file.

`-o "output name"`
: This parameter is mandatory, it should contain the name of the binary/.crt file you want to convert the input file to.

`-t carttype`
: This parameter is optional. It is only needed when converting to a .crt file. See below for the supported cartridge types.

`-n "cart name"`
: This parameter is optional and is used as the cartridge name when creating a .crt file.

`-l loadaddress`
: This parameter is optional and is used as the load-address when converting a .crt file to a .prg file, or when converting to a generic type .crt file.

`-f "input name"`
: This parameter is optional, and is meant to output information about the named file. It can't be used in conjuction with any of the other parameters.

`-r`
: This parameter is optional, it enables repair mode (accept broken input files)

The following cartridge types are supported:

`bin`	Binary .bin file (Default crt->bin)
`normal`	Generic 8kB/12kB/16kB .crt file (Default bin->crt)
`prg`	Binary C64 .prg file with load-address
`ulti`	Ultimax mode 4kB/8kB/16kB .crt file
`ap`	Atomic Power .crt file
`ar2`	Action Replay MK2 .crt file
`ar3`	Action Replay MK3 .crt file
`ar4`	Action Replay MK4 .crt file
`ar5`	Action Replay V5 .crt file
`cap`	Capture .crt file
`comal`	Comal 80 .crt file
`dep256`	Dela EP256 .crt file, extra files can be inserted (1)(2)
`dep64`	Dela EP64 .crt file, extra files can be inserted (1)
`dep7x8`	Dela EP7x8 .crt file, extra files can be inserted (1)(2)(3)
`din`	Dinamic .crt file
`dsm`	Diashow-Maker .crt file
`easy`	EasyFlash .crt file
`epyx`	Epyx FastLoad .crt file
`exos`	EXOS .crt file
`expert`	Expert Cartridge .crt file
`fc1`	The Final Cartridge .crt file
`fc3`	The Final Cartridge III .crt file
`fcp`	Final Cartridge Plus .crt file
`ff`	Freeze Frame .crt file

`fm`	Freeze Machine .crt file
`fp`	Fun Play .crt file
`gk`	Game Killer .crt file
`gs`	C64 Games System .crt file
`ide64`	IDE64 .crt file
`ieee`	IEEE-488 Interface .crt file
`kcs`	KCS Power Cartridge .crt file
`mach5`	MACH 5 .crt file
`md`	Magic Desk .crt file
`mf`	Magic Formel .crt file
`mikro`	Mikro Assembler .crt file
`mmc64`	MMC64 .crt file
`mmcr`	MMC Replay .crt file
`mv`	Magic Voice .crt file
`ocean`	Ocean .crt file
`p64`	Prophet64 .crt file
`rep256`	REX 256k EPROM Cart .crt file, extra files can be inserted (1)(2)(3)
`ross`	ROSS .crt file
`rr`	Retro Replay .crt file
`ru`	REX Utility .crt file
`s64`	Snapshot 64 .crt file
`sb`	Structured BASIC .crt file
`se5`	Super Explode V5.0 .crt file
`sg`	Super Games .crt file
`simon`	Simons' BASIC .crt file
`ss4`	Super Snapshot V4 .crt file
`ss5`	Super Snapshot V5 .crt file
`star`	Stardos .crt file
`wl`	Westermann Learning .crt file
`ws`	Warp Speed .crt file
`zaxxon`	Zaxxon .crt file

- (1) insertion of 32kB EPROM files supported.
- (2) insertion of 8kB .crt/binary files supported.
- (3) insertion of 16kB .crt/binary files supported.

13.2 cartconv examples

`cartconv -i foo.crt -o foo.bin`
Convert a .crt file to a binary file with no load-address.

`cartconv -t prg -i foo.crt -o foo.prg`
Convert a .crt file to a .prg file with default load-address.

`cartconv -t prg -l 49152 -i foo.crt -o foo.prg`
Convert a .crt file to a .prg file with 49152 as the load-address.

`cartconv -t ocean -i foo.bin -o foo.crt`
Convert a binary file to an ocean type cartridge.

`cartconv -t dep64 -i dep64.bin -i eprom.prg -o foo.crt`
Inserting a 32kB EPROM file into an dep64 type cartridge.

- step 1 : use the dep64 binary file in VICE as a generic 8kB cartridge.
- step 2 : generate an EPROM file.
- step 3 : get the EPROM file to the host computer.
- step 4 : insert the EPROM file into the final dep64 .crt file:

`cartconv -t dep256 -i dep256.bin -i somegame.crt -o foo.crt`
Insert an 8kB .crt file into a dep256 type cartridge.

`cartconv -t rep256 -i rep256.bin -i foo1.crt -i foo2.crt -i foo3.crt -o foo.crt`
Insert multiple 8kB .crt files into a rep256 type cartridge.

`cartconv -f foo.crt`
Get information about a .crt file.

14 petcat

The petcat program is a text conversion utility, it can convert between ASCII, PETSCII and tokenized BASIC.

14.1 petcat command line options

`-help` Output help text

`-v` Same as above

`-c` controls (interpret also control codes) (default if textmode)

`-nc` no controls (suppress control codes in printout) (default if non-textmode)

`-ic` interpret control codes case-insensitive

`-h` write header (default if output is stdout)

`-nh` no header (default if output is a file)

`-skip <n>` Skip `<n>` bytes in the beginning of input file. Ignored on P00.

`-text` Force text mode

`-<version>`
: use keywords for <version> instead of the v7.0 ones

`-w<version>`
: tokenize using keywords on specified Basic version.

`-k<version>`
: list all keywords for the specified Basic version

`-k` list all Basic versions available.

`-l` Specify load address for program (in hex, no loading chars!).

`-o <name>` Specify the output file name

`-f` Force overwritten the output file. The default depends on the BASIC version.

BASIC Versions:

`1` PET Basic V1.0

`2` Basic v2.0

`superexp` Basic v2.0 with Super Expander (VIC20)

`turtle` Basic v2.0 with Turtle Basic by Craig Bruce (VIC20)

`mighty` Basic v2.0 with Mighty Basic by Craig Bruce (VIC20)

`a` Basic v2.0 with AtBasic (C64)

`simon` Basic v2.0 with Simon's Basic extension (C64)

`speech` Basic v2.0 with Speech Basic v2.7 (C64)

`F` Basic v2.0 with Final Cartridge III (C64)

`ultra` Basic v2.0 with Ultrabasic-64 (C64)

`graph` Basic v2.0 with Graphics basic (C64)

`WSB` Basic v2.0 with WS basic (C64)

`WSBF` Basic v2.0 with WS basic final (C64)

`Pegasus` Basic v2.0 with Pegasus basic 4.0 (C64)

`Xbasic` Basic v2.0 with Xbasic (C64)

`Drago` Basic v2.0 with Drago basic 2.2 (C64)

`REU` Basic v2.0 with REU-basic (C64)

`Lightning`
: Basic v2.0 with Basic Lightning (C64)

`magic` Basic v2.0 with Magic Basic (C64)

`easy` Basic v2.0 with Easy Basic (VIC20)

`blarg` Basic v2.0 with Blarg (C64)

`Game` Basic v2.0 with Game Basic (C64)

`BSX`	Basic v2.0 with Basex (C64)
`superbas`	Basic v2.0 with Super Basic (C64)
`exp20`	Basic 2.0 with Expanded Basic (VIC20)
`exp64`	Basic 2.0 with Expanded Basic (C64)
`sxc`	Basic 2.0 with Super Expander Chip (C64)
`warsaw`	Basic 2.0 with Warsaw Basic (C64)
`4v`	Basic 2.0 with Basic 4.0 extensions (VIC20)
`4 -w4e`	PET Basic v4.0 program (PET/C64)
`5`	Basic 2.0 with Basic 5.0 extensions (VIC20)
`3`	Basic v3.5 program (C16)
`70`	Basic v7.0 program (C128)
`71`	Basic v7.1 program (C128)
`10`	Basic v10.0 program (C64DX)

14.2 petcat examples

`petcat -2 -o outputfile.txt -- inputfile.prg`
: Convert inputfile.prg to a text file in outputfile.txt, using BASIC V2 only

`petcat -wsimon -o outputfile.prg -- inputfile.txt`
: Convert inputfile.txt to a PRG file in outputfile.prg, using Simon's BASIC

15 The emulator file formats

This chapter gives a technical description of the various files supported by the emulators.

15.1 The T64 tape image format

(This section was taken from the C64S distribution.)

The `T64` File Structure was developed by Miha Peternel for use in the C64S emulator. It is easy to use and allows future extensions.

15.1.1 T64 File structure

Offset	Size	Description
0	64	tape record
64	32*n	file records for n directory entries
64+32*n	varies	binary contents of the files

15.1.2 Tape Record

Offset	Size	Description
0	32	DOS tape description + EOF (for type)
32	2	tape version ($0200)
34	2	number of directory entries
36	2	number of used entries (can be 0 in my loader)
38	2	free
40	24	user description as displayed in tape menu

15.1.3 File record

Offset	Size	Description
0	1	entry type (see below)
1	1	C64 file type
2	2	start address
4	2	end address
6	2	free
8	4	offset of file contents start within T64 file
12	4	free
16	16	C64 file name

Valid entry types are:

Code	Explanation
0	free entry
1	normal tape file
2	tape file with header: header is saved just before file data
3	memory snapshot v0.9, uncompressed
4	tape block
5	digitized stream
6 ... 255	reserved

Notes:

- VICE only supports file type 1.
- Types 3, 4 and 5 are subject to change (and are rarely used).

15.2 The G64 GCR-encoded disk image format

(This section was contributed by Peter Schepers and slightly edited by Ettore Perazzoli.)

This format was defined in 1998 as a cooperative effort between several emulator people, mainly Per Hkan Sundell, author of the CCS64 C64 emulator, Andreas Boose of the VICE CBM emulator team and Joe Forster/STA, the author of Star Commander. It was the first real public attempt to create a format for the emulator community which removed almost all of the drawbacks of the other existing image formats, namely D64.

The intention behind G64 is not to replace the widely used D64 format, as D64 works fine with the vast majority of disks in existence. It is intended for those small percentage of programs which demand to work with the 1541 drive in a non-standard way, such as reading or writing data in a custom format. The best example is with speeder software such as Action Cartridge in Warp Save mode or Vorpal which write track/sector data in another format other than standard GCR. The other obvious example is copy-protected software which looks for some specific data on a track, like the disk ID, which is not stored in a standard D64 image.

G64 has a deceptively simply layout for what it is capable of doing. We have a signature, version byte, some predefined size values, and a series of offsets to the track data and speed zones. It is what's contained in the track data areas and speed zones which is really at the heart of this format.

Each track entry in simply the raw stream of GCR data, just what a read head would see when a diskette is rotating past it. How the data gets interpreted is up to the program trying to access the disk. Because the data is stored in such a low-level manner, just about anything can be done. Most of the time I would suspect the data in the track would be standard sectors, with SYNC, GAP, header, data and checksums. The arrangement of the data when it is in a standard GCR sector layout is beyond the scope of this document.

Since it is a flexible format in both track count and track byte size, there is no "standard" file size. However, given a few constants like 42 tracks and halftracks, a track size of 7928 bytes and no speed offset entries, the typical file size will a minimum of 333744 bytes.

Below is a dump of the header, broken down into its various parts. After that will be an explanation of the track offset and speed zone offset areas, as they demand much more explanation.

```
Addr  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
----  -----------------------------------------------
0000: 47 43 52 2D 31 35 34 31 00 54 F8 1E .. .. .. ..
```

Offset	Description
$0000-0007	File signature (GCR-1541)
$0008	G64 version (presently only $00 defined)
$0009	Number of tracks in image (usually $54, decimal 84)
$000A-000B	Size of each stored track in bytes (usually 7928, or $1EF8) in LO/HI format.

An obvious question here is "why are there 84 tracks defined when a normal D64 disk only has 35 tracks?" Well, by definition, this image includes all half-tracks, so there are actually 42 tracks and 42 half tracks. The 1541 stepper motor can access up to 42 tracks and the in-between half-tracks. Even though using more than 35 tracks is not typical, it was important to define this format from the start with what the 1541 is capable of doing, and not just what it typically does.

At first, the defined track size value of 7928 bytes may seem to be arbitrary, but it is not. It is determined by the fastest write speed possible (speed zone 0), coupled with the average rotation speed of the disk (300 rpm). After some math, the answer that actually comes up is 7692 bytes. Why the discrepency between the actual size of 7692 and the defined size of 7928? Simply put, not all drives rotate at 300 rpm. Some can be faster or slower, so a upper safety margin of +3% was built added, in case some disks rotate slower and can write

more data. After applying this safety factor, and some rounding-up, 7928 bytes per track was arrived at.

Also note that this upper limit of 7928 bytes per track really only applies to 1541 and compatible disks. If this format were applied to another disk type like the SFD1001, this value would be higher.

Below is a dump of the first section of a `G64` file, showing the offsets to the data portion for each track and half-track entry. Following that is a dump of the speed zone offsets.

```
Addr  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
----  -----------------------------------------------
0000: .. .. .. .. .. .. .. .. .. .. .. .. AC 02 00 00
0010: 00 00 00 00 A6 21 00 00 00 00 00 00 A0 40 00 00
0020: 00 00 00 00 9A 5F 00 00 00 00 00 00 94 7E 00 00
0030: 00 00 00 00 8E 9D 00 00 00 00 00 00 88 BC 00 00
0040: 00 00 00 00 82 DB 00 00 00 00 00 00 7C FA 00 00
0050: 00 00 00 00 76 19 01 00 00 00 00 00 70 38 01 00
0060: 00 00 00 00 6A 57 01 00 00 00 00 00 64 76 01 00
0070: 00 00 00 00 5E 95 01 00 00 00 00 00 58 B4 01 00
0080: 00 00 00 00 52 D3 01 00 00 00 00 00 4C F2 01 00
0090: 00 00 00 00 46 11 02 00 00 00 00 00 40 30 02 00
00A0: 00 00 00 00 3A 4F 02 00 00 00 00 00 34 6E 02 00
00B0: 00 00 00 00 2E 8D 02 00 00 00 00 00 28 AC 02 00
00C0: 00 00 00 00 22 CB 02 00 00 00 00 00 1C EA 02 00
00D0: 00 00 00 00 16 09 03 00 00 00 00 00 10 28 03 00
00E0: 00 00 00 00 0A 47 03 00 00 00 00 00 04 66 03 00
00F0: 00 00 00 00 FE 84 03 00 00 00 00 00 F8 A3 03 00
0100: 00 00 00 00 F2 C2 03 00 00 00 00 00 EC E1 03 00
0110: 00 00 00 00 E6 00 04 00 00 00 00 00 E0 1F 04 00
0120: 00 00 00 00 DA 3E 04 00 00 00 00 00 D4 5D 04 00
0130: 00 00 00 00 CE 7C 04 00 00 00 00 00 C8 9B 04 00
0140: 00 00 00 00 C2 BA 04 00 00 00 00 00 BC D9 04 00
0150: 00 00 00 00 B6 F8 04 00 00 00 00 00 .. .. .. ..
```

Offset	**Description**
$000C-000F	Offset to stored track 1.0 ($000002AC, in LO/HI format, see below for more)
$0010-0013	Offset to stored track 1.5 ($00000000)
$0014-0017	Offset to stored track 2.0 ($000021A6)
...	
$0154-0157	Offset to stored track 42.0 ($0004F8B6)
$0158-015B	Offset to stored track 42.5 ($00000000)

```
      00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
      -----------------------------------------------
0150: .. .. .. .. .. .. .. .. .. .. .. .. 03 00 00 00
0160: 00 00 00 00 03 00 00 00 00 00 00 00 03 00 00 00
0170: 00 00 00 00 03 00 00 00 00 00 00 00 03 00 00 00
0180: 00 00 00 00 03 00 00 00 00 00 00 00 03 00 00 00
0190: 00 00 00 00 03 00 00 00 00 00 00 00 03 00 00 00
```

```
01A0: 00 00 00 00 03 00 00 00 00 00 00 00 03 00 00 00
01B0: 00 00 00 00 03 00 00 00 00 00 00 00 03 00 00 00
01C0: 00 00 00 00 03 00 00 00 00 00 00 00 03 00 00 00
01D0: 00 00 00 00 03 00 00 00 00 00 00 00 03 00 00 00
01E0: 00 00 00 00 02 00 00 00 00 00 00 00 02 00 00 00
01F0: 00 00 00 00 02 00 00 00 00 00 00 00 02 00 00 00
0200: 00 00 00 00 02 00 00 00 00 00 00 00 02 00 00 00
0210: 00 00 00 00 02 00 00 00 00 00 00 00 01 00 00 00
0220: 00 00 00 00 01 00 00 00 00 00 00 00 01 00 00 00
0230: 00 00 00 00 01 00 00 00 00 00 00 00 01 00 00 00
0240: 00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00
0250: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0260: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0270: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0280: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0290: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
02A0: 00 00 00 00 00 00 00 00 00 00 00 00 .. .. .. ..
```

Offset	Description
$015C-015F	Speed zone entry for track 1 ($03, in LO/HI format, see below for more)
$0160-0163	Speed zone entry for track 1.5 ($03)
...	
$02A4-02A7	Speed zone entry for track 42 ($00)
$02A8-02AB	Speed zone entry for track 42.5 ($00)

Starting here at $02AC is the first track entry (from above, it is the first entry for track 1.0)

The track offsets (from above) require some explanation. When one is set to all 0's, no track data exists for this entry. If there is a value, it is an absolute reference into the file (starting from the beginning of the file). From the track 1.0 entry we see it is set for $000002AC. Going to that file offset, here is what we see...

```
       00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
       -----------------------------------------------
02A0: .. .. .. .. .. .. .. .. .. .. .. .. 0C 1E FF FF
02B0: FF FF FF 52 54 B5 29 4B 7A 5E 95 55 55 55 55 55
02C0: 55 55 55 55 55 55 FF FF FF FF FF 55 D4 A5 29 4A
02D0: 52 94 A5 29 4A 52 94 A5 29 4A 52 94 A5 29 4A 52
```

Offset	Description
$02AC-02AD	Actual size of stored track (7692 or $1E0C, in LO/HI format)
$02AE-02AE+$1E0C	Track data

Following the track data is filler bytes. In this case, there are 368 bytes of unused space. This space can contain anything, but for the sake of those wishing to compress these images for storage, they should all be set to the same value. In the sample I used, these are all set to $FF.

Below is a dump of the end of the track 1.0 data area. Note the actual track data ends at address $20B9, with the rest of the block being unused, and set to $FF.

```
        00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
        -----------------------------------------------
1FE0: 52 94 A5 29 4A 52 94 A5 29 4A 52 94 A5 29 4A 52
1FF0: 94 A5 29 4A 52 94 A5 29 4A 52 94 A5 29 4A 52 94
2000: A5 29 4A 52 94 A5 29 4A 52 94 A5 29 4A 52 94 A5
2010: 29 4A 52 94 A5 29 4A 52 94 A5 29 4A 52 94 A5 29
2020: 4A 52 94 A5 29 4A 52 94 A5 29 4A 52 94 A5 29 4A
2030: 55 55 55 55 55 55 FF FF FF FF FF FF FF FF FF FF
2040: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
2050: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
2060: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
2070: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
2080: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
2090: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
20A0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
20B0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
20C0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
20D0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
20E0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
20F0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
2100: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
2110: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
2120: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
2130: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
2140: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
2150: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
2160: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
2170: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
2180: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
2190: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
21A0: FF FF FF FF FF FF .. .. .. .. .. .. .. .. .. ..
```

The speed offset entries can be a little more complex. The 1541 has four speed zones defined, which means the drive can write data at four distinct speeds. On a normal 1541 disk, these zones are as follows:

Track Range	**Speed Zone**
1-17	3 (highest writing speed)
18-24	2
25-30	1
31 and up	0 (lowest writing speed)

Note that you can, through custom programming of the 1541, change the speed zone of any track to something different (change the 3 to a 0) and write data differently. From the dump of the speed offset entries above, we see that all the entries are in the range of 0-3. If any entry is less than 4, this is not considered a speed offset but defines the whole track to be recorded at that one speed.

In the example I had, there were no offsets defined, so no speed zone dump can be shown. However, I can define what should be there. You will have a block of data, 1982 bytes long. Each byte is encoded to represent the speed of 4 bytes in the track offset area, and is broken down as follows:

```
Speed entry $FF:  in binary %11111111
                             |'|'|'|'
                             | | | |
                             | | | +- 4'th byte speed (binary 11, 3 dec)
                             | | +--- 3'rd byte speed (binary 11, 3 dec)
                             | +----- 2'nd byte speed (binary 11, 3 dec)
                             +------- 1'st byte speed (binary 11, 3 dec)
```

It was very smart thinking to allow for two speed zone settings, one in the offset block and another defining the speed on a per-byte basis. If you are working with a normal disk, where each track is one constant speed, then you don't need the extra blocks of information hanging around the image, wasting space.

What may not be obvious is the flexibility of this format to add tracks and speed offset zones at will. If a program decides to write a track out with varying speeds, and no speed offset exist, a new block will be created by appending it to the end of the image, and the offset pointer for that track set to point to the new block. If a track has no offset yet, meaning it doesn't exist (like a half-track), and one needs to be added, the same procedure applies. The location of the actual track or speed zone data is not important, meaning they do not have to be in any particular order since they are all referenced by the offsets at the beginning of the image.

15.3 The D64 disk image format

(This section was contributed by Peter Schepers and slightly edited by Marco van den Heuvel.)

First and foremost we have D64, which is basically a sector-for-sector copy of a 1540/1541 disk. There are several versions of these which I will cover shortly. The standard D64 is a 174848 byte file comprised of 256 byte sectors arranged in 35 tracks with a varying number of sectors per track for a total of 683 sectors. Track counting starts at 1, not 0, and goes up to 35. Sector counting starts at 0, not 1, for the first sector, therefore a track with 21 sectors will go from 0 to 20.

The original media (a 5.25" disk) has the tracks laid out in circles, with track 1 on the very outside of the disk (closest to the sides) to track 35 being on the inside of the disk (closest to the inner hub ring). Commodore, in their infinite wisdom, varied the number of sectors per track and data densities across the disk to optimize available storage, resulting in the chart below. It shows the sectors/track for a standard D64. Since the outside diameter of a circle is the largest (versus closer to the center), the outside tracks have the largest amount of storage.

Track	**Sectors/track**	**# Sectors**
1-17	21	357
18-24	19	133
25-30	18	108
31-35	17	85

36-40(*)	17	85	
Track	**#Sect**	**#SectorsIn**	**D64 Offset**
1	21	0	$00000
2	21	21	$01500
3	21	42	$02A00
4	21	63	$03F00
5	21	84	$05400
6	21	105	$06900
7	21	126	$07E00
8	21	147	$09300
9	21	168	$0A800
10	21	189	$0BD00
11	21	210	$0D200
12	21	231	$0E700
13	21	252	$0FC00
14	21	273	$11100
15	21	294	$12600
16	21	315	$13B00
17	21	336	$15000
18	19	357	$16500
19	19	376	$17800
20	19	395	$18B00
21	19	414	$19E00
22	19	433	$1B100
23	19	452	$1C400
24	19	471	$1D700
25	18	490	$1EA00
26	18	508	$1FC00
27	18	526	$20E00
28	18	544	$22000
29	18	562	$23200
30	18	580	$24400
31	17	598	$25600
32	17	615	$26700
33	17	632	$27800
34	17	649	$28900
35	17	666	$29A00
36(*)	17	683	$2AB00
37(*)	17	700	$2BC00
38(*)	17	717	$2CD00
39(*)	17	734	$2DE00
40(*)	17	751	$2EF00

(*) Tracks 36-40 apply to 40-track images only.

The directory track should be contained totally on track 18. Sectors 1-18 contain the entries and sector 0 contains the BAM (Block Availability Map) and disk name/ID. Since the directory is only 18 sectors large (19 less one for the BAM), and each sector can contain

only 8 entries (32 bytes per entry), the maximum number of directory entries is 18 * 8 = 144. The first directory sector is always 18/1, even though the t/s pointer at 18/0 (first two bytes) might point somewhere else. It then follows the same chain structure as a normal file, using a sector interleave of 3. This makes the chain links go 18/1, 18/4, 18/7 etc.

Note that you can extend the directory off of track 18, but only when reading the disk or image. Attempting to write to a directory sector not on track 18 will cause directory corruption. Each directory sector has the following layout (18/1 partial dump):

```
00: 12 04 81 11 00 4E 41 4D 45 53 20 26 20 50 4F 53 <- notice the T/S link
10: 49 54 A0 A0 A0 00 00 00 00 00 00 00 00 15 00 <- to 18/4 ($12/$04)
20: 00 00 84 11 02 41 44 44 49 54 49 4F 4E 41 4C 20 <- and how its not here
30: 49 4E 46 4F A0 11 0C FE 00 00 00 00 00 00 61 01 <- ($00/$00)
```

The first two bytes of the sector ($12/$04) indicate the location of the next track/sector of the directory (18/4). If the track is set to $00, then it is the last sector of the directory. It is possible, however unlikely, that the directory may *not* be competely on track 18 (some disks do exist like this). Just follow the chain anyhow.

When the directory is done, the track value will be $00. The sector link should contain a value of $FF, meaning the whole sector is allocated, but the actual value doesn't matter. The drive will return all the available entries anyways.

This is a breakdown of a standard directory sector:

Bytes	**Description**
$00-$1F	First directory entry
$20-$3F	Second dir entry
$40-$5F	Third dir entry
$60-$7F	Fourth dir entry
$80-$9F	Fifth dir entry
$A0-$BF	Sixth dir entry
$C0-$DF	Seventh dir entry
$E0-$FF	Eighth dir entry

This is a breakdown of a standard directory entry:

Bytes	**Description**
$00-$01	Track/Sector location of next directory sector ($00 $00 if not the first entry in the sector)
$02	File type
$03-$04	Track/sector location of first sector of file
$05-$14	16 character filename (in PETASCII, padded with $A0)
$15-$16	Track/Sector location of first side-sector block (REL file only)
$17	REL file record length (REL file only, max. value 254)
$18-$1D	Unused (except with GEOS disks)
$1E-$1F	File size in sectors, low/high byte order ($1E+$1F*256). The approx. filesize in bytes is <= #sectors * 254

The file type field is used as follows:

Bits	**Description**
0-3	The actual file type
4	Unused
5	Used only during SAVE- replacement

6	Locked flag (Set produces ">" locked files)
7	Closed flag (Not set produces "*", or "splat" files)

The actual file type can be one of the following:

Binary	Decimal	File type
0000	0	DEL
0001	1	SEQ
0010	2	PRG
0011	3	USR
0100	4	REL

Values 5-15 are illegal, but if used will produce very strange results. The 1541 is inconsistent in how it treats these bits. Some routines use all 4 bits, others ignore bit 3, resulting in values from 0-7.

Files, on a standard 1541, are stored using an interleave of 10. Assuming a starting track/sector of 17/0, the chain would run 17/0, 17/10, 17/20, 17/8, 17/18, etc.

*** Non-Standard & Long Directories

Most Commdore floppy disk drives use a single dedicated directory track where all filenames are stored. This limits the number of files stored on a disk based on the number of sectors on the directory track. There are some disk images that contain more files than would normally be allowed. This requires extending the directory off the default directory track by changing the last directory sector pointer to a new track, allocating the new sectors in the BAM, and manually placing (or moving existing) file entries there. The directory of an extended disk can be read and the files that reside there can be loaded without problems on a real drive. However, this is still a very dangerous practice as writing to the extended portion of the directory will cause directory corruption in the non-extended part. Many of the floppy drives core ROM routines ignore the track value that the directory is on and assume the default directory track for operations.

To explain: assume that the directory has been extended from track 18 to track 19/6 and that the directory is full except for a few slots on 19/6. When saving a new file, the drive DOS will find an empty file slot at 19/6 offset $40 and correctly write the filename and a few other things into this slot. When the file is done being saved the final file information will be written to 18/6 offset $40 instead of 19/6 causing some directory corruption to the entry at 18/6. Also, the BAM entries for the sectors occupied by the new file will not be saved and the new file will be left as a SPLAT (*) file.

Attempts to validate the disk will result in those files residing off the directory track to not be allocated in the BAM, and could also send the drive into an endless loop. The default directory track is assumed for all sector reads when validating so if the directory goes to 19/6, then the validate code will read 18/6 instead. If 18/6 is part of the normal directory chain then the validate routine will loop endlessly.

*** BAM layout

The layout of the BAM area (sector 18/0) is a bit more complicated...

```
      00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
      -----------------------------------------------
00: 12 01 41 00 12 FF F9 17 15 FF FF 1F 15 FF FF 1F
10: 15 FF FF 1F 12 FF F9 17 00 00 00 00 00 00 00 00
```

```
20: 00 00 00 00 0E FF 74 03 15 FF FF 1F 15 FF FF 1F
30: 0E 3F FC 11 07 E1 80 01 15 FF FF 1F 15 FF FF 1F
40: 15 FF FF 1F 15 FF FF 1F 0D C0 FF 07 13 FF FF 07
50: 13 FF FF 07 11 FF CF 07 13 FF FF 07 12 7F FF 07
60: 13 FF FF 07 0A 75 55 01 00 00 00 00 00 00 00 00
70: 00 00 00 00 00 00 00 00 01 08 00 00 03 02 48 00
80: 11 FF FF 01 11 FF FF 01 11 FF FF 01 11 FF FF 01
90: 53 48 41 52 45 57 41 52 45 20 31 20 20 A0 A0 A0
A0: A0 A0 56 54 A0 32 41 A0 A0 A0 A0 00 00 00 00 00
B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Bytes	**Description**
$00-$01	Track/Sector location of the first directory sector (should be set to 18/1 but it doesn't matter, and don't trust what is there, always go to 18/1 for first directory entry)
$02	Disk DOS version type (see note below) $41 ("A")
$03	Unused
$04-$8F	BAM entries for each track, in groups of four bytes per track, starting on track 1 (see below for more details)
$90-$9F	Disk Name (padded with $A0)
$A0-$A1	Filled with $A0
$A2-$A3	Disk ID
$A4	Usually $A0
$A5-$A6	DOS type, usually "2A"
$A7-$AA	Filled with $A0
$AB	Unused ($00)
$AC-$BF	For DOLPHIN DOS track 36-40 BAM entries, otherwise unused ($00)
$C0-$D3	For SPEED DOS track 36-40 BAM entries, otherwise unused ($00)
$D4-$FF	Unused ($00)

Note: The BAM entries for SPEED, DOLPHIN and ProLogic DOS use the same layout as standard BAM entries. One of the interesting things from the BAM sector is the byte at offset $02, the DOS version byte. If it is set to anything other than $41 or $00, then we have what is called "soft write protection". Any attempt to write to the disk will return the "DOS Version" error code 73 ,"CBM DOS V 2.6 1541". The 1541 is simply telling you that it thinks the disk format version is incorrect. This message will normally come up when you first turn on the 1541 and read the error channel. If you write a $00 or a $41 into 1541 memory location $00FF (for device 0), then you can circumvent this type of write-protection, and change the DOS version back to what it should be.

The BAM entries require a bit (no pun intended) more of a breakdown. Take the first entry at bytes $04-$07 ($12 $FF $F9 $17). The first byte ($12) is the number of free sectors on that track. Since we are looking at the track 1 entry, this means it has 18 (decimal) free sectors. The next three bytes represent the bitmap of which sectors are used/free. Since

it is 3 bytes (8 bits/byte) we have 24 bits of storage. Remember that at most, each track only has 21 sectors, so there are a few unused bits.

Bytes	**Data**	**Description**
$04-$07	$12 $FF $F9 $17	Track 1 BAM
$08-$0B	$15 $FF $FF $FF	Track 2 BAM
$0C-$0F	$15 $FF $FF $1F	Track 3 BAM
...	...	...
$8C-$8F	$11 $FF $FF $01	Track 35 BAM

These entries must be viewed in binary to make any sense. We will use the first entry (track 1) at bytes 04-07:

```
FF=11111111, F9=11111001, 17=00010111
```

In order to make any sense from the binary notation, flip the bits around.

```
           111111 11112222
01234567 89012345 67890123
-------------------------
11111111 10011111 11101000
^                     ^

sector 0           sector 20
```

Since we are on the first track, we have 21 sectors, and only use up to the bit 20 position. If a bit is on (1), the sector is free. Therefore, track 1 has sectors 9, 10 and 19 used, all the rest are free. Any leftover bits that refer to sectors that don't exist, like bits 21-23 in the above example, are set to allocated.

Each filetype has its own unique properties, but most follow one simple structure. The first file sector is pointed to by the directory and follows a t/s chain, until the track value reaches $00. When this happens, the value in the sector link location indicates how much of the sector is used. For example, the following chain indicates a file 6 sectors long, and ends when we encounter the $00/$34 chain. At this point the last sector occupies from bytes $02-$34.

1	2	3	4	5	6
—-	—-	—-	—-	—-	—-
17/0	17/10	17/20	17/1	17/11	0/52
(11/00)	(11/0A)	(11/14)	(11/01)	(11/0B)	(0/34)

*** Variations on the D64 layout

These are some variations of the D64 layout:

1. Standard 35 track layout but with 683 error bytes added on to the end of the file. Each byte of the error info corresponds to a single sector stored in the D64, indicating if the sector on the original disk contained an error. The first byte is for track 1/0, and the last byte is for track 35/16.

2. A 40 track layout, following the same layout as a 35 track disk, but with 5 extra tracks. These contain 17 sectors each, like tracks 31-35. Some of the PC utilities do allow you to create and work with these files. This can also have error bytes attached like variant #1.

3. The Commodore 128 allowed for "auto-boot" disks. With this, t/s 1/0 holds a specific byte sequence which the computer recognizes as boot code.

Below is a small chart detailing the standard file sizes of D64 images, 35 or 40 tracks, with or without error bytes.

Disk type	**Size**
35 track, no errors	174848
35 track, 683 error bytes	175531
40 track, no errors	196608
40 track, 768 error bytes	197376

The following table (provided by Wolfgang Moser) outlines the differences between the standard 1541 DOS and the various "speeder" DOS's that exist. The 'header 7/8' category is the 'fill bytes' as the end of the sector header of a real 1541 disk.

Disk format	**tracks**	**header 7/8**	**Dos type**	**Diskdos type**	**vs.**
Original CBM DOS v2.6	35	$0f $0f	"2A"	$41/'A'	
*SpeedDOS+	40	$0f $0f	"2A"	$41/'A'	
Professional DOS Initial	35	$0f $0f	"2A"	$41/'A'	
Professional DOS Version 1/Prototype	40	$0f $0f	"2A"	$41/'A'	
ProfDOS Release	40	$0f $0f	"4A"	$41/'A'	
Dolphin-DOS 2.0/3.0	35	$0f $0f	"2A"	$41/'A'	
Dolphin-DOS 2.0/3.0	40	$0d $0f	"2A"	$41/'A'	
PrologicDOS 1541	35	$0f $0f	"2A"	$41/'A'	
PrologicDOS 1541	40	$0f $0f	"2P"	$50/'P'	
ProSpeed 1571 2.0	35	$0f $0f	"2A"	$41/'A'	
ProSpeed 1571 2.0	40	$0f $0f	"2P"	$50/'P'	

*Note: There are also clones of SpeedDOS that exist, such as RoloDOS and DigiDOS. Both are just a change of the DOS startup string.

The location of the extra BAM information in sector 18/0, for 40 track images, will be different depending on what standard the disks have been formatted with. SPEED DOS stores them from $C0 to $D3, DOLPHIN DOS stores them from $AC to $BF and PrologicDOS stored them right after the existing BAM entries from $90-A3. PrologicDOS also moves the disk label and ID forward from the standard location of $90 to $A4. 64COPY and Star Commander let you select from several different types of extended disk formats you want to create/work with.

All three of the speeder DOS's mentioned above don't alter the standard sector interleave of 10 for files and 3 for directories. The reason is that they use a memory cache installed in the drive which reads the entire track in one pass. This alleviates the need for custom interleave values. They do seem to alter the algorithm that finds the next available free sector so that the interleave value can deviate from 10 under certain circumstances, but I don't know why they would bother.

Below is a HEX dump of a Speed DOS BAM sector. Note the location of the extra BAM info from $C0-D3.

```
        00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
        -----------------------------------------------
0070: 12 FF FF 03 12 FF FF 03 12 FF FF 03 11 FF FF 01
0080: 11 FF FF 01 11 FF FF 01 11 FF FF 01 11 FF FF 01
```

```
0090: A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0
00A0: A0 A0 30 30 A0 32 41 A0 A0 A0 A0 00 00 00 00 00
00B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00C0: 11 FF FF 01 11 FF FF 01 11 FF FF 01 11 FF FF 01
00D0: 11 FF FF 01 00 00 00 00 00 00 00 00 00 00 00 00
```

Below is a HEX dump of a Dolphin DOS BAM sector. Note the location of the extra BAM info from $AC-BF.

```
      00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
      -----------------------------------------------
0070: 12 FF FF 03 12 FF FF 03 12 FF FF 03 11 FF FF 01
0080: 11 FF FF 01 11 FF FF 01 11 FF FF 01 11 FF FF 01
0090: A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0
00A0: A0 A0 30 30 A0 32 41 A0 A0 A0 A0 00 11 FF FF 01
00B0: 11 FF FF 01 11 FF FF 01 11 FF FF 01 11 FF FF 01
00C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Below is a HEX dump of a PrologicDOS BAM sector. Note that the disk name and ID are now located at $A4 instead of starting at $90.

```
      00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
      -----------------------------------------------
0070: 12 FF FF 03 12 FF FF 03 12 FF FF 03 11 FF FF 01
0080: 11 FF FF 01 11 FF FF 01 11 FF FF 01 11 FF FF 01
0090: 11 FF FF 01 11 FF FF 01 11 FF FF 01 11 FF FF 01
00A0: 11 FF FF 01 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0
00B0: A0 A0 A0 A0 A0 A0 30 30 A0 32 50 A0 A0 A0 A0 00
00C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

*** Error codes

Here is the meaning of the error bytes added onto the end of any extended D64. The CODE is the same as that generated by the 1541 drive controller... it reports these numbers, not the error code we usually see when an error occurs.

Some of what comes below is taken from Immers/Neufeld book "Inside Commodore DOS". Note the descriptions are not completely accurate as to what the drive DOS is actually doing to seek/read/decode/write sectors, but serve as simple examples only. The "type" field is where the error usually occurs, whether it's searching for any SYNC mark, any header ID, any valid header, or reading a sector.

These first errors are "seek" errors, where the disk controller is simply reading headers and looking at descriptor bytes, checksums, format ID's and reporting what errors it sees. These errors do *not* necessarily apply to the exact sector being looked for. This fact makes duplication of these errors very unreliable.

Code : $03 Error : 21 Type : Seek Message : No SYNC sequence found.

Each sector data block and header block are preceeded by SYNC marks. If *no* sync sequence is found within 20 milliseconds (only ~1/10 of a disk rotation!) then this error is generated. This error used to mean the entire track is bad, but it does not have to be the

case. Only a small area of the track needs to be without a SYNC mark and this error will be generated.

Converting this error to a D64 is very problematic because it depends on where the physical head is on the disk when a read attempt is made. If it is on valid header/sectors then it won't occur. If it happens over an area without SYNC marks, it will happen.

Code : $02 Error : 20 Type : Seek Message : Header descriptor byte not found (HEX $08, GCR $52)

Each sector is preceeded by an 8-byte GCR header block, which starts with the value $52 (GCR). If this value is not found after 90 attempts, this error is generated.

Basically, what a track has is SYNC marks, and possibly valid data blocks, but no valid header descriptors.

Code : $09 Error : 27 Type : Seek Message : Checksum error in header block

The header block contains a checksum value, calculated by XOR'ing the TRACK, SECTOR, ID1 and ID2 values. If this checksum is wrong, this error is generated.

Code : $0B Error : 29 Type : Seek Message : Disk sector ID mismatch

The ID's from the header block of the currently read sector are compared against the ones from the low-level header of 18/0. If there is a mismatch, this error is generated.

Code : $02 Error : 20 Type : Seek Message : Header block not found

This error can be reported again when searching for the correct header block. An image of the header is built and searched for, but not found after 90 read attempts. Note the difference from the first occurance. The first one only searches for a valid ID, not the whole header.

Note that error 20 occurs twice during this phase. The first time is when a header ID is being searched for, the second is when the proper header pattern for the sector being searched for is not found.

From this point on, all the errors apply to the specific sector you are looking for. If a read passed all the previous checks, then we are at the sector being searched for.

Note that the entire sector is read before these errors are detected. Therefore the data, checksum and off bytes are available.

Code : $04 Error : 22 Type : Read Message : Data descriptor byte not found (HEX $07, GCR $55)

Each sector data block is preceeded by the value $07, the "data block" descriptor. If this value is not there, this error is generated. Each encoded sector has actually 260 bytes. First is the descriptor byte, then follows the 256 bytes of data, a checksum, and two "off" bytes.

Code : $05 Error : 23 Type : Read Message : Checksum error in data block

The checksum of the data read of the disk is calculated, and compared against the one stored at the end of the sector. If there's a discrepancy, this error is generated.

Code : $0F Error : 74 Type : Read Message : Drive Not Ready (no disk in drive or no device 1)

These errors only apply when writing to a disk. I don't see the usefulness of having these as they cannot be present when only *reading* a disk.

Code : $06 Error : 24 Type : Write Message : Write verify (on format)

Code : $07 Error : 25 Type : Write Message : Write verify error

Once the GCR-encoded sector is written out, the drive waits for the sector to come around again and verifies the whole 325-byte GCR block. Any errors encountered will generate this error.

Code : $08 Error : 26 Type : Write Message : Write protect on

Self explanatory. Remove the write-protect tab, and try again.

Code : $0A Error : 28 Type : Write Message : Write error

In actual fact, this error never occurs, but it is included for completeness.

This is not an error at all, but it gets reported when the read of a sector is ok.

Code : $01 Error : 00 Type : N/A Message : No error.

Self explanatory. No errors were detected in the reading and decoding of the sector.

The advantage with using the 35 track D64 format, regardless of error bytes, is that it can be converted directly back to a 1541 disk by either using the proper cable and software on the PC, or send it down to the C64 and writing it back to a 1541. It is the best documented format since it is also native to the C64, with many books explaining the disk layout and the internals of the 1541.

15.4 The X64 disk image format

(This section was contributed by Peter Schepers and slightly edited by Marco van den Heuvel.)

This file type, created by Teemu Rantanen, is used on the X64 emulator (a UNIX-based emulator) which has been superceeded by VICE. Both VICE and X64 support the X64 file standard, with VICE also supporting the regular D64 and T64 files.

X64 is not a specific type of file, but rather encompasses *all* known C64 disk types (hard disk, floppies, etc). An X64 is created by prepending a 64-byte header to an existing image (1541, 1571, etc) and setting specific bytes which describe what type of image follows. This header has undergone some revision, and this description is based on the 1.02 version, which was the last known at the time of writing.

The most common X64 file you will see is the D64 variety, typically 174912 bytes long (174848 for the D64 and 64 bytes for the header, assuming no error bytes are appended). The header layout (as used in 64COPY) is as follows:

```
          00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
          -----------------------------------------------
0000: 43 15 41 64 01 02 01 23 00 00 00 00 00 00 00 00
0010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040: XX XX XX <- standard C64 image starts here....
```

Bytes	Description
$00-$03	This is the "Magic header" ($43 $15 $41 $64)
$04	Header version major ($01)
$05	Header version minor ($01, now its up to $02)

$06	Device type represented
$07	Maximum tracks in image (only in version 1.02 or greater) 1540/41/70: 35 1571: 35 1581: 80 (Logical single-sided disk)
$08	Number of disk sides in image. This value must be $00 for all 1541 and 1581 formats. $00=No second side $01=Second side
$09	Error data flag.
$0A-$1F	Unused, set to $00
$20-$3E	Disk image description (in ASCII or ISO Latin/1)
$3F	Always set to $00
$40-	Standard C64 file begins here.

The device types are:

Value	**Drive type**
$00	1540 See note below...
$01	1541 (Default)
$02	1542
$03	1551
$04	1570
$05	1571
$06	1572
$08	1581
$10	2031 or 4031
$11	2040 or 3040
$12	2041
$18	4040
$20	8050
$21	8060
$22	8061
$30	SFD-1001
$31	8250
$32	8280

The first four bytes used for the device type at position $06 ($00 to $03) are functionally the same, and are compatible with older version of X64 files. Some old X64 files might have $00 for the device type (instead of $01), but it makes no real difference.

As most instances of X64 files will be strictly 1541 images, bytes $08-$3F are set to zero, and some versions of the X64 emulator don't use bytes $08-$3F.

15.5 The D71 disk image format

(This section was contributed by Peter Schepers and slightly edited by Marco van den Heuvel.)

Similar to the D64 (1541), the 1571 drive can operate in either single-sided (1541 compatible) mode or double-sided (1571) mode. In this section I will be dealing with the double-sided mode only. For the breakdown of the single-sided mode, see the D64 section.

The D71 has 70 tracks, double that of the 1541, with a DOS file size of 349696 bytes. If the error byte block (1366 bytes) is attached, this makes the file size 351062 bytes. The track range and offsets into the D71 files are as follows:

Track	Sec/trk	# Sectors
1-17 (side 0)	21	357
18-24 (side 0)	19	133
25-30 (side 0)	18	108
31-35 (side 0)	17	85
36-52 (side 1)	21	357
53-59 (side 1)	19	133
60-65 (side 1)	18	108
66-70 (side 1)	17	85

Track	#Sect	#SectorsIn	D71 Offset
1	21	0	$00000
2	21	21	$01500
3	21	42	$02A00
4	21	63	$03F00
5	21	84	$05400
6	21	105	$06900
7	21	126	$07E00
8	21	147	$09300
9	21	168	$0A800
10	21	189	$0BD00
11	21	210	$0D200
12	21	231	$0E700
13	21	252	$0FC00
14	21	273	$11100
15	21	294	$12600
16	21	315	$13B00
17	21	336	$15000
18	19	357	$16500
19	19	376	$17800
20	19	395	$18B00
21	19	414	$19E00
22	19	433	$1B100
23	19	452	$1C400
24	19	471	$1D700
25	18	490	$1EA00
26	18	508	$1FC00
27	18	526	$20E00
28	18	544	$22000
29	18	562	$23200
30	18	580	$24400
31	17	598	$25600
32	17	615	$26700
33	17	632	$27800
34	17	649	$28900
35	17	666	$29A00
36	21	683	$2AB00

37	21	704	$2C000
38	21	725	$2D500
39	21	746	$2EA00
40	21	767	$2FF00
41	21	788	$31400
42	21	809	$32900
43	21	830	$33E00
44	21	851	$35300
45	21	872	$36800
46	21	893	$37D00
47	21	914	$39200
48	21	935	$3A700
49	21	956	$3BC00
50	21	977	$3D100
51	21	998	$3E600
52	21	1019	$3FB00
53	19	1040	$41000
54	19	1059	$42300
55	19	1078	$43600
56	19	1097	$44900
57	19	1116	$45C00
58	19	1135	$46F00
59	19	1154	$48200
60	18	1173	$49500
61	18	1191	$4A700
62	18	1209	$4B900
63	18	1227	$4CB00
64	18	1245	$4DD00
65	18	1263	$4EF00
66	17	1281	$50100
67	17	1298	$51200
68	17	1315	$52300
69	17	1332	$53400
70	17	1349	$54500

The directory structure is the same as a D64/1541. All the same filetypes apply, the directory still only holds 144 files per disk and should only exist on track 18.

The first two bytes of the sector ($12/$04 or 18/4) indicate the location of the next track/sector of the directory. If the track value is set to $00, then it is the last sector of the directory. It is possible, however unlikely, that the directory may *not* be competely on track 18 (some disks do exist like this). Just follow the chain anyhow.

When the directory is done, the track value will be $00. The sector link should contain a value of $FF, meaning the whole sector is allocated, but the actual value doesn't matter. The drive will return all the available entries anyways. This is a breakdown of a standard directory sector and entry:

```
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
-----------------------------------------------
```

```
00: 12 04 82 11 00 4A 45 54 20 53 45 54 20 57 49 4C
10: 4C 59 A0 A0 A0 00 00 00 00 00 00 00 00 00 2B 00
20: 00 00 82 0F 01 4A 53 57 20 31 A0 A0 A0 A0 A0 A0
30: A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 00 BF 00
40: 00 00 82 06 03 53 4F 4E 20 4F 46 20 42 4C 41 47
50: 47 45 52 A0 A0 00 00 00 00 00 00 00 00 00 AE 00
60: 00 00 82 15 0D 50 4F 54 54 59 20 50 49 47 45 4F
70: 4E A0 A0 A0 A0 00 00 00 00 00 00 00 00 00 A2 00
80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Bytes	Description
$00-$1F	First directory entry
$20-$3F	Second dir entry
$40-$5F	Third dir entry
$60-$7F	Fourth dir entry
$80-$9F	Fifth dir entry
$A0-$BF	Sixth dir entry
$C0-$DF	Seventh dir entry
$E0-$FF	Eighth dir entry

This is a breakdown of a standard directory entry:

Bytes	Description
$00-$01	Track/Sector location of next directory sector ($00/$FF if its the last sector)
$02	File type
$03-$04	Track/sector location of first sector of file
$05-$14	16 character filename (in PETASCII, padded with $A0)
$15-$16	Track/Sector location of first side-sector block (REL file only)
$17	REL file record length (REL file only, max. value 254)
$18-$1D	Unused (except with GEOS disks)
$1E-$1F	File size in sectors, low/high byte order ($1E+$1F*256). The approx. filesize in bytes is <= #sectors * 254

The file type field is used as follows:

Bits	Description
0-3	The actual file type
4	Unused
5	Used only during SAVE- replacement
6	Locked flag (Set produces ">" locked files)
7	Closed flag (Not set produces "*", or "splat" files)

The actual file type can be one of the following:

Binary	Decimal	File type
0000	0	DEL
0001	1	SEQ
0010	2	PRG
0011	3	USR
0100	4	REL

Values 5-15 are illegal, but if used will produce very strange results. The 1571 is inconsistent in how it treats these bits. Some routines use all 4 bits, others ignore bit 3, resulting in values from 0-7.

When the 1571 is in is native ("1571") mode, files are stored with a sector interleave of 6, rather than 10 which the 1541 (and the 1571 in "1541" mode) uses. The directory still uses an interleave of 3.

*** Non-Standard & Long Directories

Most Commodore floppy disk drives use a single dedicated directory track where all filenames are stored. This limits the number of files stored on a disk based on the number of sectors on the directory track. There are some disk images that contain more files than would normally be allowed. This requires extending the directory off the default directory track by changing the last directory sector pointer to a new track, allocating the new sectors in the BAM, and manually placing (or moving existing) file entries there. The directory of an extended disk can be read and the files that reside there can be loaded without problems on a real drive. However, this is still a very dangerous practice as writing to the extended portion of the directory will cause directory corruption in the non- extended part. Many of the floppy drives core ROM routines ignore the track value that the directory is on and assume the default directory track for operations.

To explain: assume that the directory has been extended from track 18 to track 19/6 and that the directory is full except for a few slots on 19/6. When saving a new file, the drive DOS will find an empty file slot at 19/6 offset $40 and correctly write the filename and a few other things into this slot. When the file is done being saved the final file information will be written to 18/6 offset $40 instead of 19/6 causing some directory corruption to the entry at 18/6. Also, the BAM entries for the sectors occupied by the new file will not be saved and the new file will be left as a SPLAT (*) file.

Attempts to validate the disk will result in those files residing off the directory track to not be allocated in the BAM, and could also send the drive into an endless loop. The default directory track is assumed for all sector reads when validating so if the directory goes to 19/6, then the validate code will read 18/6 instead. If 18/6 is part of the normal directory chain then the validate routine will loop endlessly.

*** Bam layout The BAM is somewhat different as it now has to take 35 new tracks into account. In order to do this, most of the extra BAM information is stored on track 53/0, and the remaining sectors on track 53 are marked in the BAM as allocated. This does mean that except for one allocated sector on track 53, the rest of the track is unused and wasted. (Track 53 is the equivalent to track 18, but on the flip side of the disk). Here is a dump of the first BAM sector...

```
      00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
      -----------------------------------------------
00: 12 01 41 80 12 FF F9 17 15 FF FF 1F 15 FF FF 1F
```

```
10: 15 FF FF 1F 15 FF FF 1F 15 FF FF 1F 15 FF FF 1F
20: 15 FF FF 1F 15 FF FF 1F 15 FF FF 1F 15 FF FF 1F
30: 15 FF FF 1F 15 FF FF 1F 15 FF FF 1F 15 FF FF 1F
40: 15 FF FF 1F 15 FF FF 1F 11 FC FF 07 13 FF FF 07
50: 13 FF FF 07 13 FF FF 07 13 FF FF 07 13 FF FF 07
60: 13 FF FF 07 12 FF FF 03 12 FF FF 03 12 FF FF 03
70: 12 FF FF 03 12 FF FF 03 12 FF FF 03 11 FF FF 01
80: 11 FF FF 01 11 FF FF 01 11 FF FF 01 11 FF FF 01
90: A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0
A0: A0 A0 30 30 A0 32 41 A0 A0 A0 A0 00 00 00 00 00
B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 15 15 15
E0: 15 15 15 15 15 15 15 15 15 15 15 15 15 15 00 13
F0: 13 13 13 13 13 12 12 12 12 12 12 11 11 11 11 11
```

Bytes	**Description**
$00-$01	Track/Sector location of the first directory sector (should be set to 18/1 but it doesn't matter, and don't trust what is there, always go to 18/1 for first directory entry)
$02	Disk DOS version type (see note below) $41 ('A') = 1541
$03	Double-sided flag $00 - Single sided disk $80 - Double sided disk
$04-8F	BAM entries for each track, in groups of four bytes per track, starting on track 1.
$90-$9F	Disk Name (padded with $A0)
$A0-$A1	Filled with $A0
$A2-$A3	Disk ID
$A4	Usually $A0
$A5-$A6	DOS type, usually "2A"
$A7-$AA	Filled with $A0
$AB-$DC	Not used ($00's)
$DD-$FF	Free sector count for tracks 36-70 (1 byte/track).

The "free sector" entries for tracks 36-70 are likely included here in the first BAM sector due to some memory restrictions in the 1571 drive. There is only enough memory available for one BAM sector, but in order to generate the "blocks free" value at the end of a directory listing, the drive needs to know the extra track "free sector" values. It does make working with the BAM a little more difficult, though.

These are the values that would normally be with the 4-byte BAM entry, but the rest of the entry is contained on 53/0.

Note: If the DOS version byte is set to anything other than $41 or $00, then we have what is called "soft write protection". Any attempt to write to the disk will return the "DOS Version" error code 73. The 1571 is simply telling you that it thinks the disk format version is incorrect.

The BAM entries require some explanation. Take the first entry at bytes $04-$07 ($12 $FF $F9 $17). The first byte ($12) is the number of free sectors on that track. Since we are looking at the track 1 entry, this means it has 18 (decimal) free sectors.

The next three bytes represent the bitmap of which sectors are used/free. Since it is 3 bytes (8 bits/byte) we have 24 bits of storage. Remember that at most, each track only has 21 sectors, so there are a few unused bits. These entries must be viewed in binary to make any sense. We will use the first entry (track 1) at bytes 04-07:

```
FF=11111111, F9=11111001, 17=00010111
```

In order to make any sense from the binary notation, flip the bits around.

```
           111111 11112222
01234567 89012345 67890123
-------------------------
11111111 10011111 11101000
^                     ^

sector 0           sector 20
```

Since we are on the first track, we have 21 sectors, and only use up to the bit 20 position. If a bit is on (1), the sector is free. Therefore, track 1 has sectors 9,10 and 19 used, all the rest are free.

In order to complete the BAM, we must check 53/0.

```
     00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
     -----------------------------------------------
00: FF FF 1F FF FF 1F FF FF 1F FF FF 1F FF FF 1F FF
10: FF 1F FF FF 1F FF FF 1F FF FF 1F FF FF 1F FF FF
20: 1F FF FF 1F FF FF 1F FF FF 1F FF FF 1F FF FF 1F
30: FF FF 1F 00 00 00 FF FF 07 FF FF 07 FF FF 07 FF
40: FF 07 FF FF 07 FF FF 07 FF FF 03 FF FF 03 FF FF
50: 03 FF FF 03 FF FF 03 FF FF 03 FF FF 01 FF FF 01
60: FF FF 01 FF FF 01 FF FF 01 00 00 00 00 00 00 00
70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Each track from 36-70 has 3 byte entries, starting at address $00.

```
Byte: $00-$02: $FF $FF $1F - BAM map for track 36
      $03-$05: $FF $FF $1F - BAM map for track 37
      ...
      $33-$35: $00 $00 $00 - BAM map for track 53
      ...
      $66-$68: $FF $FF $01 - BAM map for track 70
      $69-$FF:             - Not used
```

You can break down the entries for tracks 36-70 the same way as track 1, just combine the free sector bytes from 18/0 and the BAM usage from 53 to get the full 4-byte entry.

Just like a D64, you can attach error bytes to the file, for sector error information. This block is 1366 bytes long, 1 byte for each of the 1366 sectors in the image. With the error bytes, the file size is 351062 bytes.

15.6 The D81 disk image format

(This section was contributed by Peter Schepers and slightly edited by Marco van den Heuvel.)

Like D64 and D71, this is a byte for byte copy of a physical 1581 disk. It consists of 80 tracks, 40 sectors each (0 to 39) for a size of 819200 bytes, or 3200 sectors. If the error byte block is attached, this makes the file size 822400 bytes.

There are three sectors on the directory track used for disk internals (header and BAM), leaving 37 sectors for filename entries, thus allowing for 296 files (37 * 8) to be stored at the root level of the disk.

The actual physical layout on the disk is quite different from what the user sees, but this is unimportant to the scope of this section. One important difference from the D64 and D71 is all the sector interleaves are now 1 for both files and directory storage (rather than 3 for directory and 10 for file on a D64/D71). This is due to the built-in buffering in the 1581. When reading a sector, the whole track will be buffered in memory, and any sectors being modified will be done in memory. Once it has to be written, the whole track will be written out in one step.

The track range and offsets into the D81 files are as follows:

Track	**#Sect**	**#SectorsIn**	**D81 Offset**
1	40	0	$00000
2	40	40	$02800
3	40	80	$05000
4	40	120	$07800
5	40	160	$0A000
6	40	200	$0C800
7	40	240	$0F000
8	40	280	$11800
9	40	320	$14000
10	40	360	$16800
11	40	400	$19000
12	40	440	$1B800
13	40	480	$1E000
14	40	520	$20800
15	40	560	$23000
16	40	600	$25800
17	40	640	$28000
18	40	680	$2A800
19	40	720	$2D000
20	40	760	$2F800
21	40	800	$32000

22	40	840	$34800
23	40	880	$37000
24	40	920	$39800
25	40	960	$3C000
26	40	1000	$3E800
27	40	1040	$41000
28	40	1080	$43800
29	40	1120	$46000
30	40	1160	$48800
31	40	1200	$4B000
32	40	1240	$4D800
33	40	1280	$50000
34	40	1320	$52800
35	40	1360	$55000
36	40	1400	$57800
37	40	1440	$5A000
38	40	1480	$5C800
39	40	1520	$5F000
40	40	1560	$61800
41	40	1600	$64000
42	40	1640	$66800
43	40	1680	$69000
44	40	1720	$6B800
45	40	1760	$6E000
46	40	1800	$70800
47	40	1840	$73000
48	40	1880	$75800
49	40	1920	$78000
50	40	1960	$7A800
51	40	2000	$7D000
52	40	2040	$7F800
53	40	2080	$82000
54	40	2120	$84800
55	40	2160	$87000
56	40	2200	$89800
57	40	2240	$8C000
58	40	2280	$8E800
59	40	2320	$91000
60	40	2360	$93800
61	40	2400	$96000
62	40	2440	$98800
63	40	2480	$9B000
64	40	2520	$9D800
65	40	2560	$A0000
66	40	2600	$A2800
67	40	2640	$A5000
68	40	2680	$A7800

69	40	2720	$AA000
70	40	2760	$AC800
71	40	2800	$AF000
72	40	2840	$B1800
73	40	2880	$B4000
74	40	2920	$B6800
75	40	2960	$B9000
76	40	3000	$BB800
77	40	3040	$BE000
78	40	3080	$C0800
79	40	3120	$C3000
80	40	3160	$C5800

The header sector is stored at 40/0, and contains the disk name, ID and DOS version bytes, but the BAM is no longer contained here (like the D64).

```
      00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
      -----------------------------------------------
00: 28 03 44 00 31 35 38 31 20 55 54 49 4C 49 54 59
10: 20 56 30 31 A0 A0 47 42 A0 33 44 A0 A0 00 00 00
20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Bytes	**Description**
$00-$01	Track/Sector location of the first directory sector (should be set to 40/3 but it doesn't matter, and don't trust what is there, always go to 40/3 for first directory entry)
$02	Disk DOS version type (see note below) $44 ('D')=1581
$03	$00
$04-$13	16 character Disk Name (padded with $A0)
$14-$15	$A0
$16-$17	Disk ID
$18	$A0
$19	DOS Version ("3")
$1A	Disk version ("D")
$1B-$1C	$A0
$1D-$FF	Unused (usually $00)

The following might be set if the disk is a GEOS format (this info is based on the D64 layout, and might not prove to be true)

Bytes	Description
$AB-$AC	Border sector (GEOS only, else set to $00)
$AD-$BC	GEOS ID string ("geos FORMAT V1.x" GEOS only, else $00)
$BD-$FF	Unused (usually $00)

Note: If the DOS version byte is changed to anything other than a $44 (or $00), then we have what is called "soft write protection". Any attempt to write to the disk will return the "DOS Version" error code 73. The drive is simply telling you that it thinks the disk format version is incompatible.

The directory track should be contained totally on track 40. Sectors 3-39 contain the entries and sector 1 and 2 contain the BAM (Block Availability Map). Sector 0 holds the disk name and ID. The first directory sector is always 40/3, even though the t/s pointer at 40/0 (first two bytes) might point somewhere else. It goes linearly up the sector count, 3-4-5-6-etc. Each sector holds up to eight entries.

The first two bytes of the sector ($28/$04) indicate the location of the next track/sector of the directory (40/4). If the track is set to $00, then it is the last sector of the directory. It is possible, however unlikely, that the directory may *not* be competely on track 40. Just follow the chain anyhow.

When the directory is done (track=$00), the sector should contain an $FF, meaning the whole sector is allocated. Theactual value doesn't matter as all the entries will be returned anyways. Each directory sector has the following layout:

```
      00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
      -----------------------------------------------
00: 28 04 81 2B 00 53 43 52 45 45 4E 20 20 33 A0 A0
10: A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 00 02 00
20: 00 00 81 2B 01 53 43 52 45 45 4E 20 20 34 A0 A0
30: A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 00 03 00
40: 00 00 81 2B 02 53 43 52 45 45 4E 20 20 35 A0 A0
50: A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 00 07 00
60: 00 00 81 2B 08 53 43 52 45 45 4E 20 20 36 A0 A0
70: A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 00 08 00
80: 00 00 81 2B 14 53 43 52 45 45 4E 20 20 37 A0 A0
90: A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 00 07 00
A0: 00 00 81 24 00 53 43 52 45 45 4E 20 20 38 A0 A0
B0: A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 00 0B 00
C0: 00 00 82 24 04 46 49 4C 45 34 32 39 33 36 39 30
D0: A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 00 07 00
E0: 00 00 82 24 06 46 49 4C 45 32 35 37 38 38 31 35
F0: A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 00 05 00
```

Bytes	Description
$00-$1F	First directory entry
$20-$3F	Second dir entry
$40-$5F	Third dir entry
$60-$7F	Fourth dir entry

$80-$9F	Fifth dir entry
$A0-$BF	Sixth dir entry
$C0-$DF	Seventh dir entry
$E0-$FF	Eighth dir entry

This is a breakdown of a standard directory entry:

Bytes	**Description**
$00-$01	Track/Sector location of next directory sector
$02	File type
$03-$04	Track/sector location of first sector of file or partition
$05-$14	16 character filename (in PETASCII, padded with $A0)
$15-$16	Track/Sector location of first SUPER SIDE SECTOR block (REL file only)
$17	REL file record length (REL file only)
$18-$1B	Unused (except with GEOS disks)
$1C-$1D	(Used during an SAVE or OPEN, holds the new t/s link)
$1E-$1F	File or partition size in sectors, low/high byte order ($1E+$1F*256). The approx. file size in bytes is <= #sectors * 254

The file type field is used as follows:

Bits	**Description**
0-3	The actual file type
4	Unused
5	Used only during SAVE- replacement
6	Locked flag (Set produces ">" locked files)
7	Closed flag (Not set produces "*", or "splat" files)

The actual file type can be one of the following:

Binary	**Decimal**	**File type**
0000	0	DEL
0001	1	SEQ
0010	2	PRG
0011	3	USR
0100	4	REL
0101	5	CBM (partition or sub-directory)

Values 6-15 are illegal, but if used will produce very strange results.

*** Non-Standard & Long Directories

Most Commdore floppy disk drives use a single dedicated directory track where all filenames are stored. This limits the number of files stored on a disk based on the number of sectors on the directory track. There are some disk images that contain more files than would normally be allowed. This requires extending the directory off the default directory track by changing the last directory sector pointer to a new track, allocating the new sectors in the BAM, and manually placing (or moving existing) file entries there. The directory of an extended disk can be read and the files that reside there can be loaded without problems on a real drive. However, this is still a very dangerous practice as writing to the extended portion of the directory will cause directory corruption in the non-extended part. Many of the floppy drives core ROM routines ignore the track value that the directory is on and assume the default directory track for operations.

*** BAM layout

The BAM is located on 40/1 (for side 0, tracks 1-40) and 40/2 (for side 1, tracks 41-80). Each entry takes up six bytes, one for the "free sector" count and five for the allocation bitmap.

```
     00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
     -----------------------------------------------
00: 28 02 44 BB 47 42 C0 00 00 00 00 00 00 00 00 00
10: 28 FF FF FF FF FF 28 FF FF FF FF FF 28 FF FF FF
20: FF FF 28 FF FF FF FF FF 28 FF FF FF FF FF 28 FF
30: FF FF FF FF 28 FF FF FF FF FF 28 FF FF FF FF FF
40: 28 FF FF FF FF FF 28 FF FF FF FF FF 28 FF FF FF
50: FF FF 28 FF FF FF FF FF 28 FF FF FF FF FF 28 FF
60: FF FF FF FF 28 FF FF FF FF FF 28 FF FF FF FF FF
70: 28 FF FF FF FF FF 28 FF FF FF FF FF 28 FF FF FF
80: FF FF 28 FF FF FF FF FF 28 FF FF FF FF FF 28 FF
90: FF FF FF FF 28 FF FF FF FF FF 28 FF FF FF FF FF
A0: 28 FF FF FF FF FF 28 FF FF FF FF FF 28 FF FF FF
B0: FF FF 28 FF FF FF FF FF 28 FF FF FF FF FF 28 FF
C0: FF FF FF FF 28 FF FF FF FF FF 28 FF FF FF FF FF
D0: 28 FF FF FF FF FF 28 FF FF FF FF FF 28 FF FF FF
E0: FF FF 28 FF FF FF FF FF 28 FF FF FF FF FF 28 FF
F0: FF FF FF FF 28 FF FF FF FF FF 24 F0 FF 2D FF FE
```

```
Bytes:
$00-$01: Track/sector of next bam sector (40/2)
    $02: Version # ('D')
    $03: One's complement of version# ($BB)
$04-$05: Disk ID bytes (same as 40/0 Disk ID)
    $06: I/O byte
         bit 7 set - Verify on
         bit 7 clear - Verify off
         bit 6 set - Check header CRC
         bit 6 clear - Don't check header CRC
    $07: Auto-boot-loader flag
$08-$0F: Reserved for future (set to $00)
$10-$15: BAM entry for track 1 (track 41, side 1)
$16-$1B: BAM entry for track 2 (track 42, side 1)
         ...
$46-$4B: BAM entry for track 10 (track 50, side 1)
         ...
$82-$87: BAM entry for track 20 (track 60, side 1)
         ...
$BE-$C3: BAM entry for track 30 (track 70, side 1)
         ...
$FA-$FF: BAM entry for track 40 (track 80, side 1)
```

The BAM entries require some explanation, so lets look at the track 40 entry at bytes $FA-FF ($24 $F0 $FF $2D $FF $FE). The first byte ($24, or 36 decimal) is the number

of free sectors on that track. The next five bytes represent the bitmap of which sectors are used/free. Since it is five bytes (8 bits/byte) we have 40 bits of storage. Since this format has 40 sectors/track, the whole five bytes are used.

```
F0: .. .. .. .. .. .. .. .. .. .. 24 F0 FF 2D FF FE
```

The last five bytes of any BAM entry must be viewed in binary to make any sense. We will once again use track 40 as our reference:

```
F0=11110000, FF=11111111, 2D=00101101, FF=11111111, FE=11111110
```

In order to make any sense from the binary notation, flip the bits around.

```
                  111111 11112222 22222233 33333333
Sector 01234567 89012345 67890123 45678901 23456789
       ------------------------ -------- --------
       00001111 11111111 10110100 11111111 01111111
```

Note that if a bit is on (1), the sector is free. Therefore, track 40 has sectors 0-3, 17, 20, 22, 23 and 32 used, all the rest are free.

The second BAM (for side 1) contains the entries for tracks 41-80.

```
     00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
     -----------------------------------------------
00: 00 FF 44 BB 47 42 C0 00 00 00 00 00 00 00 00 00
10: 28 FF FF FF FF FF 28 FF FF FF FF FF 28 FF FF FF
20: FF FF 28 FF FF FF FF FF 28 FF FF FF FF FF 28 FF
30: FF FF FF FF 28 FF FF FF FF FF 28 FF FF FF FF FF
40: 28 FF FF FF FF FF 28 FF FF FF FF FF 28 FF FF FF
50: FF FF 28 FF FF FF FF FF 28 FF FF FF FF FF 28 FF
60: FF FF FF FF 28 FF FF FF FF FF 28 FF FF FF FF FF
70: 28 FF FF FF FF FF 28 FF FF FF FF FF 28 FF FF FF
80: FF FF 28 FF FF FF FF FF 28 FF FF FF FF FF 28 FF
90: FF FF FF FF 28 FF FF FF FF FF 28 FF FF FF FF FF
A0: 28 FF FF FF FF FF 28 FF FF FF FF FF 28 FF FF FF
B0: FF FF 28 FF FF FF FF FF 28 FF FF FF FF FF 28 FF
C0: FF FF FF FF 28 FF FF FF FF FF 28 FF FF FF FF FF
D0: 28 FF FF FF FF FF 28 FF FF FF FF FF 28 FF FF FF
E0: FF FF 28 FF FF FF FF FF 28 FF FF FF FF FF 28 FF
F0: FF FF FF FF 28 FF FF FF FF FF 28 FF FF FF FF FF
```

It is laid out exactly as the side 0 BAM except for one difference. The track/sector reference for the next sector should be set to $00/$FF, indicating there is no next sector.

*** REL files The REL filetype requires some extra explaining. It was designed to make access to data *anywhere* on the disk very fast. Take a look at this directory entry...

```
00: 00 FF 84 27 00 41 44 44 49 54 49 4F 4E 41 4C 20
10: 49 4E 46 4F A0 27 02 FE 00 00 00 00 00 00 D2 0B
```

The third byte ($84) indicates this entry is a REL file and that the three normally empty entries at offset $15, $16 and $17 are now used as they are explained above. It's the track/sector chain that this entry points to, called the SUPER SIDE SECTOR, which is of interest here (in this case, 39/2). The SUPER SIDE SECTOR is very different from the D64 format. If you check the D64 entry for a REL file and do the calculations, you will

find that the maximum file size of the REL file is 720 data sectors. With the new SUPER SIDE SECTOR, you can now have 126 groups of these SIDE SECTORS chains, allowing for file sizes up to (theoretically) 90720 sectors, or about 22.15 Megabytes.

Here is a dump of the beginning of the SUPER SIDE SECTOR...

```
 00: 27 01 FE 27 01 15 09 03 0F 38 16 4A 1C 00 00 00
 10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
Bytes:
$00-$01: Track/sector of first side sector in group 0
    $02: Always $FE
$03-$04: Track/sector of first side sector in group 0 (again)
         ...
$FD-$FE: Track/sector of first side sector in group 125
    $FF: Unused (likely $00)
```

The side sector layout is the same as the D64/1571.

```
    00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
    -----------------------------------------------
00: 12 0A 00 FE 15 09 12 0A 0F 0B 0C 0C 09 0D 06 0E
10: 15 07 15 08 15 0A 15 0B 15 0C 15 0D 15 0E 15 0F
20: 15 10 15 11 15 12 15 13 15 14 15 15 15 16 15 17
30: 15 18 15 19 15 1A 15 1B 15 1C 15 1D 15 1E 15 1F
40: 15 20 15 21 15 22 15 23 15 24 15 25 15 26 15 27
50: 14 00 14 01 14 02 14 03 14 04 14 05 14 06 14 07
60: 14 08 14 09 14 0A 14 0B 14 0C 14 0D 14 0E 14 0F
70: 14 10 14 11 14 12 14 13 14 14 14 15 14 16 14 17
80: 14 18 14 19 14 1A 14 1B 14 1C 14 1D 14 1E 14 1F
90: 14 20 14 21 14 22 14 23 14 24 14 25 14 26 14 27
A0: 13 00 13 01 13 02 13 03 13 04 13 05 13 06 13 07
B0: 13 08 13 09 13 0A 13 0B 13 0C 13 0D 13 0E 13 0F
C0: 13 10 13 11 13 12 13 13 13 14 13 15 13 16 13 17
D0: 13 18 13 19 13 1A 13 1B 13 1C 13 1D 13 1E 13 1F
E0: 13 20 13 21 13 22 13 23 13 24 13 25 13 26 13 27
F0: 12 00 12 01 12 02 12 03 12 04 12 05 12 06 12 07
```

```
Bytes:
    $00: Track location of next side-sector ($00 if last sector)
    $01: Sector location of next side-sector
    $02: Side-sector block number (first sector is $00, the next is
         $01, then $02, etc)
    $03: REL file RECORD size (from directory entry)
$04-$0F: Track/sector locations of the six other side-sectors. Note
         the first entry is this very sector we have listed here.
         The next is the next t/s listed at the beginning of the
         sector. All of this information must be correct. If one of
         these chains is $00/$00, then we have no more side sectors.
         Also, all of these (up to six) side sectors must have the
```

```
          same values in this range.
  $10-$FF: T/S chains of *each* sector of the data portion. When we
          get a $00/$00, we are at the end of the file.
```

*** 1581 Partitions and Sub-directories

At the beginning of this section it was stated that the 1581 can hold 296 entries "at the root level". The 1581 also has the ability to partition areas of the disk. Under the right conditions these can become sub-directories, acting as a small diskette, complete with its own directory and BAM. When you are inside of a sub-directory, no other files except those in that directory are visible, or can be affected.

To the 1581, this file will show up as a "CBM" filetype in a directory. All this does is tell the disk that a file, starting at X/Y track/sector and Z sectors large exists. Doing a validate will not harm these files as they have a directory entry, and are fully allocated in the BAM.

There are two main uses for partitions. One is to simply allocate a section of the disk to be used for direct-access reads/writes, and lock it away from being overwritten after a VALIDATE. The second is as a sub-directory, basically a small "disk within a disk".

In order to use a partition as a sub-directory, it must adhere to the following four rules:

1. If must start on sector 0
2. It's size must be in multiples of 40 sectors
3. It must be a minimum of 120 sectors long (3 tracks)
4. If must not start on or cross track 40, which limits the biggest directory to 1600 sectors (tracks 1-39).

This is a dump of a sub-directory entry:

```
00: 00 FF 85 29 00 50 41 52 54 49 54 49 4F 4E 20 31
10: A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 00 40 06
```

It is a partition starting on track 41/0, extends for 1600 sectors, and has been formatted as a sub-directory. Note that when a partition is created, the area being allocated is not touched in any way. If you want it set up as a sub-directory, you must issue the FORMAT command to the 1581 to create the central directory and BAM. Also note that from the directory entry you can't tell whether it is a sub-directory or not, just that it fits the sub-directory parameters.

The BAM track for the sub-directory exists on the first track of the partition, and has the same layout as the disk BAM on track 40. The biggest difference is the "disk name" is what what given when the partition was formatted rather than what the actual disk name is. Also, except for the free sectors in the partition area, all other sectors in the BAM will be allocated.

If the partition size doesn't match the above rules for a sub-directory, it will simply exist as a "protected" area of the disk, and can't be used as a sub-directory. Either way, it still shows up as a "CBM" type in a directory listing. Below is a dump of a 10-sector partition starting on track 5/1, which does not qualify as a sub-directory...

```
00: 00 00 85 05 01 53 4D 41 4C 4C 50 41 52 54 20 32
10: A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 00 0A 00
```

The master BAM shows the entry for this partition on track 5...

```
00: 28 02 44 BB 43 44 C0 00 00 00 00 00 00 00 00 00
10: 23 C1 FF FF FF FF 28 FF FF FF FF FF 28 FF FF FF
20: FF FF 28 FF FF FF FF FF 1E 01 F8 FF FF FF 28 FF
                           ^^^^^^^^^^^^^^^^^
```

The breakdown of the BAM shows the allocation for this track, with sectors 1-10 allocated, as it should be.

```
10000000 00011111 11111111 11111111 11111111
^        ^        ^        ^        ^
0        10       20       30       39
```

Partitions and sub-directories share one very important trait. When created, the sub-directory entry simply has the starting track/sector and the size of the partition in sectors. Partitions are created linearly, meaning if one starts on 30/1 and is of size 15 sectors, then the sector range from 1 through 15 on track 30 will be allocated. If a partition size crosses a track boundary, the allocation will continue on the next track starting on sector 0, and going up.

The section allocated will *not* have a track/sector chain like a file would, but rather is dependant on the directory entry to keep it from being overwritten. You can store whatever you want to in the allocated area.

*** AUTO-BOOT LOADER

If byte $07 in the BAM is set, then when the drive is reset (and other circumstances) it will look for a USR file called "COPYRIGHT CBM 86". This file will then be loaded into the drive RAM and executed.

The format for this auto-loader file is fairly basic. It starts with a two-byte load address, a size byte, program data, and a checksum at the end.

```
Bytes:
       $00-$01: Load address, low/high format
           $02: Size of program (SZ) (smaller than 256 bytes)
$03-($03+SZ-1): Program data
        $03+SZ: Checksum byte
```

15.7 The D80 disk image format

(This section was contributed by Peter Schepers and slightly edited by Marco van den Heuvel.)

This is a sector-for-sector copy of an 8050 floppy disk. The file size for an 8050 image is 533248 bytes. It is comprised of 256-byte sectors arranged across 77 tracks, with a varying number of sectors per track for a total of 2083 sectors. Track counting starts at 1 (not 0) and sector counting starts at 0 (not 1), therefore a track with 29 sectors will go from 0 to 28.

The original media (a 5.25" disk) has the tracks laid out in circles, with track 1 on the very outside of the disk (closest to the sides) to track 77 being on the inside of the disk (closest to the inner hub ring). Commodore, in their infinite wisdom, varied the number of sectors per track and data densities across the disk to optimize available storage, resulting in the chart below. It shows the sectors/track for a D80. Since the outside diameter of a

circle is the largest (versus closer to the center), the outside tracks have the largest amount of storage.

Track Range	Sectors/track	# Sectors
1-39	29	1131
40-53	27	378
54-64	25	275
65-77	23	299

Track	#Sect	#SectorsIn	D8x Offset
1	29	0	$00000
2	29	29	$01D00
3	29	58	$03A00
4	29	87	$05700
5	29	116	$07400
6	29	145	$09100
7	29	174	$0AE00
8	29	203	$0CB00
9	29	232	$0E800
10	29	261	$10500
11	29	290	$12200
12	29	319	$13F00
13	29	348	$15C00
14	29	377	$17900
15	29	406	$19600
16	29	435	$1B300
17	29	464	$1D000
18	29	493	$1ED00
19	29	522	$20A00
20	29	551	$22700
21	29	580	$24400
22	29	609	$26100
23	29	638	$27E00
24	29	667	$29B00
25	29	696	$2B800
26	29	725	$2D500
27	29	754	$2F200
28	29	783	$30F00
29	29	812	$32C00
30	29	841	$34900
31	29	870	$36600
32	29	899	$38300
33	29	928	$3A000
34	29	957	$3BD00
35	29	986	$3DA00
36	29	1015	$3F700
37	29	1044	$41400
38	29	1073	$43100

39	29	1102	$44E00
40	27	1131	$46B00
41	27	1158	$48600
42	27	1185	$4A100
43	27	1212	$4BC00
44	27	1239	$4D700
45	27	1266	$4F200
46	27	1293	$50D00
47	27	1320	$52800
48	27	1347	$54300
49	27	1374	$55E00
50	27	1401	$57900
51	27	1428	$59400
52	27	1455	$5AF00
53	27	1482	$5CA00
54	25	1509	$5E500
55	25	1534	$5FE00
56	25	1559	$61700
57	25	1584	$63000
58	25	1609	$64900
59	25	1634	$66200
60	25	1659	$67B00
61	25	1684	$69400
62	25	1709	$6AD00
63	25	1734	$6C600
64	25	1759	$6DF00
65	23	1784	$6F800
66	23	1807	$70F00
67	23	1830	$72600
68	23	1853	$73D00
69	23	1876	$75400
70	23	1899	$76B00
71	23	1922	$78200
72	23	1945	$79900
73	23	1968	$7B000
74	23	1991	$7C700
75	23	2014	$7DE00
76	23	2037	$7F500
77	23	2060	$80C00

The BAM (Block Availability Map) is on track 38. The D80 is only 77 tracks and so the BAM is contained on 38/0 and 38/3. The BAM interleave is 3.

The directory is on track 39, with 39/0 contains the header (DOS type, disk name, disk ID's) and sectors 1-28 contain the directory entries. Both files and the directory use an interleave of 1. Since the directory is only 28 sectors large (29 less one for the header), and each sector can contain only 8 entries (32 bytes per entry), the maximum number of

directory entries is 28 * 8 = 224. The first directory sector is always 39/1. It then follows a chain structure using a sector interleave of 1 making the links go 39/1, 39/2, 39/3 etc.

When reading a disk, you start with 39/0 (disk label/ID) which points to 38/0 (BAM0), 38/3 (BAM1), and finally to 39/1 (first dir entry sector). When writing a file to a blank disk, it will start at 38/1 because 38/0 is already allocated.

Below is a dump of the header sector 39/0:

```
      00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
      -----------------------------------------------
00: 26 00 43 00 00 00 73 61 6D 70 6C 65 20 64 38 30
10: A0 A0 A0 A0 A0 A0 A0 A0 65 72 A0 32 43 A0 A0 A0
20: A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
...
F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Bytes	**Description**
$00-$01	T/S pointer to first BAM sector (38/0)
$02	$43 'C' is for DOS format version
$03	Reserved
$04-$05	Unused
$06-$16	Disk name, padded with 0xA0 ("sample d80")
$17	0xA0
$18-$19	Disk ID bytes "er"
$1A	0xA0
$1B-$1C	DOS version bytes "2C"
$1D-$20	0xA0
$21-$FF	Unused

Below is a dump of the first directory sector, 39/1

```
      00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
      -----------------------------------------------
00: 27 02 82 26 01 54 45 53 54 A0 A0 A0 A0 A0 A0 A0
10: A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 00 01 00
20: 00 00 82 26 02 54 45 53 54 32 A0 A0 A0 A0 A0 A0
30: A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 00 01 00
40: 00 00 82 26 04 54 45 53 54 33 A0 A0 A0 A0 A0 A0
50: A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 00 05 00
60: 00 00 82 26 0B 54 45 53 54 34 A0 A0 A0 A0 A0 A0
70: A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 00 09 00
80: 00 00 82 26 14 54 45 53 54 35 A0 A0 A0 A0 A0 A0
90: A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 00 0C 00
A0: 00 00 82 28 00 54 45 53 54 36 A0 A0 A0 A0 A0 A0
B0: A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 00 01 00
C0: 00 00 82 28 01 54 45 53 54 37 A0 A0 A0 A0 A0 A0
D0: A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 00 01 00
E0: 00 00 82 28 02 54 45 53 54 38 A0 A0 A0 A0 A0 A0
F0: A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 00 01 00
```

The first two bytes of the directory sector ($27/$02) indicate the location of the next track/sector of the directory (39/2). If the track is set to $00, then it is the last sector of the directory.

When the directory is done, the track value will be $00. The sector link should contain a value of $FF, meaning the whole sector is allocated, but the actual value doesn't matter. The drive will return all the available entries anyways. This is a breakdown of a standard directory sector:

Bytes	**Description**
$00-$1F	First directory entry
$20-$3F	Second dir entry
$40-$5F	Third dir entry
$60-$7F	Fourth dir entry
$80-$9F	Fifth dir entry
$A0-$BF	Sixth dir entry
$C0-$DF	Seventh dir entry
$E0-$FF	Eighth dir entry

This is a breakdown of a standard directory entry:

Bytes	**Description**
$00-$01	Track/Sector location of next directory sector ($00 $00 if not the first entry in the sector)
$02	File type
$03-$04	Track/sector location of first sector of file
$05-$14	16 character filename (in PETASCII, padded with $A0)
$15-$16	Track/Sector location of first side-sector block (REL file only)
$17	REL file record length (REL file only, max. value 254)
$18-$1D	Unused
$1E-$1F	File size in sectors, low/high byte order ($1E+$1F*256). The approx. filesize in bytes is <= #sectors * 254 0

The file type field is used as follows:

Bits	**Description**
0-3	The actual file type
4	Unused
5	Used only during SAVE- replacement
6	Locked flag (Set produces ">" locked files)
7	Closed flag (Not set produces "*", or "splat" files)

The actual file type can be one of the following:

Binary	**Decimal**	**File type**
0000	0	DEL
0001	1	SEQ
0010	2	PRG
0011	3	USR
0100	4	REL

Values 5-15 are illegal, but if used will produce very strange results.

*** Non-Standard & Long Directories

Most Commdore floppy disk drives use a single dedicated directory track where all filenames are stored. This limits the number of files stored on a disk based on the number of sectors on the directory track. There are some disk images that contain more files than would normally be allowed. This requires extending the directory off the default directory track by changing the last directory sector pointer to a new track, allocating the new sectors in the BAM, and manually placing (or moving existing) file entries there. The directory of an extended disk can be read and the files that reside there can be loaded without problems on a real drive. However, this is still a very dangerous practice as writing to the extended portion of the directory will cause directory corruption in the non-extended part. Many of the floppy drives core ROM routines ignore the track value that the directory is on and assume the default directory track for operations.

*** BAM layout

The BAM only occupies up to four sectors on track 38, so the rest of the track is empty and is available for file storage. Below is a dump of the first BAM block, 38/0. A D80 will only contain two BAM sectors, 38/0 and 38/3. Each entry takes 5 bytes, 1 for the free count on that track, and 4 for the BAM bits.

```
      00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
      -----------------------------------------------
00: 26 03 43 00 01 33 1D FF FF FF 1F 1D FF FF FF 1F
10: 1D FF FF FF 1F 1D FF FF FF 1F 1D FF FF FF 1F 1D
20: FF FF FF 1F 1D FF FF FF 1F 1D FF FF FF 1F 1D FF
30: FF FF 1F 1D FF FF FF 1F 1D FF FF FF 1F 1D FF FF
40: FF 1F 1D FF FF FF 1F 1D FF FF FF 1F 1D FF FF FF
50: 1F 1D FF FF FF 1F 1D FF FF FF 1F 1D FF FF FF 1F
60: 1D FF FF FF 1F 1D FF FF FF 1F 1D FF FF FF 1F 1D
70: FF FF FF 1F 1D FF FF FF 1F 1D FF FF FF 1F 1D FF
80: FF FF 1F 1D FF FF FF 1F 1D FF FF FF 1F 1D FF FF
90: FF 1F 1D FF FF FF 1F 1D FF FF FF 1F 1D FF FF FF
A0: 1F 1D FF FF FF 1F 1D FF FF FF 1F 1D FF FF FF 1F
B0: 1D FF FF FF 1F 1D FF FF FF 1F 1D FF FF FF 1F 1B
C0: F6 FF FF 1F 1B FC FF FF 1F 1B FF FF FF 07 1B FF
D0: FF FF 07 1B FF FF FF 07 1B FF FF FF 07 1B FF FF
E0: FF 07 1B FF FF FF 07 1B FF FF FF 07 1B FF FF FF
F0: 07 1B FF FF FF 07 1B FF FF FF 07 1B FF FF FF 07
```

Bytes	**Description**
$00-$01	T/S pointer to second BAM sector (38/3)
$02	DOS version byte (0x43='C')
$03	Reserved
$04	Lowest track covered by this BAM (0x01=1)
$05	Highest+1 track covered by this BAM (0x33=51)
$06-$0A	BAM for track 1. The first byte shows the "blocks free" for this track, the remaining 4 show the BAM for the track.
$0B-$0F	BAM for track 2
. . .	. . .
$FB-$FF	BAM for track 50

Being bit-based, the BAM entries need some explanation. The first track entry in the above BAM sector is at offset 06, "1D FF FF FF 1F". The first number is how many blocks are free on this track ($1D=29) and the remainder is the bit representation of the usage map for the track. These entries must be viewed in binary to make any sense. First convert the values to binary:

```
FF=11111111, FF=11111111, FF=11111111, 1F=00011111
```

In order to make any sense from the binary notation, flip the bits around.

```
           111111 11112222 222222
01234567 89012345 67890123 456789...
------------------------- ---------
11111111 11111111 11111111 11111000
^                             ^
sector 0                   sector 28
```

Since we are on the first track, we have 29 sectors, and only use up to the bit 28 position. If a bit is on (1), the sector is free. Therefore, track 1 is clean, all sectors are free. Any leftover bits that refer to sectors that don't exist, like bits 29-31 in the above example, are set to allocated.

Second BAM block 38/3.

```
      00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
      -----------------------------------------------
00: 27 01 43 00 33 4E 1B FF FF FF 07 1B FF FF FF 07
10: 1B FF FF FF 07 19 FF FF FF 01 19 FF FF FF 01 19
20: FF FF FF 01 19 FF FF FF 01 19 FF FF FF 01 19 FF
30: FF FF 01 19 FF FF FF 01 19 FF FF FF 01 19 FF FF
40: FF 01 19 FF FF FF 01 19 FF FF FF 01 17 FF FF 7F
50: 00 17 FF FF 7F 00 17 FF FF 7F 00 17 FF FF 7F 00
60: 17 FF FF 7F 00 17 FF FF 7F 00 17 FF FF 7F 00 17
70: FF FF 7F 00 17 FF FF 7F 00 17 FF FF 7F 00 17 FF
80: FF 7F 00 17 FF FF 7F 00 17 FF FF 7F 00 00 00 00
90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Bytes	**Description**
$00-$01	T/S pointer to second BAM sector (39/1)
$02	DOS version byte (0x43='C')
$03	Reserved
$04	Lowest track covered by this BAM (0x33=51)
$05	Highest+1 track covered by this BAM (0x43=78)
$06-$0A	BAM for track 51. The first byte shows the "blocks free" for this track, the remaining 4 show the BAM for the track.
$0B-$0F	BAM for track 52

. . .	. . .
$88-$8C	BAM for track 77
$8D-$FF	Not used

15.8 The D82 disk image format

(This section was contributed by Peter Schepers and slightly edited by Marco van den Heuvel.)

This is a sector-for-sector copy of an 8250 floppy disk. The file size for an 8250 image is 1066496 bytes. It is comprised of 256-byte sectors arranged across 154 tracks, with a varying number of sectors per track for a total of 4166 sectors. Track counting starts at 1 (not 0) and sector counting starts at 0 (not 1), therefore a track with 29 sectors will go from 0 to 28.

The original media (a 5.25" disk) has the tracks laid out in circles, with track 1 on the very outside of the disk (closest to the sides) to track 77 being on the inside of the disk (closest to the inner hub ring). Commodore, in their infinite wisdom, varied the number of sectors per track and data densities across the disk to optimize available storage, resulting in the chart below. It shows the sectors/track for a D82. Since the outside diameter of a circle is the largest (versus closer to the center), the outside tracks have the largest amount of storage.

Track Range	**Sectors/track**	**# Sectors**
1-39	29	1131
40-53	27	378
55-64	25	275
65-77	23	299
78-116	29	1131
117-130	27	378
131-141	25	275
142-154	23	299

Track	**#Sect**	**#SectorsIn**	**D82 Offset**
1	29	0	$000000
2	29	29	$001D00
3	29	58	$003A00
4	29	87	$005700
5	29	116	$007400
6	29	145	$009100
7	29	174	$00AE00
8	29	203	$00CB00
9	29	232	$00E800
10	29	261	$010500
11	29	290	$012200
12	29	319	$013F00
13	29	348	$015C00
14	29	377	$017900
15	29	406	$019600
16	29	435	$01B300

17	29	464	$01D000
18	29	493	$01ED00
19	29	522	$020A00
20	29	551	$022700
21	29	580	$024400
22	29	609	$026100
23	29	638	$027E00
24	29	667	$029B00
25	29	696	$02B800
26	29	725	$02D500
27	29	754	$02F200
28	29	783	$030F00
29	29	812	$032C00
30	29	841	$034900
31	29	870	$036600
32	29	899	$038300
33	29	928	$03A000
34	29	957	$03BD00
35	29	986	$03DA00
36	29	1015	$03F700
37	29	1044	$041400
38	29	1073	$043100
39	29	1102	$044E00
40	27	1131	$046B00
41	27	1158	$048600
42	27	1185	$04A100
43	27	1212	$04BC00
44	27	1239	$04D700
45	27	1266	$04F200
46	27	1293	$050D00
47	27	1320	$052800
48	27	1347	$054300
49	27	1374	$055E00
50	27	1401	$057900
51	27	1428	$059400
52	27	1455	$05AF00
53	27	1482	$05CA00
54	25	1509	$05E500
55	25	1534	$05FE00
56	25	1559	$061700
57	25	1584	$063000
58	25	1609	$064900
59	25	1634	$066200
60	25	1659	$067B00
61	25	1684	$069400
62	25	1709	$06AD00
63	25	1734	$06C600

64	25	1759	$06DF00
65	23	1784	$06F800
66	23	1807	$070F00
67	23	1830	$072600
68	23	1853	$073D00
69	23	1876	$075400
70	23	1899	$076B00
71	23	1922	$078200
72	23	1945	$079900
73	23	1968	$07B000
74	23	1991	$07C700
75	23	2014	$07DE00
76	23	2037	$07F500
77	23	2060	$080C00
78	29	2083	$082300
79	29	2112	$084000
80	29	2141	$085D00
81	29	2170	$087A00
82	29	2199	$089700
83	29	2228	$08B400
84	29	2257	$08D100
85	29	2286	$08EE00
86	29	2315	$090600
87	29	2344	$092800
88	29	2373	$094500
89	29	2402	$096200
90	29	2431	$097F00
91	29	2460	$099C00
92	29	2489	$09B900
93	29	2518	$09D600
94	29	2547	$09F300
95	29	2576	$0A1000
96	29	2605	$0A2D00
97	29	2634	$0A4A00
98	29	2663	$0A6700
99	29	2692	$0A8400
100	29	2721	$0AA100
101	29	2750	$0ABE00
102	29	2779	$0ADB00
103	29	2808	$0AF800
104	29	2837	$0B1500
105	29	2866	$0B3200
106	29	2895	$0B4F00
107	29	2924	$0B6C00
108	29	2953	$0B8900
109	29	2982	$0BA600
110	29	3011	$0BC300

111	29	3040	$0BE000
112	29	3069	$0BFD00
113	29	3098	$0C1A00
114	29	3137	$0C3700
115	29	3156	$0C5400
116	29	3185	$0C7100
117	27	3214	$0C8E00
118	27	3241	$0CA900
119	27	3268	$0CC400
120	27	3295	$0CDF00
121	27	3322	$0CFA00
122	27	3349	$0D1500
123	27	3376	$0D3000
124	27	3403	$0D4B00
125	27	3430	$0D6600
126	27	3457	$0D8100
127	27	3484	$0D9C00
128	27	3511	$0DB700
129	27	3538	$0DD200
130	27	3565	$0DED00
131	25	3592	$0E0800
132	25	3617	$0E2100
133	25	3642	$0E3A00
134	25	3667	$0E5300
135	25	3692	$0E6C00
136	25	3717	$0E8500
137	25	3742	$0E9E00
138	25	3767	$0EB700
139	25	3792	$0ED000
140	25	3817	$0EE900
141	25	3842	$0F0200
142	23	3867	$0F1B00
143	23	3890	$0F3200
144	23	3913	$0F4900
145	23	3936	$0F6000
146	23	3959	$0F7700
147	23	3982	$0F8E00
148	23	4005	$0FA500
149	23	4028	$0FBC00
150	23	4051	$0FD300
151	23	4074	$0FEA00
152	23	4097	$100100
153	23	4120	$101800
154	23	4143	$102F00

The BAM (Block Availability Map) is on track 38. The D82 is 154 tracks and so the BAM is contained on 38/0, 38/3, 38/6 and 38/9. The BAM interleave is 3.

The directory is on track 39, with 39/0 contains the header (DOS type, disk name, disk ID's) and sectors 1-28 contain the directory entries. Both files and the directory use an interleave of 1. Since the directory is only 28 sectors large (29 less one for the header), and each sector can contain only 8 entries (32 bytes per entry), the maximum number of directory entries is 28 * 8 = 224. The first directory sector is always 39/1. It then follows a chain structure using a sector interleave of 1 making the links go 39/1, 39/2, 39/3 etc.

When reading a disk, you start with 39/0 (disk label/ID) which points to 38/0 (BAM0), 38/3 (BAM1), 38/6 (BAM2), 38/9 (BAM3, and finally to 39/1 (first dir entry sector). When writing a file to a blank disk, it will start at 38/1 because 38/0 is already allocated.

Below is a dump of the header sector 39/0:

```
      00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
      -----------------------------------------------
00: 26 00 43 00 00 00 73 61 6D 70 6C 65 20 64 38 30
10: A0 A0 A0 A0 A0 A0 A0 A0 65 72 A0 32 43 A0 A0 A0
20: A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
...
F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Bytes	**Description**
$00-$01	T/S pointer to first BAM sector (38/0)
$02	$43 'C' is for DOS format version
$03	Reserved
$04-$05	Unused
$06-$16	Disk name, padded with 0xA0 ("sample d82")
$17	0xA0
$18-$19	Disk ID bytes "er"
$1A	0xA0
$1B-$1C	DOS version bytes "2C"
$1D-$20	0xA0
$21-$FF	Unused

Below is a dump of the first directory sector, 39/1

```
      00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
      -----------------------------------------------
00: 27 02 82 26 01 54 45 53 54 A0 A0 A0 A0 A0 A0 A0
10: A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 00 01 00
20: 00 00 82 26 02 54 45 53 54 32 A0 A0 A0 A0 A0 A0
30: A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 00 01 00
40: 00 00 82 26 04 54 45 53 54 33 A0 A0 A0 A0 A0 A0
50: A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 00 05 00
60: 00 00 82 26 0B 54 45 53 54 34 A0 A0 A0 A0 A0 A0
70: A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 00 09 00
80: 00 00 82 26 14 54 45 53 54 35 A0 A0 A0 A0 A0 A0
90: A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 00 0C 00
A0: 00 00 82 28 00 54 45 53 54 36 A0 A0 A0 A0 A0 A0
B0: A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 00 01 00
C0: 00 00 82 28 01 54 45 53 54 37 A0 A0 A0 A0 A0 A0
```

```
D0: A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 00 01 00
E0: 00 00 82 28 02 54 45 53 54 38 A0 A0 A0 A0 A0 A0
F0: A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 00 01 00
```

The first two bytes of the directory sector ($27/$02) indicate the location of the next track/sector of the directory (39/2). If the track is set to $00, then it is the last sector of the directory.

When the directory is done, the track value will be $00. The sector link should contain a value of $FF, meaning the whole sector is allocated, but the actual value doesn't matter. The drive will return all the available entries anyways. This is a breakdown of a standard directory sector:

Bytes	**Description**
$00-$1F	First directory entry
$20-$3F	Second dir entry
$40-$5F	Third dir entry
$60-$7F	Fourth dir entry
$80-$9F	Fifth dir entry
$A0-$BF	Sixth dir entry
$C0-$DF	Seventh dir entry
$E0-$FF	Eighth dir entry

This is a breakdown of a standard directory entry:

Bytes	**Description**
$00-$01	Track/Sector location of next directory sector ($00 $00 if not the first entry in the sector)
$02	File type
$03-$04	Track/sector location of first sector of file
$05-$14	16 character filename (in PETASCII, padded with $A0)
$15-$16	Track/Sector location of first side-sector block (REL file only)
$17	REL file record length (REL file only, max. value 254)
$18-$1D	Unused
$1E-$1F	File size in sectors, low/high byte order ($1E+$1F*256). The approx. filesize in bytes is <= #sectors * 254 0

The file type field is used as follows:

Bits	**Description**
0-3	The actual file type
4	Unused
5	Used only during SAVE- replacement
6	Locked flag (Set produces ">" locked files)
7	Closed flag (Not set produces "*", or "splat" files)

The actual file type can be one of the following:

Binary	**Decimal**	**File type**
0000	0	DEL
0001	1	SEQ
0010	2	PRG
0011	3	USR
0100	4	REL

Values 5-15 are illegal, but if used will produce very strange results.

*** Non-Standard & Long Directories

Most Commdore floppy disk drives use a single dedicated directory track where all filenames are stored. This limits the number of files stored on a disk based on the number of sectors on the directory track. There are some disk images that contain more files than would normally be allowed. This requires extending the directory off the default directory track by changing the last directory sector pointer to a new track, allocating the new sectors in the BAM, and manually placing (or moving existing) file entries there. The directory of an extended disk can be read and the files that reside there can be loaded without problems on a real drive. However, this is still a very dangerous practice as writing to the extended portion of the directory will cause directory corruption in the non-extended part. Many of the floppy drives core ROM routines ignore the track value that the directory is on and assume the default directory track for operations.

*** BAM layout

The BAM only occupies up to four sectors on track 38, so the rest of the track is empty and is available for file storage. Below is a dump of the first BAM block, 38/0. A D82 will contain four BAM sectors, 38/0, 38/3, 38/6 and 38/9. Each entry takes 5 bytes, 1 for the free count on that track, and 4 for the BAM bits.

```
      00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
      -----------------------------------------------
00: 26 03 43 00 01 33 1D FF FF FF 1F 1D FF FF FF 1F
10: 1D FF FF FF 1F 1D FF FF FF 1F 1D FF FF FF 1F 1D
20: FF FF FF 1F 1D FF FF FF 1F 1D FF FF FF 1F 1D FF
30: FF FF 1F 1D FF FF FF 1F 1D FF FF FF 1F 1D FF FF
40: FF 1F 1D FF FF FF 1F 1D FF FF FF 1F 1D FF FF FF
50: 1F 1D FF FF FF 1F 1D FF FF FF 1F 1D FF FF FF 1F
60: 1D FF FF FF 1F 1D FF FF FF 1F 1D FF FF FF 1F 1D
70: FF FF FF 1F 1D FF FF FF 1F 1D FF FF FF 1F 1D FF
80: FF FF 1F 1D FF FF FF 1F 1D FF FF FF 1F 1D FF FF
90: FF 1F 1D FF FF FF 1F 1D FF FF FF 1F 1D FF FF FF
A0: 1F 1D FF FF FF 1F 1D FF FF FF 1F 1D FF FF FF 1F
B0: 1D FF FF FF 1F 1D FF FF FF 1F 1D FF FF FF 1F 1B
C0: F6 FF FF 1F 1B FC FF FF 1F 1B FF FF FF 07 1B FF
D0: FF FF 07 1B FF FF FF 07 1B FF FF FF 07 1B FF FF
E0: FF 07 1B FF FF FF 07 1B FF FF FF 07 1B FF FF FF
F0: 07 1B FF FF FF 07 1B FF FF FF 07 1B FF FF FF 07
```

Bytes	**Description**
$00-$01	T/S pointer to second BAM sector (38/3)
$02	DOS version byte (0x43='C')
$03	Reserved
$04	Lowest track covered by this BAM (0x01=1)
$05	Highest+1 track covered by this BAM (0x33=51)
$06-$0A	BAM for track 1. The first byte shows the "blocks free" for this track, the remaining 4 show the BAM for the track.
$0B-$0F	BAM for track 2

...	...
$FB-$FF	BAM for track 50

Being bit-based, the BAM entries need some explanation. The first track entry in the above BAM sector is at offset 06, "1D FF FF FF 1F". The first number is how many blocks are free on this track ($1D=29) and the remainder is the bit representation of the usage map for the track. These entries must be viewed in binary to make any sense. First convert the values to binary:

```
FF=11111111, FF=11111111, FF=11111111, 1F=00011111
```

In order to make any sense from the binary notation, flip the bits around.

```
           111111 11112222 222222
01234567 89012345 67890123 456789...
------------------------- ---------
11111111 11111111 11111111 11111000
^                             ^
sector 0                  sector 28
```

Since we are on the first track, we have 29 sectors, and only use up to the bit 28 position. If a bit is on (1), the sector is free. Therefore, track 1 is clean, all sectors are free. Any leftover bits that refer to sectors that don't exist, like bits 29-31 in the above example, are set to allocated.

Second BAM block 38/3

```
      00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
      -----------------------------------------------
00: 26 06 43 00 33 65 1B FF FF FF 07 1B FF FF FF 07
10: 1B FF FF FF 07 19 FF FF FF 01 19 FF FF FF 01 19
20: FF FF FF 01 19 FF FF FF 01 19 FF FF FF 01 19 FF
30: FF FF 01 19 FF FF FF 01 19 FF FF FF 01 19 FF FF
40: FF 01 19 FF FF FF 01 19 FF FF FF 01 17 FF FF 7F
50: 00 17 FF FF 7F 00 17 FF FF 7F 00 17 FF FF 7F 00
60: 17 FF FF 7F 00 17 FF FF 7F 00 17 FF FF 7F 00 17
70: FF FF 7F 00 17 FF FF 7F 00 17 FF FF 7F 00 17 FF
80: FF 7F 00 17 FF FF 7F 00 17 FF FF 7F 00 1D FF FF
90: FF 1F 1D FF FF FF 1F 1D FF FF FF 1F 1D FF FF FF
A0: 1F 1D FF FF FF 1F 1D FF FF FF 1F 1D FF FF FF 1F
B0: 1D FF FF FF 1F 1D FF FF FF 1F 1D FF FF FF 1F 1D
C0: FF FF FF 1F 1D FF FF FF 1F 1D FF FF FF 1F 1D FF
D0: FF FF 1F 1D FF FF FF 1F 1D FF FF FF 1F 1D FF FF
E0: FF 1F 1D FF FF FF 1F 1D FF FF FF 1F 1D FF FF FF
F0: 1F 1D FF FF FF 1F 1D FF FF FF 1F 1D FF FF FF 1F
```

Bytes	**Description**
$00-$01	T/S pointer to third BAM sector (38/6)
$02	DOS version byte (0x43='C')
$03	Reserved
$04	Lowest track covered by this BAM (0x33=51)
$05	Highest+1 track covered by this BAM (0x65=101)

$06-$0A	BAM for track 51. The first byte shows the "blocks free" for this track, the remaining 4 show the BAM for the track.
$0B-$0F	BAM for track 52
...	...
$FB-$FF	BAM for track 100

Third BAM block 38/6

```
      00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
      -----------------------------------------------
00: 26 09 43 00 65 97 1D FF FF FF 1F 1D FF FF FF 1F
10: 1D FF FF FF 1F 1D FF FF FF 1F 1D FF FF FF 1F 1D
20: FF FF FF 1F 1D FF FF FF 1F 1D FF FF FF 1F 1D FF
30: FF FF 1F 1D FF FF FF 1F 1D FF FF FF 1F 1D FF FF
40: FF 1F 1D FF FF FF 1F 1D FF FF FF 1F 1D FF FF FF
50: 1F 1D FF FF FF 1F 1B FF FF FF 07 1B FF FF FF 07
60: 1B FF FF FF 07 1B FF FF FF 07 1B FF FF FF 07 1B
70: FF FF FF 07 1B FF FF FF 07 1B FF FF FF 07 1B FF
80: FF FF 07 1B FF FF FF 07 1B FF FF FF 07 1B FF FF
90: FF 07 1B FF FF FF 07 1B FF FF FF 07 19 FF FF FF
A0: 01 19 FF FF FF 01 19 FF FF FF 01 19 FF FF FF 01
B0: 19 FF FF FF 01 19 FF FF FF 01 19 FF FF FF 01 19
C0: FF FF FF 01 19 FF FF FF 01 19 FF FF FF 01 19 FF
D0: FF FF 01 17 FF FF 7F 00 17 FF FF 7F 00 17 FF FF
E0: 7F 00 17 FF FF 7F 00 17 FF FF 7F 00 17 FF FF 7F
F0: 00 17 FF FF 7F 00 17 FF FF 7F 00 17 FF FF 7F 00
```

Bytes	**Description**
$00-$01	T/S pointer to fourth BAM sector (38/9)
$02	DOS version byte (0x43='C')
$03	Reserved
$04	Lowest track covered by this BAM (0x65=101)
$05	Highest+1 track covered by this BAM (0x97=151)
$06-$0A	BAM for track 101. The first byte shows the "blocks free" for this track, the remaining 4 show the BAM for the track.
$0B-$0F	BAM for track 102
...	...
$FB-$FF	BAM for track 150

Fourth BAM block 38/9

```
      00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
      -----------------------------------------------
00: 27 01 43 00 97 9B 17 FF FF 7F 00 17 FF FF 7F 00
10: 17 FF FF 7F 00 17 FF FF 7F 00 00 00 00 00 00 00
20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Bytes	**Description**
$00-$01	T/S pointer to first directory sector (39/1)

$02	DOS version byte (0x43='C')
$03	Reserved
$04	Lowest track covered by this BAM (0x97=151)
$05	Highest+1 track covered by this BAM (0x9B=155)
$06-$0A	BAM for track 151. The first byte shows the "blocks free" for this track, the remaining 4 show the BAM for the track.
$0B-$0F	BAM for track 152
...	...
$15-$19	BAM for track 154
$1A-$FF	Not used

15.9 The P00 image format

(This section was contributed by Peter Schepers and slightly edited by Marco van den Heuvel.)

These files were created for use in the PC64 emulator, written by Wolfgang Lorenz. Each one has the same layout with the filetype being stored in the DOS extension (i.e. Pxx is a PRG, Sxx is a SEQ, Uxx is a USR and Rxx is a RELative file), and the header is only 26 bytes long.

This is a dump of a Pxx file (PRG)...

```
        00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
        -----------------------------------------------
  0000: 43 36 34 46 69 6C 65 00 43 52 49 53 49 53 20 4D
  0010: 4F 55 4E 54 41 49 4E 00 00 00
```

Bytes	**Description**
$00-$06	ASCII string "C64File"
$07	Always $00
$08-$17	Filename in PETASCII, padded with $00 (not $A0, like a D64)
$18	Always $00
$19	REL file record size ($00 if not a REL file)
$1A-??	Program data

The 'xx' in the extension of the file is usually 00, except when we have two DOS filenames which would be the same, but the C64 filenames are different! If we have two C64 filenames which are the same, they *cannot* co-exist in the same directory. If we have two files which do convert down to be the same DOS filename, the extension is incremented until an unused one is found (x01, x02, x03, up to x99). We can have up to 99 different C64 files with the same corresponding DOS names as that's all the extension will hold (from P00 to P99).

Each PC64 file only has one entry, there are no multi-file containers allowed. This could result in a large number of these files in a directory, even for only a few programs, as each C64 file will result in a PC64 file entry. The best use for a PC64 file is a single-file program, one which does not load anything else.

16 Acknowledgments

VICE derives from X64, the first Commodore 64 emulator for the X Window System. Here is an informal list of the people who were mostly involved in the development of X64 and VICE:

The VICE core team:

- **Dag Lem** Implemented the reSID SID emulation engine and video hardware scaling.
- **Andreas Matthies** Improved the datasette support, the VIC20 video emulation and some UI stuff in the Win32 and DOS ports. He also wrote the BeOS port and implemented video/audio capture support. Improved history recording/playback and implemented support for video recording and the netlink feature. Made the Win32 user changable keyboard shortcut system. Improved CIA and VIA emulation. Worked on x64sc, especially interrupt timing. Wrote test programs. Various bug(fixe)s. ;-)
- **Martin Pottendorfer** Implemented the Gnome Port based on Oliver Schaertels GTK+ port. Added support code for internationalization based on gettext. Improved the *nix fullscreen support. Translated the UI to German. Implemented the fliplists + UI (*nix).
- **Spiro Trikaliotis** Wrote the Win32 console implementation for the built-in monitor, corrected some REU related bugs, improved the CIA emulation, added com-port CIA support to the Win32 port, added text copy and paste support to the Win32 port, added support for the TFE and RR-Net (cs8900a), and wrote some further patches.
- **Marco van den Heuvel** Translated the UI to Dutch. Made the internationalization support for the Win32 and Amiga ports. Wrote the GEO-RAM and RamCart cartridge code. Wrote the c64 +60K, +256K and 256K memory expansions code. Wrote the pet REU code. Wrote the plus4 memory expansions code. Made the ethernet support for the DOS port. Maintains the QNX 4.x, QNX 6.x, Solaris, Openserver, Unixware, Minix 3.x, Amiga, Syllable and OS/2 binary ports. Maintains the Win64 and Open Watcom project files. Maintains the SDL port(s). Added new .crt support. Added new screenshot formats. Added new sound recording support. Added SIDcart support for xpet, xplus4 and xvic. Improved the MMC64 emulation. Added 2 MHz mode and banks 2/3 support for x128. Added the various userport joystick emulations. Added text copy and paste support to the Amiga and BeOS ports. Added DQBB and ISEPIC cartridge support. Added SFX Sound Sampler and SFX Sound Expander support. Added PCI support to the Amiga and DOS ports. Rewrote the sound system into a modular one, added always mono and always stereo support for the sound output. Added the RTC system. Added digiblaster support. And lots of other fixes and improvements.
- **Christian Vogelgsang** Maintains the Mac OS X port. Added Intel Mac support and universal binary creation. Wrote the build scripts for all external Mac libraries and the bindist bundle tool. Improved the TFE chip emulation. Added some GTK+ fixes.
- **Fabrizio Gennari** Added some improvements to the DOS and GTK+ ports. Changed the Windows video to use GDI as fallback, making it compile without DX if needed.
- **Daniel Kahlin** Worked on DTV VIC emulation, palette, DTV SID support in resid, better DMA/Blitter support and did lots of refactoring. Added new monitor commands and features. Improved the VIC emulation for xvic. Made MIDI driver code for Win32. Rewrote the xvic cartridge system. Added Mega-Cart and Final Expansion

V3.2 support to xvic. Wrote large parts of the new VIC-II emulation used in x64sc, especially the dot clock domain emulation. Wrote many test programs for hardware analysis.

- **Antti S. Lankila** Made the ReSID-fp engine, rewrote the PAL emulation code and fixed the sound core for lower latency. Rewrote DTV SID support (ReSID-dtv). Improved 1541 drive rotation emulation. Worked on x64sc.
- **Groepaz** Added new more precise CRT emulation. Added support for the new cartridge system and many new cartridges. Fixed up parts of cartconv. Implemented many bug fixes. Wrote test programs.
- **Ingo Korb** Corrected block allocation and interleave for c1541/vdrive, added rudimentary xplus4 tape recording support, corrected a case of missing Pi symbols in petcat, changed the trap opcode byte, stopped the high-level serial drive code from responding to addresses 16-30 and was forced to update this entry himself.
- **Errol Smith** Improved VDC emulation.
- **Olaf Seibert** Contributed some PET, including PET DWW hires, Xaw, lightpen, hardware scaling, and disk drive patches. Maintains the Xaw UI.
- **Stefan Haubenthal** Added some Amiga fixes.
- **Thomas Giesel** Added new monitor commands and features.
- **Marcus Sutton** Made some console, dialog and joystick fixes for the BeOS port. Maintains the BeOS port.

  Ulrich Schulz Maintains the Dingoo port(s).
- **Kajtar Zsolt** Wrote the IDE64 interface emulation, 2000/4000 drive emulation and did some fixes.

Former/inactive team members:

- **Hannu Nuotio** Copyright © 2007-2011 Implemented DTV flash emulation, DTV support in the monitor, large parts of the DTV VIC, burst mode and skip cycle emulation as well as many other things. Added NEOS and Amiga mouse, paddle and light pen support. Added new monitor commands and features, including memmap. Made MIDI support and OSS MIDI driver. Implemented most of the SDL UI. Rewrote xvic CPU/VIC-I core for cycle based emulation. Implemented C64 cartridge snapshot support. Initiated and worked on all parts of implementing x64sc. Wrote test programs.
- **Andreas Boose** Copyright © 1998-2010 Gave lots of information and bug reports about the VIC-II, the 6510 and the CIAs; moreover, he wrote several test-routines that were used to improve the emulation. He also added cartridge support and has been the main head behind the drive and datasette emulation since version 0.15. Also added several UI elements to the DOS, Win32 and *nix ports. He rewrote the C128 emulation adding Z80 mode, C64 mode and function ROM support, wrote the screenshot and the event system and started the plus4 emulator. Restructured the serial bus emulation and added realdrive and rawdrive support.
- **Tibor Biczo** Copyright © 1998-2010 Improved the Win32 port and plus4 emulation.
- **M. Kiesel** Copyright © 2007-2010 Started implementing x64dtv. The C64DTV memory model and early versions of the DMA and Blitter engine have been implemented by him. Added new monitor commands and features.

- **Andreas Dehmel** Copyright © 1999-2007 Wrote the Acorn RISC OS port.
- **David Hansel** Copyright © 2003-2005 Wrote the Star NL10 printer driver, implemented IEC devices and improved the tape emulation.
- **Markus Brenner** Copyright © 2000-2004 Added VDC emulation to x128 and added support for some more cartridges.
- **Thomas Bretz** Copyright © 1999-2004 Started the OS/2 port.
- **Daniel Sladic** Copyright © 1997-2001 Started the work on hardware-level 1541 emulation and wrote the new monitor introduced with VICE 0.15.
- **Andr Fachat** Copyright © 1996-2001 Wrote the PET and CBM-II emulators, the CIA and VIA emulation, the IEEE488 interface, implemented the IEC serial bus in 'xvic' and made tons of bug fixes.
- **Ettore Perazzoli** Copyright © 1996-1999 Made the 6510, VIC-II, VIC-I and CRTC emulations, part of the hardware-level 1541 emulation, speed optimizations, bug fixes, the event-driven cycle-exact engine, the Xt/Xaw/Xfwf-based GUI for X11, a general code reorganization, the new resource handling, most of the documentation. He also wrote the DOS port and the initial Win32 port (well, somebody had to do it).
- **Teemu Rantanen** Copyright © 1993-1994, 1997-1999 Implemented the SID emulation and the trap-based disk drive and serial bus implementation; added support for multiple display depths under X11. Also wrote `c1541`
- **Jouko Valta** Copyright © 1993-1996 Wrote `petcat` and `c1541`, `T64` handling, user service and maintenance (most of the work in x64 0.3.x was made by him); retired from the project in July 96, after VICE 0.10.0.
- **Jarkko Sonninen** Copyright © 1993-1994 He was the founder of the project, wrote the old version of the 6502 emulation and the XDebugger, and retired from the project after x64 0.2.1.

Internationalization Team:

- **Mikkel Holm Olsen** Provided the Danish user interface translations and fixed a few monitor bugs.

  Manuel Antonio Rodriguez Bas Provided the Spanish user interface translations.

  Paul Dub From Rivire-du-Loup, Qubec, provided the French user interface translations.

  Czirkos Zoltan and **Karai Csaba** Provided the Hungarian user interface translations.

  Andrea Musuruane Provided the Italian user interface translations.

  Jesse Lee Provided the Korean user interface translations.

  Jarek Sobolewski Provided the new Polish user interface translations.

  Michael Litvinov Provided the Russian user interface translations.

  Peter Krefting Provided the Swedish user interface translations.

  Emir Akaydin Provided the Turkish user interface translations (in world record time).

External contributors:

- **Christian Bauer** Wrote the very interesting "VIC article" from which we got invaluable information about the VIC-II chip: without this, the VIC-II implementation would not have been possible.

- **Eliseo Bianchi** Provided the italian Amiga translations.
- **ck!** Provided a win32 cbm character font.
- **iAN CooG** Added win32 vsid GUI and contributed various patches.
- **Mike Dawson** Provided the GP2X port.
- **Paul David Doherty** Wrote `zip2disk`, on which the Zipcode support in `c1541` is based.
- **Peter Edwards** Implemented the SDL UI slider control and fixed some GP2X/Dingoo SDL UI issues.
- **Daniel Fandrich** Contributed some disk drive patches.
- **Dirk Farin** Rewrote the MITSHM code.
- **Georg Feil** Added support for toggling CB2 sound output line in the PET emulator.
- **Peter Andrew Felvegi aka Petschy** Fixed a couple of bugs in the fast serial emulation.
- **Ricardo Ferreira** Contributed the `unlynx` and `system` commands in `c1541` and added aRts sound support.
- **Flooder** Provided parts of the Polish user interface translations.
- **Robert H. Forsman Jr.** Provided parts of the widget set for implementing the Xaw GUI.
- **Peter Gordon** Provided support for native AmigaOS4 compiling.
- **Richard Hable** Contributed the initial version of the REU emulation.
- **Shawn Hargreaves** Wrote Allegro, the graphics and audio library used in the DOS version.
- **Ville-Matias Heikkila** Rewrote the vic20 sound code.
- **David Holz** Provided a label file which gives the built-in monitor the labels for the C64.
- **Nathan Huizinga** Added support for Expert and Super Snapshot carts.
- **Craig Jackson** Contributed miscellaneous patches in the old X64 times.
- **Dirk Jagdmann** Wrote the Catweasel sound driver.
- **Lasse Jyrkinen** Contributed miscellaneous patches in the old X64 times.
- **Peter Karlsson** Provided the swedish UI translations in the past.
- **Greg King** Added a working RTC to the emulation of the IDE64 cartridge.
- **Michael Klein** Contributed the ESD sound driver, basic support for the OPENCBM library and some other patches.
- **Frank Knig** Contributed the Win32 joystick autofire feature.
- **Bernd Kortz** Provided some fixes for ZETA and the ZETA binary package.
- **Bernhard Kuhn** Made some joystick improvements for Linux.
- **Alexander Lehmann** Added complete support for all the VIC20 memory configurations for the old VICE 0.12.
- **Ilkka "itix" Lehtoranta** Provided the routines for the cybergraphics support for the Amiga ports.
- **Magnus Lind** Atari ST mouse and Atari CX-22 trackball emulation and pixel aspect fixes.
- **Wolfgang Lorenz** Wrote an excellent 6510 test suite that helped us to debug the CPU emulation.

- **Marko Mkel** Wrote lots of CPU documentation. Wrote the VIC Flash Plugin cartridge emulation in xvic.
- **Robert W. McMullen** Provided parts of the widget set for implementing the Xaw GUI.
- **Jennifer Medkief** Is in charge of checking up on the GUIs for elements that are wrong, unaccessable, and missing. Future user documentation writer.
- **Dan Miner** Contributed some patches to the fast disk drive emulation.
- **Luca Montecchiani** Contributed a new Unix joystick driver.
- **Wolfgang Moser** Provided small optimization fixes to the GCR code, provided an excellent REU test suite and added REU fixes, and is always the good guy reviewing and commenting changes in the background.
- **Roberto Muscedere** Improved support for REL files.
- **Tomi Ollila** Donated `findpath.c`.
- **Per Olofsson** Digitalized the C64 colors used in the (old) default palette.
- **Lasse rni** Contributed the Windows Multimedia sound driver
- **Helfried Peyrl** Supplied a patch that fixes REL file records larger 256 bytes when using vdrive.
- **Frank Prindle** Contributed some patches.
- **Giuliano Procida** Used to maintain the VICE `deb` package for the Debian distribution, and also helped proofreading the documentation.
- **Vesa-Matti Puro** Wrote the very first 6502 CPU emulator in x64 0.1.0. That was the beginning of the story. . .
- **Rami Rasanen** Rewrote the VIC20 sound code.
- **David Roden** Fixed various issues related to ffmpeg settings.
- **Pablo Roldn** Contributed initial patch for VIC-II PAL-N model selection.
- **Mathias Roslund** Provided the AmigaOS4 port.
- **Gunnar Ruthenberg** Provided some VIC-II enhancements and improved the Win32 port.
- **Johan Samuelsson** Provided the Swedish Amiga translations.
- **Oliver Schaertel** Wrote the X11 full screen, parts of custom ROM set support and 1351 mouse emulation for unix.
- **Peter Schepers** Contributed a document describing the G64 image format.
- **Michael Schwendt** Helped with the SID (audio) chip emulation, bringing important suggestions and bug reports, as well as the wave tables and filter emulation from his SIDplay emulator.
- **Heiko Selber** Contributed some VIC20 I/O patches.
- **John Selck** Improved the video rendering and added the fast PAL emulation. Implemented new color generation based on P. Timmermanns knowledge.
- **Chris Sharp** Wrote the AIX sound driver.
- **Andr351 "JoBBo" Siegel** Provided the native MorphOS icons.
- **Harry "Piru" Sintonen** Provided lots of fixes and improvements for the Amiga ports.
- **Manfred Spraul** Wrote the Win32 text lister.

- **Markus Stehr** Provided the MMC64 emulation.
- **Dominique Strigl** Contributed miscellaneous patches in the old X64 times.
- **Samuli Suominen** Fixed XShm includes for newer xextproto versions and updated libpng check for newer versions.
- **Steven Tieu** Added initial support for 16/24 bpp X11 displays.
- **Philip Timmermann** Did a lot of research about the VIC-II colors.
- **Brian Totty** Provided parts of the widget set for implementing the Xaw GUI.
- **Mustafa "GnoStiC" Tufan** Made improvements to the GP2x port.
- **Lionel Ulmer** Implemented joystick support for Linux and a first try of a SID emulation for SGI machines.
- **Krister Walfridsson** Implemented joystick and sound support for NetBSD.
- **webulator** Provided Win32 drag & drop support
- **Robert Willie** Added some additional commands to the fsdevice emulation.
- **Peter Weighill** Gave many ideas and contributed the ROM patcher.
- **Gerhard Wesp** Contributed the `extract` command in `c1541`.
- **Maciej Witkowiak** Did some IDE64 and C1541 fixes.

(We hope we have not forgotten anybody; if you think we have, please tell us.)

The people around the world providing results from running our test programs on various machines deserve a special mention:

- **hedning** (Drean C64 PAL-N, various C64 PAL boxes)
- **Jason Compton** (Various C64 and C128 NTSC boxes)
- **The Woz** (Drean C64 PAL-N)
- **Thierry** (Drean C64 PAL-N)
- **MOS6569** (C64C PAL)
- **Mike** (VIC-20 PAL)
- **Wilson** (VIC-20 NTSC)
- **Vicassembly** (VIC-20 NTSC)
- **David "jbevren" Wood** (C64 NTSC-OLD)

Thanks also to everyone else for sending suggestions, ideas, bug reports, questions and requests. In particular, a warm thanks goes to the following people:

- **Lutz Sammer**
- **Ralph Mason**
- **George Caswell**
- **Jasper Phillips**
- **Luca Forcucci**
- **Asger Alstrup**
- **Bernhard Schwall**
- **Salvatore Valente**
- **Arthur Hagen**

- **Douglas Carmichael**
- **Ferenc Veres**
- **Frank Reichel**
- **Ullrich von Bassewitz**
- **Holger Busse**
- **David "jbevren" Wood**
- **Gary Glenn**

Last but not least, a very special thank to Andreas Arens, Lutz Sammer, Edgar Tornig, Christian Bauer, Wolfgang Lorenz, Miha Peternel, Per Hkan Sundell and David Horrocks for writing cool emulators to compete with. `:-)`

17 Copyright

18 Contact information

18.1 VICE home page

You can find the latest news about VICE at the official VICE home page:

http://vice-emu.sourceforge.net/

VICE has moved its source repository to public services provided by SourceForge. You can find it at

http://sf.net/projects/vice-emu.

We would like to thank the SourceForge staff for that help.

If you are going to report a bug, please check those pages *first*; it is possible that the problem you encountered has already been fixed with a newer version.

Please, also have a look at the VICE knowledge base at

http://vicekb.trikaliotis.net/

18.2 How to send feedback

Before contacting us, have a look at the VICE knowledge base at http://vicekb.trikaliotis.net/ if your question is answered there. Keep in mind that we work on VICE in our spare-time, so the more time we don't need to answer the same questions over and over again, the more time we have to improve the emulation itself. On the other hand, that does not mean that you should not contact us, especially if you find bugs or have suggestions which might improve the emulation.

Bug reports, suggestions, support requests should be directed to the SourceForge trackers at

- http://sourceforge.net/tracker/?group_id=223021.

This way, you, the users, and we, the developers, can track what has been reported and what has been already fixed. Ideally, also sent the report to the mailing address of the Vice team at

- VICE Mailing List (vice-emu-mail@lists.sourceforge.net) for all general questions, bug reports, suggestions.

You can also contact (some of) us on IRC, at #vice-dev on freenode.

It's always nice to receive feedback and/or bugreports about VICE, but please read these few notes before sending mail to anybody in the team.

- Please put the word '`VICE`' *in all capitals* in your subject line (e.g., '`VICE fails to run game XXX`'). This helps mail splitting and reduces chances that your message is unintentionally deleted, forgotten or lost.
- Please don't send any HTML mail (we really hate that!). If you use M$ Outlook or Netscape Communicator, make sure you turn off the "rich text" (HTML) feature.
- Please don't send *any* binaries without asking first.
- Please read the following documents carefully before reporting a bug or a problem you cannot solve:
  - the VICE documentation (you are reading it!);
  - the VICE FAQ (it is available on the Internet, and reachable from the VICE home page: http://vice-emu.sourceforge.net/);
  - The VICE knowledge base (it is available on the Internet at http://vicekb.trikaliotis.net/);
  - the `comp.emulators.cbm` and `comp.sys.cbm` FAQs (see Section 18.5 [FAQs you should read], page 178).
- When you report a bug, please try to be as accurate as possible and describe how it can be reproduced to the very detail. You should also tell us what machine you are running on, what operating system you are using as well as the version of it.
- Please don't ask us how to transfer original C64 disk or tapes to your PC; this has been asked a gazillion times through email. To transfer disks, you can use the Star Commander (http://sta.c64.org/sc.html) on DOS, and OpenCBM (http://www.trikaliotis.net/opencbm) on Windows and Linux. And no, you cannot read C64 disks with your old 5"1/4 PC drive.
- Please don't ask us where to find games for the emulator on the Internet.
- Please don't ask us when the next version will be out, because we really don't know.
- Please write in English.

In any case, we would be *really* glad to receive your comments about VICE. We cannot always answer all the email, but we surely read all of it.

Thanks!

18.3 How to contribute

If you want to make a major contribution, please *ask* first. It has already happened a couple of times that somebody started working at something that had already been done but not released to the public yet, and we really do *not* want anybody to waste time.

If you are going to make a patch, please make sure the patch is relative to the very latest version, and provide us with the following:

- Make sure you are giving us a diff against the latest Subversion trunk version of VICE. For instructions on accessing the Subversion repository, first read http://sourceforge.net/svn/?group_id=223021 and get it with the command:

  ‘`svn co https://vice-emu.svn.sourceforge.net/svnroot/vice-emu/trunk vice-src`’
- send a unified diff file against the trunk version of VICE (see above bullet point) by using the command: ‘`svn diff`’ inside of the SVN workspace you checked out before.
- If you cannot use SVN for one or the other reason, send a unified diff file containing all the changes you have made (‘`diff -u`’; please don’t use plain ‘`diff`’), as it adds much work for us to get it working;
- GNU-style ‘`ChangeLog`’ entries with a description of the changes you have made (look at the ‘`ChangeLog`’s provided with the original VICE sources for an example).

This is very important, and makes adding patches much smoother and safer.

People willing to port VICE to other platforms are always welcome. But notice from experience it will take at least a full year of continuous work to write a well working and stable port.

18.4 Interesting newsgroups

There are some Usenet newsgroups you might be interested in:

- `comp.emulators.cbm`, discussing about emulators of Commodore 8-bit machines (definitely not Amiga emulators).
- `comp.sys.cbm`, discussing various topics regarding real Commodore 8-bit machines. This newsgroup is mainly for people who actually use original Commodore equipment (so please don’t talk about emulation here).
- `comp.emulators.misc`, discussing emulators in general.

18.5 FAQs you should read

We recommend reading the `comp.emulators.cbm` and `comp.sys.cbm` FAQs, which are posted regularly on the corresponding newsgroups and are also available via FTP from `ftp://rtfm.mit.edu`.

Concept Index

+

+cart 53

-

-? 14
-1 15
-10 15
-11 15
-256k, +256k 68
-256kbase 68
-256kimage 68
-40col 70
-8 15
-80col 70
-9 15
-acia1, +acia1 65
-attach10ro 15
-attach10rw 15
-attach11ro 15
-attach11rw 15
-attach8ro 15
-attach8rw 15

-attach9ro ... 15
-attach9rw ... 15
-autoload ... 15
-autostart ... 15, 16
-autostart-handle-tde, +autostart-handle-tde ... 15
-autostart-warp, +autostart-warp ... 15
-autostartprgdiskimage ... 15
-autostartprgmode ... 15
-autostartwithcolon, +autostartwithcolon ... 15
-basic ... 67, 77, 86
-basic1, +basic1 ... 84
-basic1char, +basic1char ... 84
-basic64 ... 69
-basichi ... 69
-basiclo ... 69
-basicload, +basicload ... 15
-bdesymkeymap, -bdeposkeymap ... 30
-brightness ... 61, 69, 75, 79, 83
-buksymkeymap, -bukposkeymap ... 30
-burstmod ... 66
-c ... 118
-c1hi ... 80
-c1lo ... 79
-c2hi ... 80
-c2lo ... 80
-c64dtvromimage ... 71
-c64dtvromrw, +c64dtvromrw ... 71
-cart1 ... 86
-cart16 ... 53
-cart2 ... 73, 77, 86
-cart4 ... 73, 77, 86
-cart6 ... 73, 77, 86
-cart8 ... 53
-cartA ... 73, 77
-cartap ... 53
-cartar2 ... 53
-cartar3 ... 53
-cartar4 ... 54
-cartar5 ... 54
-cartB ... 73, 77
-cartcap ... 54
-cartcomal ... 54
-cartcrt ... 53
-cartdep256 ... 54
-cartdep64 ... 54
-cartdep7x8 ... 54
-cartdin ... 54
-cartdqbb ... 54
-cartdsm ... 54
-carteasy ... 54
-cartepyx ... 54
-cartexos ... 54
-cartexpert ... 55
-cartfc1 ... 55
-cartfc3 ... 55
-cartfcplus ... 55
-cartfe ... 74
-cartff ... 55
-cartfm ... 55
-cartfp ... 55, 74
-cartgeneric ... 73
-cartgeoram ... 65
-cartgk ... 55
-cartgs ... 55
-cartide64 ... 55
-cartieee ... 56
-cartisepic ... 56
-cartkcs ... 56
-cartks ... 56
-cartmach5 ... 57
-cartmd ... 57
-cartmega ... 73
-cartmf ... 57
-cartmikro ... 57
-cartmmc64 ... 57
-cartmmcr ... 57
-cartmv ... 58
-cartocean ... 58
-cartp64 ... 58
-cartpf ... 58
-cartramcart ... 58
-cartrep256 ... 58
-cartreset, +cartreset ... 53
-cartreu ... 66
-cartross ... 58
-cartrr ... 58
-cartru ... 58
-carts64 ... 58
-cartsb ... 59
-cartse5 ... 59
-cartsg ... 59
-cartsimon ... 59
-cartss4 ... 59
-cartss5 ... 59
-cartstar ... 59
-cartultimax ... 53
-cartwl ... 59
-cartws ... 59
-cartzaxxon ... 59
-chargde ... 70
-chargen ... 67, 70, 77, 83, 86
-chargfr ... 70
-chargse ... 70
-chdir ... 15
-cia1model ... 59
-cia2model ... 59
-ciamodel ... 59
-config ... 14
-confirmexit, +confirmexit ... 47
-console ... 15
-contrast ... 61, 69, 75, 78, 83
-core, +core ... 47
-crtblur ... 62, 69, 75, 79, 83
-Crtcdscan, +Crtcdscan ... 82
-Crtcdsize, +Crtcdsize ... 82
-Crtcextpal ... 82

-Crtcfulldevice ... 82
-Crtchwscale, +Crtchwscale ... 82
-Crtcintpal ... 82
-Crtcpalette ... 82
-Crtcscale2x, +Crtcscale2x ... 82
-Crtcvcache, +Crtcvcache ... 82
-CrtcVidmodefullmode ... 83
-CrtcXRANDRfullmode ... 82
-crtscanlineshade ... 62, 69, 75, 79, 83
-cs256k ... 80
-default ... 14
-device10 ... 39
-device11 ... 39
-device4 ... 40
-device5 ... 40
-device8 ... 39
-device9 ... 39
-diagpin, +diagpin ... 84
-digiblaster, +digiblaster ... 79
-digimax, +digimax ... 65
-digimaxbase ... 65
-directory ... 46
-displaydepth ... 29
-doodlecrtctextcolor ... 103
-doodlemc ... 103
-doodleoversize ... 103
-doodletedlum ... 103
-dos1001 ... 36
-dos1541 ... 36
-dos1541II ... 36
-dos1551 ... 36
-dos1570 ... 36
-dos1571 ... 36
-dos1571cr ... 36
-dos1581 ... 36
-dos2000 ... 36
-dos2031 ... 36
-dos2040 ... 36
-dos3040 ... 36
-dos4000 ... 36
-dos4040 ... 36
-dqbb, +dqbb ... 54
-dqbbimage ... 54
-dqbbimagerw, +dqbbimagerw ... 54
-drive10extend ... 36
-drive10idle ... 36
-drive10profdos, +drive10profdos ... 37
-drive10ram2000, +drive10ram2000 ... 36
-drive10ram4000, +drive10ram4000 ... 37
-drive10ram6000, +drive10ram6000 ... 37
-drive10ram8000, +drive10ram8000 ... 37
-drive10rama000, +drive10rama000 ... 37
-drive10type ... 35
-drive11extend ... 36
-drive11idle ... 36
-drive11profdos, +drive11profdos ... 37
-drive11ram2000, +drive11ram2000 ... 36
-drive11ram4000, +drive11ram4000 ... 37
-drive11ram6000, +drive11ram6000 ... 37
-drive11ram8000, +drive11ram8000 ... 37
-drive11rama000, +drive11rama000 ... 37
-drive11type ... 35
-drive8extend ... 36
-drive8idle ... 36
-drive8profdos, +drive8profdos ... 37
-drive8ram2000, +drive8ram2000 ... 36
-drive8ram4000, +drive8ram4000 ... 37
-drive8ram6000, +drive8ram6000 ... 37
-drive8ram8000, +drive8ram8000 ... 37
-drive8rama000, +drive8rama000 ... 37
-drive8type ... 35
-drive9extend ... 36
-drive9idle ... 36
-drive9profdos, +drive9profdos ... 37
-drive9ram2000, +drive9ram2000 ... 36
-drive9ram4000, +drive9ram4000 ... 37
-drive9ram6000, +drive9ram6000 ... 37
-drive9ram8000, +drive9ram8000 ... 37
-drive9rama000, +drive9rama000 ... 37
-drive9type ... 35
-drivesound, +drivesound ... 35
-dsresetwithcpu, +dsresetwithcpu ... 33
-dsspeedtuning ... 33
-dszerogapdelay ... 33
-dtvblitterlog, +dtvblitterlog ... 72
-dtvdmalog, +dtvdmalog ... 72
-dtvflashlog, +dtvflashlog ... 72
-dtvrev ... 71
-easyflashcrtwrite, +easyflashcrtwrite ... 54
-easyflashjumper, +easyflashjumper ... 54
-editor ... 83
-eoiblank, +eoiblank ... 84
-expert, +expert ... 55
-expertimagename ... 55
-expertimagerw, +expertimagerw ... 55
-extfrom ... 70
-extfunc, +extfunc ... 70
-extrajoydev1 ... 30
-extrajoydev2 ... 30
-f ... 116, 119
-fewriteback, +fewriteback ... 74
-ffmpegaudiobitrate ... 103
-ffmpegvideobitrate ... 103
-flipname ... 39
-fpwriteback, +fpwriteback ... 74
-fs10 ... 39
-fs11 ... 39
-fs8 ... 39
-fs9 ... 39
-fsflash ... 71
-fullscreen, +fullscreen ... 29
-functionhi ... 79
-functionlo ... 79
-gamma ... 61, 69, 75, 79, 83
-georam, +georam ... 65
-georamimage ... 65

-georamimagerw, +georamimagerw 66
-georamsize 66
-gluelogictype 67
-go64, +go64 70
-grsymkeymap, -grposkeymap 30
-h 118
-h1024k 80
-h256k 80
-h4096k 80
-help 14, 118
-htmlbrowser 46
-hummeradc, +hummeradc 72
-i 115
-ic 118
-IDE64autosize1, +IDE64autosize1,
-IDE64autosize2, +IDE64autosize2,
-IDE64autosize3, +IDE64autosize3,
-IDE64autosize4, +IDE64autosize4 56
-IDE64cyl1 55
-IDE64cyl2 55
-IDE64cyl3 55
-IDE64cyl4 55
-IDE64hds1 55
-IDE64hds2 56
-IDE64hds3 56
-IDE64hds4 56
-IDE64image1 55
-IDE64image2 55
-IDE64image3 55
-IDE64image4 55
-IDE64sec1 56
-IDE64sec2 56
-IDE64sec3 56
-IDE64sec4 56
-IDE64version4, +IDE64version4 56
-iecdevice10, +iecdevice10 39
-iecdevice11, +iecdevice11 39
-iecdevice4, +iecdevice4 40
-iecdevice5, +iecdevice5 40
-iecdevice8, +iecdevice8 39
-iecdevice9, +iecdevice9 39
-ieee488, +ieee488 ..... 56, 74
-ieee488image 56
-initbreak 45
-install, +install 29
-intfrom 70
-intfunc, +intfunc 70
-isepic, +isepic 56
-isepicimagename 56
-isepicimagerw, +isepicimagerw 56
-joydev1 30
-joydev2 30
-k 119
-k<version> 119
-keepenv 87
-kernal 67, 69, 77, 83, 86
-kernal64 69
-kernalde 69
-kernalfi 69
-kernalfr 70
-kernalit 70
-kernalno 70
-kernalrev 67
-kernalse 70
-keybuf 15
-keymap 30
-l 116, 119
-lightpen, +lightpen 31
-lightpentype 31
-logfile 15
-mcnvramfile 74
-mcnvramwriteback, +mcnvramwriteback 74
-memory 76
-midi, +midi 65
-mididrv 65
-midiin 65
-midiout 65
-miditype 65
-mitshm, +mitshm 13, 29
-mmc64, +mmc64 57
-mmc64bios 57
-mmc64bioswrite 57
-mmc64image 57
-mmc64readonly 57
-mmc64readwrite 57
-mmcrcardimage 57
-mmcrcardrw, +mmcrcardrw 57
-mmcreepromimage 57
-mmcreepromrw, +mmcreepromrw 57
-mmcrimagerw, +mmcrimagerw 57
-mmcrrescue, +mmcrrescue 57
-model 83
-modelline 87
-moncommands 45
-mouse, +mouse 31
-mouseport 31
-mousetype 31
-myaciadev 43
-n 115
-nc 118
-newluminance, +newluminance 61
-nh 118
-ntsc 45
-ntscold 45
-o 115
-o <name> 119
-oddlinesoffset 62, 69, 75, 79, 83
-oddlinesphase 62, 69, 75, 79, 83
-OEMjoy, +OEMjoy 78
-pal 45
-paln 45
-parallel10 36
-parallel11 36
-parallel8 36
-parallel9 36
-petdww, +petdww 84

-petdwwimage ... 84
-petram9, +petram9 ... 84
-petramA, +petramA ... 84
-petreu, +petreu ... 84
-petreuimage ... 84
-petreuramsize ... 84
-playback ... 103
-plus256k, +plus256k ... 68
-plus256kimage ... 68
-plus60k, +plus60k ... 67
-plus60kbase ... 67
-plus60kimage ... 67
-poskeymap ... 30
-pr4drv ... 41
-pr4output ... 41
-pr4txtdev ... 41
-pr5drv ... 41
-pr5output ... 41
-pr5txtdev ... 41
-profdos1571 ... 37
-prtxtdev1 ... 41
-prtxtdev2 ... 41
-prtxtdev3 ... 41
-pruser, +pruser ... 41
-pruserdrv ... 41
-pruseroutput ... 41
-prusertxtdev ... 41
-ps2mouse, +ps2mouse ... 72
-r ... 116
-ram08, +ram08 ... 86
-ram1, +ram1 ... 86
-ram2, +ram2 ... 86
-ram4, +ram4 ... 86
-ram6, +ram6 ... 86
-ramC, +ramC ... 86
-ramcart, +ramcart ... 58
-ramcartimage ... 58
-ramcartimagerw, +ramcartimagerw ... 58
-ramcartsize ... 58
-raminitpatterninvert ... 45
-raminitstartvalue ... 45
-raminitvalueinvert ... 45
-ramsize ... 80, 86
-refresh ... 27
-remotemonitor, +remotemonitor ... 45
-remotemonitoraddress ... 45
-residfilterbias ... 64
-residgain ... 64
-residpass ... 64
-residsamp ... 63
-reu, +reu ... 66
-reuimage ... 66
-reuimagerw, +reuimagerw ... 66
-reusize ... 66
-rom9 ... 83
-romA ... 83
-romB ... 83
-rrbankjumper, +rrbankjumper ... 58
-rrbioswrite, +rrbioswrite ... 58
-rrflashjumper, +rrflashjumper ... 58
-rsdev1 ... 43
-rsdev1baud ... 43
-rsdev2 ... 43
-rsdev2baud ... 43
-rsdev3 ... 43
-rsdev3baud ... 43
-rsdev4 ... 43
-rsdev4baud ... 43
-rsuser, +rsuser ... 44
-rsuserbaud ... 44
-rsuserdev ... 44
-saturation ... 61, 69, 75, 78, 83
-saveres, +saveres ... 46
-sfxse, +sfxse ... 66
-sfxsetype ... 66
-sfxss, +sfxss ... 66
-sidcart, +sidcart ... 74, 79, 84
-sidcartjoy, +sidcartjoy ... 79
-sidenginemodel ... 63
-sidfilters, +sidfilters ... 63
-sidstereo ... 63
-sidstereoaddress ... 63
-skip <n> ... 118
-sound, +sound ... 33
-soundarg ... 33
-soundbufsize ... 33
-sounddev ... 33
-soundfragsize ... 33
-soundrate ... 33
-soundrecarg ... 33
-soundrecdev ... 33
-soundsync ... 33
-speech, +speech ... 79
-speechrom ... 79
-speed ... 27
-superpet, +superpet ... 84
-symdekeymap ... 30
-symkeymap ... 30
-t ... 115
-TEDdscan, +TEDdscan ... 78
-TEDdsize, +TEDdsize ... 78
-TEDextpal ... 78
-TEDfulldevice ... 78
-TEDhwscale, +TEDhwscale ... 78
-TEDintpal ... 78
-TEDpalette ... 78
-TEDscale2x, +TEDscale2x ... 78
-TEDvcache, +TEDvcache ... 78
-TEDVidmodefullmode ... 78
-TEDXRANDRfullmode ... 78
-text ... 118
-tfe, +tfe ... 66
-tfeif ... 66
-tferrnet, +tferrnet ... 66
-tint ... 62, 69, 75, 79, 83
-truedrive, +truedrive ... 35

-trueflashfs, +trueflashfs ... 71
-tune <number> ... 87
-userportdac, +userportdac ... 84
-usevicii, +usevicii ... 86
-v ... 118
-VDC16KB ... 69
-VDC64KB ... 69
-VDCdscan, +VDCdscan ... 68
-VDCdsize, +VDCdsize ... 68
-VDCextpal ... 68
-VDCfulldevice ... 68
-VDChwscale, +VDChwscale ... 68
-VDCintpal ... 68
-VDCpalette ... 68
-VDCRevision ... 69
-VDCvcache, +VDCvcache ... 68
-VDCVidmodefullmode ... 68
-VDCXRANDRfullmode ... 68
-verbose ... 15
-VICdscan, +VICdscan ... 74
-VICdsize, +VICdsize ... 74
-VICextpal ... 75
-VICfulldevice ... 75
-VIChwscale, +VIChwscale ... 75
-VICIIborders ... 61
-VICIIchecksb, +VICIIchecksb ... 60
-VICIIchecksb, +VICIIchecksb ... 60
-VICIIcheckss, +VICIIcheckss ... 60
-VICIIdscan, +VICIIdscan ... 61
-VICIIdsize, +VICIIdsize ... 60
-VICIIextpal ... 61
-VICIIfulldevice ... 61
-VICIIhwscale, +VICIIhwscale ... 61
-VICIIintpal ... 61
-VICIImodel ... 61
-VICIIpalette ... 61
-VICIIscale2x, +VICIIscale2x ... 61
-VICIIvcache, +VICIIvcache ... 60
-VICIIVidmodefullmode ... 61
-VICIIXRANDRfullmode ... 61
-VICintpal ... 75
-VICpalette ... 75
-VICscale2x, +VICscale2x ... 75
-VICvcache, +VICvcache ... 74
-VICVidmodefullmode ... 75
-VICXRANDRfullmode ... 75
-virtualdev, +virtualdev ... 41
-w<version> ... 119
-warp, +warp ... 27
-xsync, +xsync ... 29

A

ACIA (Swiftlink, Turbo232) ... 64
Audio buffer size ... 31

C

Converting X64 files into D64 ... 12

D

DigiMAX ... 64
Double-scan mode ... 28
Double-size mode ... 28

E

Ethernet (The Final Ethernet, RR-Net) ... 64

G

GEO-RAM ... 64

H

HP-UX and Solaris audio problems ... 13

L

Limiting emulation speed ... 27
Loosing control on low-end systems ... 28

M

MIDI (Passport, Datel, Maplin, Namesoft, Sequential) ... 64
MITSHM ... 13

O

OSS/Linux problems ... 13
Oversampling ... 31

R

Refresh rate ... 27
reSID resampling passband ... 63
reSID samping method ... 62
REU ... 64

S

Sample rate ... 31
Second SID ... 62
Second SID base address ... 62
SFX Sound Expander ... 64
SFX Sound Sampler ... 64
SID filters ... 62
SID models ... 62
Sound buffer size ... 31
Sound speed adjustment ... 31
Sound suspend time ... 31
Sound syncronization ... 31
Sprite collision detection ... 59

T

Toggling reSID emulation 62
Turning sound playback on/off 31

U

Using XSync() 28

V

VIC-II color sets 59
Video cache 28

W

Warp speed mode 27

Index of Resources

A

Acia1Dev 43
Acia1Enable 64
Acia1Irq 43

B

Basic1 80
Basic1Chars 80
BasicName 67, 77

C

Cart2Name 85
Cart4Name 85
Cart6Name 85
CartridgeFile 51
CartridgeReset 51
CartridgeType 51
ChargenName 67, 77, 81
Crtc 82
CrtcDoubleScan 82
CrtcDoubleSize 82
CrtcPaletteFile 82
CrtcVideoCache 82

D

DiagPin 81
DIGIMAX 64
DIGIMAXbase 64
Directory 45
DisplayDepth 28
DoCoreDump 46
DosName1541 35
DosName1571 35
DosName1581 35
DosName2000 35
DosName2031 35
DosName4000 35
DQBB 51
DQBBfilename 51
DQBBImageWrite 51
Drive8ExtendImagePolicy 35
Drive8IdleMethod 35
Drive8ParallelCable 35
Drive8Type 35
Drive9ExtendImagePolicy 35
Drive9IdleMethod 35
Drive9ParallelCable 35
Drive9Type 35
DriveTrueEmulation 35

E

EditorName 81
EoiBlank 81
ETHERNET_ACTIVE 64
ETHERNET_AS_RR 64
ETHERNET_DISABLED 64
ETHERNET_INTERFACE 64
ExpertCartridgeEnabled 51
ExpertCartridgeMode 51
Expertfilename 51
ExpertImageWrite 51

F

FSDevice10ConvertP00 38
FSDevice10Dir 38
FSDevice10HideCBMFiles 38
FSDevice10SaveP00 38
FSDevice11ConvertP00 38
FSDevice11Dir 38
FSDevice11HideCBMFiles 38
FSDevice11SaveP00 38
FSDevice8ConvertP00 38
FSDevice8Dir 38
FSDevice8HideCBMFiles 38
FSDevice8SaveP00 38
FSDevice9ConvertP00 38
FSDevice9Dir 38
FSDevice9HideCBMFiles 38
FSDevice9SaveP00 38

G

GenericCartridgeFile2000 77
GenericCartridgeFile4000 77
GenericCartridgeFile6000 77
GenericCartridgeFileA000 77
GenericCartridgeFileB000 77
GEORAM 64
GEORAMfilename 64
GEORAMImageWrite 65
GEORAMsize 65

H

HTMLBrowserCommand 46

I

IDE64AutodetectSize1 52
IDE64AutodetectSize2 52
IDE64AutodetectSize3 52
IDE64AutodetectSize4 52
IDE64Config 51
IDE64Cylinders1 52
IDE64Cylinders2 52
IDE64Cylinders3 52
IDE64Cylinders4 52
IDE64Heads1 52
IDE64Heads2 52
IDE64Heads3 52
IDE64Heads4 52
IDE64Image1 52
IDE64Image2 52
IDE64Image3 52
IDE64Image4 52
IDE64RTCOffset 52
IDE64Sectors1 52
IDE64Sectors2 52
IDE64Sectors3 52
IDE64Sectors4 52
IDE64version4 51
IEEE488 52
IEEE488Image 52
IOSize 80
IsepicCartridgeEnabled 52
Isepicfilename 52
IsepicImageWrite 52
IsepicSwitch 52

J

JoyDevice1 46
JoyDevice2 46

K

KernalName 67, 77, 81
KernalRev 67
KeymapBusinessDEPosFile 30
KeymapBusinessDESymFile 30
KeymapBusinessUKPosFile 29
KeymapBusinessUKSymFile 29
KeymapGraphicsPosFile 29
KeymapGraphicsSymFile 29
KeymapIndex 29
KeymapPosFile 29
KeymapSymFile 29

M

MagicVoiceCartridgeEnabled 52
MagicVoiceImage 52
MIDIEnable 65
MIDIMode 65
MITSHM 28
MMC64 53
MMC64_bios_write 53
MMC64_flashjumper 53
MMC64_revision 53
MMC64_RO 53
MMC64_sd_type 53
MMC64BIOSfilename 53
MMC64imagefilename 53
MMCRCardImage 53
MMCRCardRW 53
MMCREEPROMImage 53
MMCREEPROMRW 53
MMCRImageWrite 53
MMCRRescueMode 53
MMCRSDType 53
ModelLine 85

P

Printer4 40
Printer4Driver 40
Printer4Output 40
Printer4TextDevice 40
Printer5 40
Printer5Driver 40
Printer5Output 40
Printer5TextDevice 40
PrinterTextDevice1 40
PrinterTextDevice2 40
PrinterTextDevice3 40
PrinterUserport 40
PrinterUserportDriver 40
PrinterUserportOutput 40
PrinterUserportTextDevice 40
PrivateColormap 28

R

Ram08 85
Ram1 85
Ram2 85
Ram4 85

Ram6 85
Ram9 80
RamA 80
RAMBlock0 76
RAMBlock1 76
RAMBlock2 76
RAMBlock3 76
RAMBlock5 76
RamC 85
RAMCART 53
RAMCART_RO 53
RAMCARTfilename 53
RAMCARTImageWrite 53
RAMCARTsize 53
RamSize 80, 85
RefreshRate 27
REU 65
REUfilename 65
REUImageWrite 65
REUsize 65
RomModule9Name 81
RomModuleAName 81
RomModuleBName 81
RRBankJumper 53
RRBiosWrite 53
RRFlashJumper 53
RsDevice1 43
RsDevice1Baud 43
RsDevice2 43
RsDevice2Baud 43
RsDevice3 43
RsDevice3Baud 43
RsDevice4 43
RsDevice4Baud 43
RsUserBaud 43
RsUserDev 43
RsUserEnable 43

S

SaveResourcesOnExit 46
SFXSoundExpander 65
SFXSoundExpanderChip 65
SFXSoundSampler 65
SidEngine 63
SidFilters 63
SidModel 63
SidResidPassband 63
SidResidSampling 63
SidStereo 63
SidStereoAddressStart 63
Sound 31
SoundBufferSize 32
SoundDeviceArg 32
SoundDeviceName 32
SoundSampleRate 32
SoundSpeedAdjustment 31
SoundSuspendTime 32
Speed 27
SuperPET 80

U

UseVicII 85
UseXSync 28

V

VICDoubleScan 74
VICDoubleSize 74
VICIICheckSbColl 60
VICIICheckSsColl 60
VICIIDoubleScan 60
VICIIDoubleSize 60
VICIIPaletteFile 60
VICIIVideoCache 60
VICPaletteFile 74
VICVideoCache 74
VideoSize 80
VirtualDevices 41

W

WarpMode 27

Table of Contents

1 GNU GENERAL PUBLIC LICENSE 1

Preamble 1
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION 2
How to Apply These Terms to Your New Programs 6

2 About VICE 7

2.1 C64 emulator features 7
2.2 C64DTV emulator features 7
2.3 C128 emulator features 8
2.4 VIC20 emulator features 8
2.5 PET emulator features 8
2.6 CBM-II emulator features 9
2.7 The keyboard emulation 9
2.8 The joystick emulation 10
2.9 The disk drive emulation 10
2.10 Supported file formats 12
2.11 Common problems 13
2.11.1 Sound problems 13
2.11.2 Shared memory problems 13
2.11.3 Printer problems 13
2.11.4 PET keyboard problems 14

3 Invoking the emulators 14

3.1 Command-line options used during initialization 14
3.2 Autostarting programs from the command-line 16

4 System files 16

4.1 ROM files 17
4.2 Keymap files 19
4.3 Palette files 20
4.4 Romset files 21

5 Basic operation 21

5.1 The emulation window 21
5.2 Using the menus 22
5.3 Getting help 22
5.4 Using the file selector 22
5.5 Using disk and tape images 23
5.5.1 Previewing the image contents 23
5.5.2 “Autostarting” an image 24
5.5.3 Using compressed files 24

5.5.4 Using Zipcode and Lynx images 24
5.6 Resetting the machine 25

6 Settings and resources 25

6.1 Format of resource files 26
6.2 Using command-line options to change resources 26
6.3 Performance settings 27
6.3.1 Performance resources 27
6.3.2 Performance command-line options 27
6.4 Video settings 28
6.4.1 Video resources 28
6.4.2 Video command-line options 28
6.5 Keyboard settings 29
6.5.1 Keyboard resources 29
6.5.2 Keyboard command-line options 30
6.6 Joystick settings 30
6.6.1 Joystick command-line options 30
6.7 Sound settings 31
6.7.1 Sound resources 31
6.7.2 Sound command-line options 33
6.8 Tape settings 33
6.8.1 Tape command-line options 33
6.9 Drive settings 33
6.9.1 Drive resources 35
6.9.2 Drive command-line options 35
6.10 Peripheral settings 37
6.10.1 Settings for file system devices 37
6.10.1.1 Resources for file system devices 38
6.10.1.2 Command-line options for file system devices 39
6.10.2 Printer settings 39
6.10.2.1 Printer resources 40
6.10.2.2 Printer command-line options 40
6.10.3 Disabling kernal traps 41
6.10.3.1 Resources to control Kernal traps 41
6.10.3.2 Command-line options to control Kernal traps 41
6.11 RS232 settings 42
6.11.1 RS232 resources 42
6.11.2 RS232 command-line options 43
6.11.3 RS232 usage example 44
6.12 Monitor settings 44
6.12.1 Monitor command-line options 44
6.13 Machine settings 45
6.13.1 Machine command-line options 45
6.14 Memory settings 45
6.14.1 Memory command-line options 45
6.15 Miscellaneous settings 45
6.15.1 Miscellaneous resources 45
6.15.2 Miscellaneous command-line options 46

7 Machine-specific features 47

7.1 C64/128-specific commands and settings 47
7.1.1 Using cartridges 47
7.1.1.1 Slot 0 48
7.1.1.2 Slot 1 48
7.1.1.3 Main Slot 48
7.1.1.4 I/O Slot 50
7.1.1.5 Expected behaviour 50
7.1.1.6 Common problems 50
7.1.1.7 IEEE-488 interface 51
7.1.1.8 The Final Cartridge 3 51
7.1.2 C64 cartridge settings 51
7.1.2.1 C64 cartridge resources 51
7.1.2.2 C64 cartridge command-line options 53
7.1.3 CIA settings 59
7.1.3.1 CIA command-line options 59
7.1.4 VIC-II settings 59
7.1.4.1 VIC-II resources 60
7.1.4.2 VIC-II command-line options 60
7.1.5 SID settings 62
7.1.5.1 SID resources 63
7.1.5.2 SID command-line options 63
7.1.6 C64 I/O extension settings 64
7.1.6.1 C64 I/O extension resources 64
7.1.6.2 C64 I/O extension command-line options 65
7.1.7 C64/128 system ROM settings 66
7.1.7.1 C64/128 system ROM resources 67
7.1.7.2 C64/128 system ROM command-line options 67
7.1.8 C64 settings 67
7.1.8.1 C64 command-line options 67
7.2 C128-specific commands and settings 68
7.2.1 VDC settings 68
7.2.1.1 VDC command-line options 68
7.2.2 C128 system ROM settings 69
7.2.2.1 C128 system ROM command-line options 69
7.2.3 C128 settings 70
7.2.3.1 C128 command-line options 70
7.3 C64DTV-specific commands and settings 70
7.3.1 C64DTV ROM image 71
7.3.2 DTV revision 71
7.3.3 LumaFix 71
7.3.4 Userport 72
7.3.5 Debug 72
7.3.6 Monitor DTV features 72
7.4 VIC20-specific commands and settings 72
7.4.1 Using cartridge images 73
7.4.2 VIC20 cartridge settings 73
7.4.2.1 VIC20 cartridge command-line options 73

7.4.3 VIC settings . . . 74
7.4.3.1 VIC resources . . . 74
7.4.3.2 VIC command-line options . . . 74
7.4.4 Changing memory configuration . . . 76
7.4.4.1 VIC20 memory configuration resources . . . 76
7.4.4.2 VIC20 memory configuration command-line options . . . 76
7.4.5 VIC20 system ROM settings . . . 77
7.4.5.1 VIC20 system ROM resources . . . 77
7.4.5.2 VIC20 system ROM command-line options . . . 77
7.4.6 VIC20 settings . . . 78
7.4.6.1 VIC20 command-line options . . . 78
7.5 PLUS4-specific commands and settings . . . 78
7.5.1 TED settings . . . 78
7.5.1.1 TED command-line options . . . 78
7.5.2 PLUS4 I/O extension settings . . . 79
7.5.2.1 PLUS4 I/O extension command-line options . . . 79
7.5.3 PLUS4 system ROM settings . . . 79
7.5.3.1 PLUS4 system ROM command-line options . . . 79
7.5.4 PLUS4 settings . . . 80
7.5.4.1 PLUS4 command-line options . . . 80
7.6 PET-specific commands and settings . . . 80
7.6.1 Changing PET model settings . . . 80
7.6.2 CRTC Settings . . . 82
7.6.2.1 CRTC resources . . . 82
7.6.2.2 CRTC command-line options . . . 82
7.6.3 The PET diagnostic pin . . . 83
7.6.4 PET command line options . . . 83
7.6.5 Changing screen colors . . . 84
7.7 CBM-II-specific commands and settings . . . 85
7.7.1 Changing CBM-II model . . . 85
7.7.2 CBM-II command line options . . . 86
7.7.3 Changing screen colors . . . 87
7.8 VSID-specific commands and settings . . . 87
7.8.1 VSID settings . . . 87
7.8.1.1 VSID command-line options . . . 87

8 Snapshots . . . 87

8.1 Snapshot usage . . . 87
8.2 Snapshot format . . . 88
8.2.1 Emulator modules . . . 88
8.2.1.1 x64 modules . . . 88
8.2.1.2 x128 modules . . . 88
8.2.1.3 xvic modules . . . 89
8.2.1.4 xpet modules . . . 89
8.2.1.5 xcbm2 modules . . . 90
8.2.1.6 Drive modules . . . 90
8.2.2 Module formats . . . 90
8.2.2.1 Terminology . . . 91

8.2.2.2 Module framework ... 91
8.2.2.3 CPU module ... 92
8.2.2.4 CIA module ... 92
8.2.2.5 VIA module ... 93
8.2.2.6 PIA module ... 94
8.2.2.7 TPI module ... 94
8.2.2.8 RIOT module ... 95
8.2.2.9 SID module ... 95
8.2.2.10 ACIA module ... 95
8.2.2.11 VIC-I module ... 95
8.2.2.12 VIC-II module ... 96
8.2.2.13 CRTC module ... 96
8.2.2.14 C64 memory module ... 97
8.2.2.15 C128 memory module ... 98
8.2.2.16 VIC20 memory module ... 98
8.2.2.17 PET memory module ... 99
8.2.2.18 CBM-II memory module ... 101
8.2.2.19 C500 data module ... 102

9 Media images ... 102

9.1 Media images command-line options ... 103

10 Event history ... 103

10.1 Event history command-line options ... 103

11 Monitor ... 103

11.1 Terminology ... 103
11.2 Machine state commands ... 105
11.3 Memory commands ... 106
11.4 Assembly commands ... 107
11.5 Checkpoint commands ... 107
11.6 General commands ... 109
11.7 Disk commands ... 109
11.8 Command file commands ... 110
11.9 Label commands ... 111
11.10 Miscellaneous commands ... 111

12 c1541 ... 112

12.1 Specifying files in c1541 ... 113
12.2 Using quotes and backslashes ... 113
12.3 c1541 commands and options ... 113
12.4 Executing shell commands ... 115
12.5 c1541 examples ... 115

13 cartconv ... 115

13.1 cartconv command line options ... 115
13.2 cartconv examples ... 118

14 petcat 118

14.1 petcat command line options 118
14.2 petcat examples 120

15 The emulator file formats 120

15.1 The T64 tape image format 120
15.1.1 T64 File structure 120
15.1.2 Tape Record 121
15.1.3 File record 121
15.2 The G64 GCR-encoded disk image format 121
15.3 The D64 disk image format 126
15.4 The X64 disk image format 135
15.5 The D71 disk image format 136
15.6 The D81 disk image format 143
15.7 The D80 disk image format 152
15.8 The D82 disk image format 159
15.9 The P00 image format 168

16 Acknowledgments 169

17 Copyright 175

18 Contact information 176

18.1 VICE home page 176
18.2 How to send feedback 176
18.3 How to contribute 177
18.4 Interesting newsgroups 178
18.5 FAQs you should read 178

Concept Index 178

Index of Resources 184