SIDin 6

"Holly Mary Combs"

Vice snapshot with CCS64 palette

Made with the GIMP from a HMC photo
and converted to C64 320x200
HiRes Mode Bitmap
by Stefano Tognon
in 2003

"4th voice"
...

Ice Team Free Software Group

SIDin 6
version 1.00
8 May 2004

General Index

Editorials

Stefano Tognon <ice00@libero.it>

Hi, again.

In this number we go to see the new Asterion Sid Tracker music editor program, by looking to his characteristics.

Instead the planned rip of Marble Madness is delayed to the next number, as due to my activity of this period, I had not give all the times the rip needs. Many progress was made into understand the engine and many part reveals that it is not used at his maximum power.

So, as I had other articles ready, for maintaining the scheduled date of publication, this number is out, because until September I will be full-time involving in minigame competition.

You can so look at the internal of DigiOrganizer, a great editor for 4bit sample music that can be used with you preferred music editor too.

Finally, version 0.3 of HVMEC (High Voltage Music Engines Collection, see previous issue of SIDin) is in high working stage and it will be released soon.

Now there is the Other categories that contains all other programs that don't fit in the Tracker/Editor as they manage only sample or sound effects.

Bye
S.T.

News

Some various news of players, programs , competition:

- Goattracker 1.5/1.51/1.52
- Ninjatracker 1.1
- PocketSid Player
- ACID 64 Player 2.0
- Asterion Sid-Tracker V1.0
- Audio::SID v3.03
- SIDedit v4.00/v401
- Sidplay2/w 20-02-2004
- HVSC update 38
- SidWine3 Compo

Goattracker 1.5/1.51/1.52

Released on 25 January 2004 the new version of Cadaver's PC tracker with lot of changes:

- Playroutine rewritten. Uses testbit hard restart from now on for much sharper sounds
- ADSR parameter for hard restart configurable (previously only AD)
- Added orderlist cut/copy/paste
- Added indicators for ADSR parameter & hard restart length to the title bar
- Added delayed wavetable execution
- Added proper gateoff, so keyoff in the patterns works correctly even if your wavetable has gatebit set to 1
- Added master fader command (7F0-7FF)
- Timing mark is now command 7EF
- Standard playroutine saves all zeropage locations it uses
- INS2SND modified, now it will no longer insert delays, as these are unsupported by the new gamemusic playroutine

On 16 February, version 1.51 was released with these changes:

- Funktempo hack works differently
- Copybuffer no longer erased on song load

In April,version 1.52 was released with these changes:

- Frequency table should have better tuning now
- Relocation address no longer significant for gamemusic tunes

Download the standard and stereo version at:

http://www.student.oulu.fi/~loorni/covert/tools/goattrk.zip
http://www.student.oulu.fi/~loorni/covert/tools/gstereo.zip

Ninjatracker 1.1

Released on 25 January 2004 the new version of Cadaver's C64 tracker with some changes:

- Hardrestart is more solid (set both AD,SR to $00) and same in both standard & gamemusic versions
- INS in pulse & filtertable inserts a 00,00-row, instead of 90,00 as in wavetable
- Pulse/filter-pointers in wavetable are adjusted when INS/DEL is used in pulse- or filtertable
- Movement in patterns speed-optimized (no unnecessary workpattern->pattern conversion anymore)

Download the program at:
http://www.student.oulu.fi/~loorni/covert/tools/ninjatrk.zip

PocketSid player

PocketSID is a music player for Windows Mobile 2003 devices that plays Commodore 64 SID music files. PocketSID is a Pocket PC front-end on top of a port of the sidplay2 / reSID emulation engine.
Download PocketSID:

PocketSID 0.47 for Windows Mobile 2003 (ARM):
http://pocketsid.progenitus.com/PocketSID.PPC2003_ARM.CAB

PocketSID 0.47 for Windows Mobile 2003 (Emulator):
http://pocketsid.progenitus.com/PocketSID.PPC2003_x86.CAB

On 30/03/2004 version 0.64 was released.
This version fixes a number of minor problems and supports song skipping and pausing via your PDA's buttons:
http://pocketsid.progenitus.com/PocketSID-0.64.zip

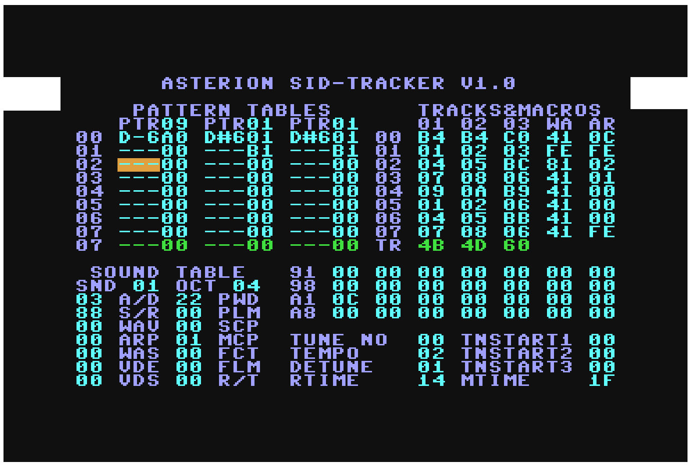
ACID 64 Player v2.0

ACID 64 is a sid player for windows and now it is at version 2.0 (released 5 May) that has this new features:

- Cycle based emulation of 6510 CPU, 6526 CIA and 6569 VIC-II chip
- Cycle based playback of SID data
- Support for sample playback of RSID and C64 Program files
- Bad line support for character and sprite data reading
- Support for playing PRG and PC64 files
- Simulation of space bar and joystick fire buttons to skip intros
- Support for restoring screen size
- Added 6510 CPU performance indicator
- Support for skipping silents of beginning of tunes
- Better resetting of SID chip
- Faster voice bar animation
- Fix for frequency adjustment to avoid warble sounds in some tunes
- Many other improvements

Download: http://www.xs4all.nl/~boswme/download/acid64_player_v20.zip

Asterion Sid-Tracker V1.0



```
         ASTERION SID-TRACKER V1.0

       PATTERN TABLES        TRACKS&MACROS
      PTR09 PTR01 PTR01      01 02 03 WA AR
   00 D-6A0 D#601 D#601 00 B4 B4 C0 41 0C
   01 ---00 ---B1 ---B1 01 01 02 03 FE FE
   02 ---00 ---00 ---00 02 04 05 BC 81 02
   03 ---00 ---00 ---00 03 07 08 06 41 01
   04 ---00 ---00 ---00 04 09 0A B9 41 00
   05 ---00 ---00 ---00 05 01 02 06 41 00
   06 ---00 ---00 ---00 06 04 05 BB 41 00
   07 ---00 ---00 ---00 07 07 08 06 41 FE
   07 ---00 ---00 ---00 TR 4B 4D 60

    SOUND TABLE   91 00 00 00 00 00 00 00
   SND 01 OCT 04  98 00 00 00 00 00 00 00
   03 A/D 22 PWD  A1 0C 00 00 00 00 00 00
   88 S/R 00 PLM  A8 00 00 00 00 00 00 00
   00 WAV 00 SCP
   00 ARP 01 MCP  TUNE NO   00 TNSTART1 00
   00 WAS 00 FCT  TEMPO     02 TNSTART2 00
   00 VDE 00 FLM  DETUNE    01 TNSTART3 00
   00 VDS 00 R/T  RTIME     14 MTIME    1F
```

Released on 26 January 2004 a new Sid-Tracker from Asterion of Tinnitus.

The editor is tracked-based and come out with lot of sample musics, from Asterion, Borgon and Trompkins.

Download the disk image from
http://tinnitus.prv.pl

Audio::SID v3.03

Audio::SID Perl module by Lala. In this new version:

- Added a check in validate() to make sure the initAddress is within the load range for PSIDs.
- Added complete support for the new 'C64 BASIC' RSID flag: added the new getC64BASIC(), isC64BASIC(), setC64BASIC() methods, changed the PlaySID flag related methods. See documentation for details.

SIDedit v4.00/v4.01/v4.02

The new version of SIDedit perl ripper program was released on 02/02/2004 by LaLa. Some of the major changes in this version are:

- Added support for the new 'C64 BASIC' RSID flag. (Thanks to Simon White!) Click on "Edit flags" to set or clear this flag.

- Added rudimentary support to modify a contiguous string of bytes in the SID data itself. See "Display SID data" -> "Modify bytes". Consult the "Modifying Bytes in SID Data" section of the "Help" -> "Quick tutorial" guide for details on its usage. (WARNING: You can really screw up a SID file with this, so use it with extreme care!)

- Major overhaul of the user interface:

    - SIDedit now uses menus and toolbars for easy access to most functions.

    - The main window and the data display make use of colors.

    - In the data display window it is now possible to click on the jump/branch addresses to jump to that location directly (if that location can be displayed in the window).

    - In the data display window switching between hex dump and assembly listings retains the currently displayed address range in the window.

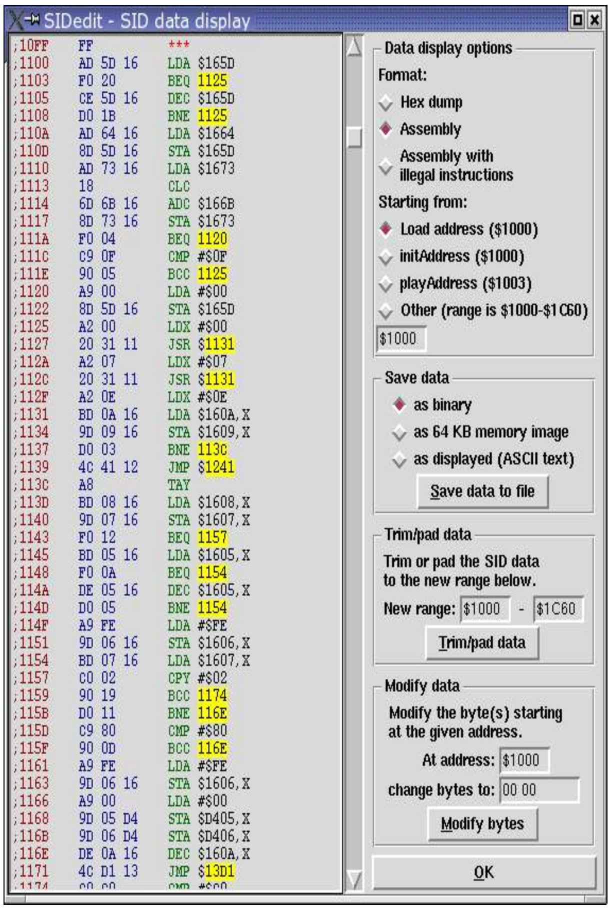
    - All the settings are now configurable via a separate "Configure settings" popup window.

  - Experimental mousewheel support. (NOTE: on some systems if you press down on the mousewheel button and move down, the window contents scroll up, and moving up scrolls them down instead of the other way around. This seems to be a bug in Perl/Tk itself.)

  - The file navigator window pane also got its own toolbar with 3 directory "shortcuts" (default dir, default save dir, HVSC dir) which are configurable via the "Configure settings" popup window. You can use these shortcuts to jump to those dirs without navigating to them.

- Lots and lots of small bugfixes. Hopefully, I managed to fix all the bugs that were reported since the previous version was released. :)

Download from http://lala.c64.org



```
;10FF    FF          ***
;1100    AD 5D 16    LDA $165D
;1103    F0 20       BEQ 1125
;1105    CE 5D 16    DEC $165D
;1108    D0 1B       BNE 1125
;110A    AD 64 16    LDA $1664
;110D    8D 5D 16    STA $165D
;1110    AD 73 16    LDA $1673
;1113    18          CLC
;1114    6D 6B 16    ADC $166B
;1117    8D 73 16    STA $1673
;111A    F0 04       BEQ 1120
;111C    C9 0F       CMP #$0F
;111E    90 05       BCC 1125
;1120    A9 00       LDA #$00
;1122    8D 5D 16    STA $165D
;1125    A2 00       LDX #$00
;1127    20 31 11    JSR $1131
;112A    A2 07       LDX #$07
;112C    20 31 11    JSR $1131
;112F    A2 0E       LDX #$0E
;1131    BD 0A 16    LDA $160A,X
;1134    9D 09 16    STA $1609,X
;1137    D0 03       BNE 113C
;1139    4C 41 12    JMP $1241
;113C    A8          TAY
;113D    BD 08 16    LDA $1608,X
;1140    9D 07 16    STA $1607,X
;1143    F0 12       BEQ 1157
;1145    BD 05 16    LDA $1605,X
;1148    F0 0A       BEQ 1154
;114A    DE 05 16    DEC $1605,X
;114D    D0 05       BNE 1154
;114F    A9 FE       LDA #$FE
;1151    9D 06 16    STA $1606,X
;1154    BD 07 16    LDA $1607,X
;1157    C0 02       CPY #$02
;1159    90 19       BCC 1174
;115B    D0 11       BNE 116E
;115D    C9 80       CMP #$80
;115F    90 0D       BCC 116E
;1161    A9 FE       LDA #$FE
;1163    9D 06 16    STA $1606,X
;1166    A9 00       LDA #$00
;1168    9D 05 D4    STA $D405,X
;116B    9D 06 D4    STA $D406,X
;116E    DE 0A 16    DEC $160A,X
;1171    4C D1 13    JMP $13D1
```

Data display options

Format: Hex dump [✓ Assembly] Assembly with illegal instructions

Starting from: [✓ Load address ($1000)] initAddress ($1000) playAddress ($1003) Other (range is $1000-$1C60)

$1000

Save data

[✓ as binary] as 64 KB memory image as displayed (ASCII text)

Trim/pad data

Trim or pad the SID data to the new range below.

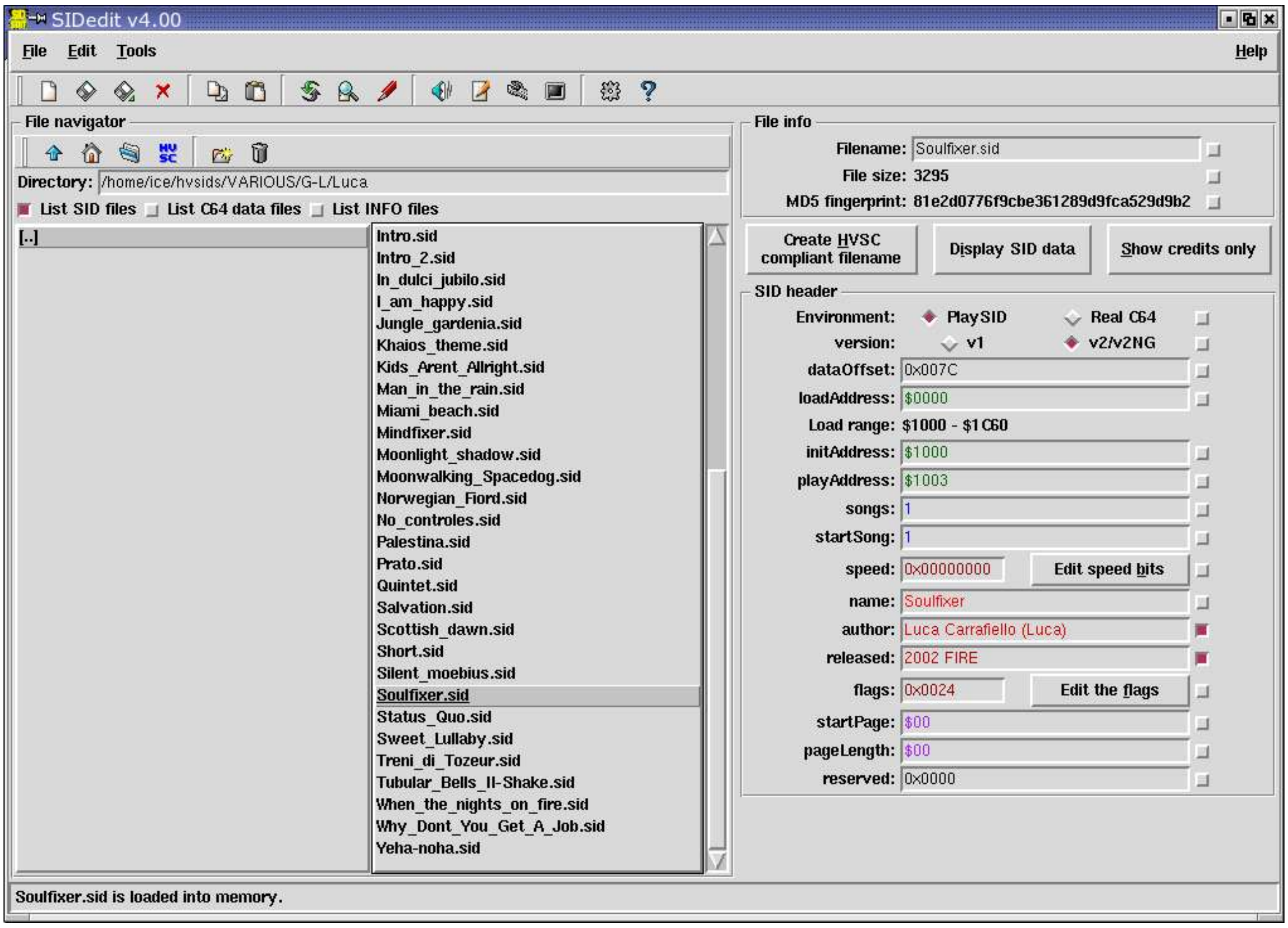
New range: $1000 - $1C60

Modify data

Modify the byte(s) starting at the given address.

At address: $1000

change bytes to: 00 00



Directory: /home/ice/hvsids/VARIOUS/G-L/Luca

[✓ List SID files] List C64 data files List INFO files

[..]

Intro.sid
Intro_2.sid
In_dulci_jubilo.sid
I_am_happy.sid
Jungle_gardenia.sid
Khaios_theme.sid
Kids_Arent_Allright.sid
Man_in_the_rain.sid
Miami_beach.sid
Mindfixer.sid
Moonlight_shadow.sid
Moonwalking_Spacedog.sid
Norwegian_Fiord.sid
No_controles.sid
Palestina.sid
Prato.sid
Quintet.sid
Salvation.sid
Scottish_dawn.sid
Short.sid
Silent_moebius.sid
Soulfixer.sid
Status_Quo.sid
Sweet_Lullaby.sid
Treni_di_Tozeur.sid
Tubular_Bells_II-Shake.sid
When_the_nights_on_fire.sid
Why_Dont_You_Get_A_Job.sid
Yeha-noha.sid

File info

Filename: Soulfixer.sid
File size: 3295
MD5 fingerprint: 81e2d0776f9cbe361289d9fca529d9b2

SID header

Environment: [✓ PlaySID] Real C64
version: v1 [✓ v2/v2NG]
dataOffset: 0x007C
loadAddress: $0000
Load range: $1000 - $1C60
initAddress: $1000
playAddress: $1003
songs: 1
startSong: 1
speed: 0x00000000
name: Soulfixer
author: Luca Carrafiello (Luca)
released: 2002 FIRE
flags: 0x0024
startPage: $00
pageLength: $00
reserved: 0x0000

Soulfixer.sid is loaded into memory.

Version 4.01 of the program follows some days after the previous release and contains:

- (added) New setting to turn off syntax coloring in data display to speed up display and clipboard operations.
- (added) Version history in the POD.
- (added) Broke up content of INI file into 4 sections within the file for better readability.
- (fixed) Colors were one off on the hex data display.
- (fixed) There was a problem entering the "Other" address in the data display. It caused focus problem, data rendering problem, etc. The address entry is now enabled permanently.

Finally version 4.02 contains:

- All illegal assembly instructions are now colored red in the data display when syntax coloring is turned on.
- A line consisting of dashes is displayed after every JMP and RTS instruction in the data display to improve readability.
- Added configuration setting "Automatically create HVSC compliant filename when 'name' field changes". Turning on this setting automatically changes the filename to be HVSC compliant whenever the 'name' field changes, and it also gets rid of the pop-up confirmation window when the "Create HVSC compliant filename" button is pressed.
- Added missing '$' signs when displaying conditional branch instruction addresses in the data display.

Sidplay2/w 20-02-2004

The new version of the Windows sid player by Adam Lorentzon was released with this changes:

- ReSID 0.15
- Improved CIA emulation in libsidplay2 (TOD emulation added)
- RSID support improved
- CPU debugging output file is now asked for each time

Download from http://www.gsldata.se/c64/spw/

HVSC update 38

Released on 21 February 2004 at http://www.hvsc.c64.org the update 38 of HVSC:

After this update, the collection should contain 25,149 SID files!

This update features (all approximates):
```
  1695 new SIDs
   126 fixed/better rips (129 SIDs)
     2 fix of PlaySID/Sidplay1 specific SIDs
     9 repeats/bad rips eliminated
   106 SID credit fixes
   280 SID model infos
    18 tunes moved out of /DEMOS to their composers' directories
     7 tunes from /DEMOS/UNKNOWN identified  :-)
    19 tunes moved out of /GAMES to their composers' directories
    (and 3 go back into /GAMES due to a wrongly credited tune)
```

This time we bring you:

- another dose of 1000+ new SIDs, in an effort to finally clean out our Unreleased archive. And it's still not empty yet... [sigh!]

- Like always we present all the latest scene releases, such as from demos and music collections that were released since christmas.
- The music compo tunes from several parties: Xmas Compo 2003, Deadline 2003, TUM 2003 and Out Of Orderia 2003 (hope I didn't forget anything important).
- You probably already noticed it, but it's worth stating again: The High Voltage SID Collection has reached the magical 25,000 mark! Who would have thought we would ever come this far? Only a couple of years ago we were thinking about if we would ever reach the 15,000 limit. How naive! :-)

Let's start the race towards the 30,000, the support of you, the SID music lovers, shall be the wind in our back. Please keep up sending tunes, notify us of missing ones, let us know about bugged rips, submit your STIL entries. And I can promise you that the SIDoine will continue flowing through your blood vessels! :-)

Main composers featured in this update:

```
M E R M A N
Stefan Hartwig                  Kochan_Maciej
Jörg Rosenstiel                 Scroll
Agemixer                        Merman
Amadeus/Attic                   Mr.Bungle
Amadeus/Meka Design             Ramos
John Ames                       Replay
Compod                          Signor
Arne                            Tips
Richard Bayliss                 Tjagvad
Klax                            Warnock
Stefan Weinert                  Welle: Erdball
Wodnik                          Peace
The Gee
```

SidWine3 compo

The SidWine 3 Online Music Competition was achieved from 21 February to 25 April at http://digilander.iol.it/ice00/tsid/sidwine3

Here the result:

Pos.	Points	Title	Author
1	174	Septic Shock	Daniel M. Gartke (Turtle)
2	152	Ole!	Hein Holt
3	141	One More Chance	Slowhand (Bekir Ogurlu)
4	134	Wolf	Hukka (Pietari Toivonen)
5	133	Sonic Dreams	Jaymz Julian (A Life in Hell)
6	129	Sea of the Seven Winds	Asterion
7	123	Distorted Frog	Tony Caven (Ferrara)
8	98	Enigma Variations	Anders Carlsson (Zapac)

Look at more user comments by downloading the result pack available in the site.

Jan Diabelez Arnt Harries Interview!

by Stefano Tognon

This time we can read this interview with Jan Diabelez Arnt Harries made in February of this year. You probably already know him as Rambones sid author and that he is a HVSC crew member from the beginning, so don't hesitate to read the interview.

Hello, Jan,
Why not start by giving us some information about you and your real life?

My full name is Jan Diabelez Arnt Harries, and I'm born on 12.01.1969 in Nyborg, Denmark.
I'm an educated and experienced IT-technician and programmer.
At the moment I'm looking for a new job, I just recently finished yet another education. I've been going to school 20.5 years out of my 35 :-)

I'm interesting to know what was working in a team in the early year of commodore story: you were a member of The Supply Team active from middle '80, so you can tell something about.

When I first met the other members of TST: TSN, Kaze, Wizz, Hagar - I only knew a little of BASIC. Me and TSN learned some BASIC during 1985, and then we met Kaze who was using assembler. Kaze worked for a whole year on his demo "Kaze Demo #2", and we decided to make the group The Supply Team.
Kaze met Wizz in school, and then we had 2 assembler programmers.
At this point, later half of 1986, we made the first TST demos that were widely spread. Later we got Hagar as a gfx artist, he was very good with Koala-painter. Things kept improving, and I started making my own demos, and later, my own music as well: using the Soundmonitor from Chris Hlsbeck.
These tunes can be found in our demos offcourse, and in the HVSC at
/VARIOUS/M-R/Rambones/*.sid
In 1987 Kaze and Wizz left for Wizax 2004, and later they formed Zetrex 2005.
We had some internal friendship trouble because me and TSN were still using Kaze's assembler routines after he left the team. This resulted in the Federation Against Supply Team (F.A.S.T). Ironically, I got more and more contacts because FAST made TST even more famous :-)
After 8 months of war, Zetrex broke up and Wizz joined 2000 A.D. - in the end we all decided to be friends again, and we co-operated on the demos "Mip Mip Police!" and "No System".
TST was never a cracking group, we were one of the first real demogroups in Denmark. Our heroes as we began were Sodan, 1001 Crew, The Judges and Dutch-USA Team.
In 1988 and 89, we all had gotten an Amiga 500/2000, and we started doing demos on it. After about a year we stopped because each member was into higher education - all of which were computer-related.
I was using an A500, later I got a harddisk for it, and last I used an A1200. Up to 1996. Then I shifted to PC, so did the other TST members.

You probably are one of the first ripper of C64 and Amiga music as you begin from '86 with your "Ripp(ed) Off" collections: can you speek about these old projects?

I started in 1986 by ripping music from TMC's "Game Music #1-9".
This was really easy, and it helped me to understand what to look for when wanting to rip a tune. I then started ripping from games, and just got better and better at it. In late 1988 I just resetted the game and hit F2 to get into my SpeedDOS opcode-monitor. (it shows the whole memory as opcodes and PETASCII)

I found the music by looking at the ASCII pattern that looked like notedata for the music, then quickly decided where the player was (usually in front of the data, and determining an end-address).
All this, without looking in the code (well only opcodes) - those were really golden days.
I also used a monitor called "Handic" which i had gotten on a cartridge. This cartridge very often fucked up the memory while I was ripping, so I often had to start all over again.
In 1987 I started spreading my rips, because there were too many to use in demos (I didn't know what to do with all the rips).
I invented the first rip-series called "Ripped-Off #1-2xx". There was about 200, from early 1987 to late 1988.
I later saw others try the same thing, but they all stopped before they reached 20 rips.
In 1986-1989 i filled 8 disk with rips. Later that number was gonna increase tenfold as I started converting tunes to PlaySID on the Amiga.
I only knew about one other ripper that was as much into it as I was, and that was Omega Supreme (Olav Mrkrid) from The Shadows (Norway).
I met him at the Danish Gold party in 1988 where TST released "New Limits", and it turned out he had ripped "Cauldron II", which I could not!
To this day I still haven't ripped it. Me and Olav had a sound competition going that made me try even harder.
By now I've ripped around 5000 tunes.

But you are even one of the first people involving in HVSC: can you tell about the born of the project, and how it grown until now?

In 1990 came the demo "The 100 Most Remembered C64 Tunes" for the Amiga, from Per H an Sundell and Ron Birk. It had a built in SID6581 emulator, and showed some koala pictures aswell. In 1991 they released PlaySID with about 300 tunes. A long time passed, and no more tunes were ported to the Amiga. One day I got home from school, my friends Wizz and Brian Nevad had 'stolen' my C64 and disks, and transferred 189 tunes to PlaySID 2.0b. The released this as "Addition", and it was the first pack from outside the PlaySID team with ripped sidtunes.
I got the cable they had been using (it was self-made), and started to transfer all my rips to PlaySID. From 1991-1995 I released 8 packs called "Ripp Off #1-8", and the first couple of years I was the only one doing it, then others started.
A guy called Nemesis1 from Island took all the collections of tunes there was released, and put them into one collection called the NemeSIDs.
At first I was angry, because he took my work, and presented it as his own, but after some time I only ripped more and more, to 'complete' the collection. As a result of building NemeSIDs, I got in contact with more and more rippers across the world, and it all grew - slowly.
In 1996 I got the PC, and as a result of my work I ended up in contact with the author of SIDPLAY for the PC, Michael Schwendt. We exchanged all the stuff we had, and he kept improving SIDPLAY.
After a while I got in contact with The Shark of INC (David Greiman), and he was just about to release the equivalent of NemeSIDs for the PC/SIDPLAY.
I added all my rips (at the time it was 1364), and we agreed that I was to be co-author of the High Voltage SID Collection (HVSC), together with BOD/Talent and Michael Schwendt.
Now we finally had a team and worked together on solving the mysteries of who really composed the various tunes. Something that NemeSIDs wasn't really good at. Shark contacted many composers by email, and we got a lot of tunes that had never been released before. Now the collection was really going somewhere!
The HVSC has grown into the mature project that it is today, only because we have gotten more and more feedback from the composers themselves, and new team members. Rippers, coders, credits and STIL experts.
With the creation of the HVSC update-tool, and the distribution as an archive that any platform can use - HVSC has been well accepted by the community.
In 1993 I estimated (based on my game/demo collection), that there was about 10000 tunes to rip. Today we have more then 25000!
This is due to people like those behind the Gamebase64 and others, who have helped preserve the

software, and because of the C64 emulaters that have been made (CCS64, VICE, Frodo, A64 etc.). All these projects combined have proven to be a good road for the preservation and future of the C64 scene in general.
This magazine you're reading right now, is a direct result of all these events.
Perhaps it was inevitable the way things got to as they are now, but without the work of all these people it certainly wouldn't have been as good as it is right now in 2004, 22 years after the release of the Commodore 64.

As you have ripped probably tons of music until now, you can give the right words for the readers that want to become a ripper. What is your advice for them?

If anyone wants to be a ripper, they either have a C64 or emulator with tons of software. So they should get a good machine-code monitor. I use Action Replay 6, but in the past I used less powerful methods. The easiest way to start is to first understand what music in a C64 is.
It is a portion of the running program, that writes data to the registers of the SID chip. The addresses go from $D400-D7FF, usually only the D400 area is used.
To make things even easier, take a simple demo with only 1 tune in it, and reset your C64. Then start the monitor, and look at the memory as PETASCII codes.
In Action Replay this is "I*0800-", it will display the memory from 0800 an upwards.
Find out what is the code (program), and learn to recognize it, apart from charset-data, sprite-data and screencodes (gfx, bitmaps).
Find out what the program is doing, and in particular, find out how many interrupts there are and what they do, what addresses they call (JSR $4000 is a call f.ex).
Writes to $0314/0315 and FFFE/FFFF are interrupts, and an interrupt usually ends with LDA #$01, STA $D019, JMP $EA31 or RTI.
Then look at the addresses called in the interrupt, and some of these subroutines will contain code that says STA $D400,X or y, and STA $D418/D412. If you find code like this, it is most probably the musicplayer.
Find out EXACTLY how the player is called (it all says so right there in the main program)
And find out what is done before the interrupt is made, this is most likely the initialization of the music routines. A music routine often needs to reset the starting point of the music to a zero.
After a while you will get accustomed to how a player looks, and understand what looks like and init-address, and what looks like a play-address.
By reading the player code, and looking at the data in memory, you can determine where it starts and ends. Save out the part(s) needed, and erase the whole memory.
Now load the part(s) saved, and write a small interrupt setup to make the music play.
You will now find out if your rip works or not.
Here's the setup I always use:

```
SEI                   ;prepare to set up interrupt
LDA #$35              ;switch off the KERNAL ROM (can now JSR music under E000-FFFF)
STA $01
LDA #>.interrupt      ;hi-byte of .interrupt address
STA $FFFF             ;hi-byte system interrupt vector used when KERNAL ROM is OFF)
LDA #<.interrupt      ;lo-byte of .interrupt address
STA $FFFE             ;lo-byte system interrupt vector used when KERNAL ROM is OFF)
LDX #$00
STX $DC0E             ;stop CIA timer, we want to use a PAL raster-interrupt here
INX
STX $D01A             ;enable interrupt
LDA #$1B
STA $D011             ;make this interrupt a raster-interrupt
LDA #$33
STA $D012             ;set rasterline to a specific line on the screen

LDA #$00              ;maybe the musicinit needs a number, maybe not, maybe in X or Y reg instead..
JSR _music_init       ;initialize the musicplayer
CLI
.lock JMP .lock       ;go into an endless loop, never return to basic!

.interrupt
LDA #$06              ;set a color on the screen-border, as long as the music is playing
STA $D020
JSR _music_play       ;play the music!
LDA #$00
STA $D020             ;set another color on the screen-border, so you can see how much rastertime the
```

```
                        ;music uses!
STA $D021
SEC
ROL $D019               ;the interrupt has occurred, and ends here
RTI                     ;return from interrupt
```

There are more simple methods to use when using an interrupt to play music, but this one gives you full control of the machine.

After getting the music to play, and making sure you have all the parts needed for it, you should try and put it together in one single file.
This is a whole different science, that I would like to explain in a future article.
After many years, I have perfected the technique needed to do this, and in many cases (especially old games) it is sometimes really difficult to do.

About ripping: if at first you don't succeed, try and try again!
For quick help, transfer a tune from the HVSC to your C64/emulator, and look how a musicplayer looks like - this will help you a lot.

From time to time you compose again with the Sid: if I see correctly, some of your last works are done with JCH editor. What do you think about the development that the most recent editor had have respect the initial ones? The SID chip is still the same, but what about the editors?

The editors have definitely improved over the years.
If you are into using real notation however, the possibilities are limited.
The only editors using notation are aged (1982-1986), and don't offer sophisticated sound creation.
So to get the best result, you must use the newest editors, trackers.
JCH and SDI (SID Duzz It, from SHAPE) are the best ones I've seen, and I now only use SDI.
A very good tracker for beginners is the Cybertracker from NoName, because it offers sound-creation in a very readable way. It also uses gfx to show ADSR, filters etc.

Now some quick final (standard) questions:
Real machine vs emulator: what do you think of?

For playing games and watching demos, I prefer the real machine.
It also has a nostalgic value to know that your C64 can still kick ass!

For listening music, off course the real machine.

For ripping and development I prefer the emulators, because it's much easier to transfer code from the assembler.
When ripping in the emulator you can by a single keystroke enter a monitor where you can see where the current interrupt is located, and thus very quickly see what is going on, while the program is running.

6581 vs 8580 chip: any (musical) preference?

6581 - not because it's the best (coz actually it isn't, technically), but because it's the chip that my C64 came with, and I've gotten used to the sound of it.

What is the worst sid that you compose and the better one?

Worst one: /VARIOUS/M-R/Rambones/Brev_til_dig.sid
Best one : /VARIOUS/M-R/Rambones/Zound_Muzak_3.sid

Who are your best sid authors?

This is the most asked question ever! :-)
Rob Hubbard, Martin Galway, Ben Daglish, David Whittaker, Fred Gray,
Mark Cooksey, Jeroen Tel, Charles Deenen, Drax, JCH, Laxity, Banana/TEK.

What are the best sids ever in your opinion?

After all these years, they are all the best.
Can't live without them.

Finally, many thanks for the time you give for this interview, and now you can say any things you want that the people will read from you!

Keep the C64 spirit alive, by making what you do best!

Since 1988 when I got my Amiga, I have been composing MOD, XM, MP3 and I released most of it on my website www.janharries.com
Go download the lot and listen to it, it's a story through time!

I proved to be a much better composer on the PC/Amiga than on the C64!

This was due to the fact that the TRACKER was invented by Karsten Obarski in 1987.

The HVSC is not a finished project, and it is very likely to be around after the C64 has completely died. If you want to contribute with new rips or fixes, credits, don't be shy and write to us: hvsc@c64.org

If you need help learning to rip music, write me: jan.harries@get2net.dk
And if you have the most irresistible music that is not ripped, send it to me as well and I'm sure we can manage it.

There will be released another interview by me at www.c64hq.com in the near future. Be sure to read it!

You can get all the C64 demos from The Supply Team (1985-1989) on:
http://hjem.get2net.dk/nmioaon/TSTDEMOS.zip (1.3 MB D64 images)

The Amiga demos we did are at:
http://hjem.get2net.dk/nmioaon/TSTAMIGA.zip (1.3 MB ADF images)

signed: Jan Diabelez Arnt Harries (rambones/nmioaon)

Webography:

Digital composer - http://www.janharries.com
Weblog - http://janharries.blogspot.com
Photo album - http://www.xpphotoalbum.com/showgallery.php?ppuser=5765&cat=500

Asterion Sid Tracker v1.0

by Stefano Tognon <ice00@libero.it>

I would like to write this article about a new music editor released this year: Asterion Sid Tracker.
The tracker is coded by Asterion of Tinnitus and come with a disk containing the editor with instructions, the packer & relocator and lot of music examples.

I try to give you an overview of the editor and my personal opinions about it.

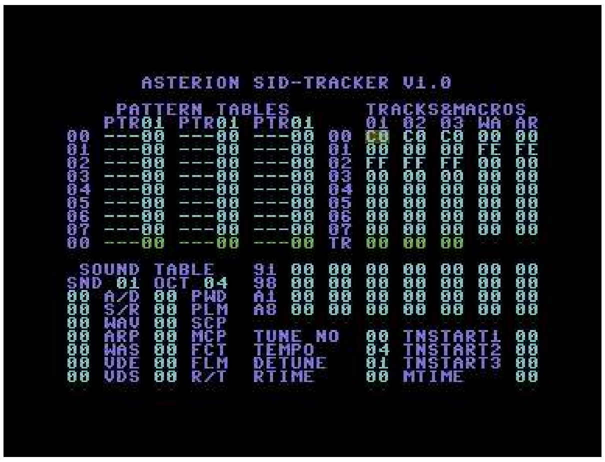
Intro

The editor starts up with a nice intro, that looks as a Halloween's one, like you can see in the left image. The music is very good and the scroller introduces the born of the editor and the Tinnitus group.

I like to see the intro in the editor, as it gives an old fashion to the production.

After you had pressed a key, the editor pops up with the main screen.

The editor



```
         ASTERION SID-TRACKER V1.0

     PATTERN TABLES        TRACKS&MACROS
    PTR01  PTR01  PTR01       01 02 03 WA AR
 00 ---00  ---00  ---00   00  C0 C0 C0 00 00
 01 ---00  ---00  ---00   01  00 00 00 FE FE
 02 ---00  ---00  ---00   02  FF FF FF 00 00
 03 ---00  ---00  ---00   03  00 00 00 00 00
 04 ---00  ---00  ---00   04  00 00 00 00 00
 05 ---00  ---00  ---00   05  00 00 00 00 00
 06 ---00  ---00  ---00   06  00 00 00 00 00
 07 ---00  ---00  ---00   07  00 00 00 00 00
 00 ---00  ---00  ---00   TR  00 00 00

  SOUND  TABLE    91 00 00 00 00 00 00 00
 SND 01 OCT 04    98 00 00 00 00 00 00 00
 00 A/D 00 PWD    A1 00 00 00 00 00 00 00
 00 S/R 00 PLM    A8 00 00 00 00 00 00 00
 00 WAV 00 SCP
 00 ARP 00 MCP    TUNE NO     00 INSTART1 00
 00 WAS 00 FCT    TEMPO       04 INSTART2 00
 00 VDE 00 FLM    DETUNE      01 INSTART3 00
 00 VDS 00 R/T    RTIME       00 MTIME    00
```

All you need for making music with the Asterion editor is in the main screen. This is probably a good features, as you don't have to open sub screens, but on the other side, this reduce the space for the pattern steps to 8.

Technically speaking, the editor is a tracker like one, similar for examples to Goattracker, Cybertracker, ecc., even if the commands you can use in each pattern steps are limited to one byte, as we see later.

Now I think that it is the case to start analyzing how you can make an instrument into this editor.

Sound table

An instrument is created with the sound table: press F3 to enter the *sound table* screen.

The parameters you can enter for controlling the sound generation are:

- A/D: Attack/Decay
- S/R: Sustain/Release
- WAV: Wave macrocommand

- ARP: Arpeggio macrocommand
- WAS: Wave/Arpeggio executing speed
- VDE: Vibrato delay
- VDS: Vibrato deeph/speed
- PWM: Pulse starting speed
- PLM: Pulse lower/upper limited
- SCP: Cuttoff/pulse speed
- MCP: Multicutoff/Multiple speed
- FCT: Filter starting cutoff
- FLM: Filter lower/upper limit
- R/T: Resonance/(1 2 4) filter type (8) Hardrestart on/off

The options are quite commons: ADSR of voice, Vibrato control, Pulse effect, advanced filter control, plus a wav/arpeggio table pointers and on/off control of hardrestart.

In fact, the values you inserted into WAV e ARP are a pointers into the table WAV and ARP that you can see into the *Track&Macros* screen.

WAV is the control shape of the voice (triangular, rectangular, ...), while APR is the value to add to the current note (for producing arpeggio, but even for the other effects) that can be relative (00-$7F values), absolute ($80-DF) or skydrive ($E0).

The use is so quite the commons:

WAV -> 30
ARP -> 32

POS	WAV	ARP
30	41	00
31	21	00
32	FE	FE

Performs a $41, followed by $21 with the same note frequency.

A $FE terminate the macro command, while a $FF will repeat the macro to the address specify by the next value:

WAV -> 20
ARP -> 20

POS	WAV	ARP
20	41	00
21	FE	0C
22	00	FF
23	00	20

This will perform an octave arpeggio to the note being played.

Note that Wave and Arpeggio tables are independent so you can achieve very complex tasks as even their speeds of execution (controlled by WAS) are separated.

Tracks

In the first 3 columns of *track&macros* screen, you have to insert the number of pattern that each voices may execute (from $01 to $7F).

Values from $80 to $FE will execute the transpose (C0-middle) of the pattern that follows.

Finally, the $FF values will loop the track specify by the followed number.

Example:

```
POS      01       02       03
-----------------------------------
00       03       04       02
01       03       8C       02
02       03       04       02
03       FF       04       FF
04       00       80       00
05                FF
06                00
```

Here 3 patterns for each voices are executed forever, but for the voice 2 the last two patterns are executed with a $0C of transposition.

Patterns

At the left upper part of the screen there is the pattern editor where you can inserted the notes and the effects to be played at each player call like in all tracker style program.

Each effect upon activated will be executed until a next one (so you don't have to repeat the command at each next positions).

Here there is the list of all available effects:

• 00	No effect
• 0x-1x	Instrument number
• 2x	1.Tie note 2.Synchronize 4.Modulation 8.No pulse restarting
• 3x	Portamento to target note with speed x
• 4x	Vibrato depth, with speed declared in vds 0.Vds depth
• 5x	Change attack
• 6x	Change decay
• 7x	Change sustain
• 8x	Change release (instrument declaration (0x-1x) restart adsr value)
• 9x	Execute waveform from index no.x 0.Normal waveform
• Ax	Execute arpeggio from index no.X 0.Normal arpeggio
• Bx	Set pulse 0. Instrument pulse LFO If MCP LSB=0 then absolute pulse (x dumped to pulse MSB)

If MCP LSB<>0 then x becomes 2Nd pulse value

- Cx Set cutoff
  0. Instrument cutoff LFO
  If MCP MSB=0 then absolute cutoff (x dumped to cutoff MSB)
  If MCP MSB<>0 then x becomes 2Nd cutoff value
- Dx set volume – default F
- Ex Change filtertype
  1. Lo pass
  2. Bd pas
  4 Hi pass
  8 No cuttoff restarting
- Fx Tempo
  0. Stop tune
  1-E Set tempo
  F End pattern – affects all tracks

The first point to observe is that the instrument to use is declared as an effect, while in other trackers you can specify it and even an effect to play. This is a choice that probably simplify the implementation of the tracker, as only one byte (over the note) is processed.

Other interesting commands to use are the $9x, $Ax that specify the position of Wave and Arpeggio to execute when the command is entered. You must use the middle screen (F6) for enter those values.

You can then add vibrato, portamento, set ADSR at the position you like, controlling the filters and the cuttoff and even controlling the volume for making fade-in/fade-out by using the other commands.

Finally you can control how to manage the pulse restarting, setting the tempo of the tune and you can let a pattern finish before by using $FF command.

Conclusion

As wee see, the editor has only one byte of commands in track, but it has all you need for making good music as instrument implementation is good (e.g. Wave/Arpeggio independent tables are very powerful).

The point that can be made better is to implement a more powerful disk menu, where you can see the directory listing and you can choose the music to load by navigate directly into the screen.

Actually, the disk operation are limited to SHIFT+L, SHIFT+S for loading and saving and F8 for a directory listing.

Choosing an editor from the all around is not a simple task, so try to test this editor as it is a good tracker with all you need that stays compacted in a screen!

Inside DigiOrganizer

by Stefano Tognon <ice00@libero.it>

DigiOrganizer is a great peaces of software wrote by Polonius of Padua for making 4bit sample music using SID volume register. I want to show you how the editor can be used, illustrating all the processes you may use for producing yourself sampled music.

However, in the style of SIDin, I cannot not give more details like the complete reverse engineering source of the player code!

Intro

When you load the program, a nice intro starts, with two scroll texts that introduce the editor and a great animation is performed into the logo pictures.

Music is very interesting and I think you will listen to it for some minutes before pressing a key!

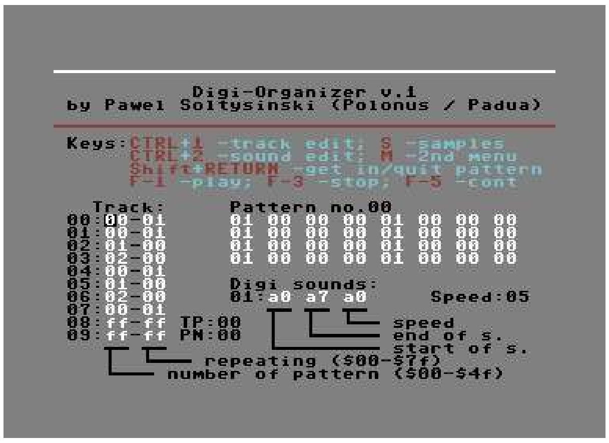
After so, the texts will disappear and after a while, even the logo will do so with a fade out music.

Now, some text screens show the porpoise of this editor and how you may use it.

Editor

DigiOrganizer is an editor that allow you to add a digi-sample music onto a normal (singlespeed) tune you had compose with other editors.

I think that this is a good solution, as you can use your favorite editor and extend the sound with full 4° voices in a easy way. There are however some problems into this cohabitation but these will be shown later.



```
        Digi-Organizer v.1
by Pawel Soltysinski (Polonus / Padua)

Keys:CTRL+1 -track edit; S -samples
     CTRL+2 -sound edit; M -2nd menu
     Shift+RETURN -get in/quit pattern
     F-1 -play; F-3 -stop; F-5 -cont

  Track:         Pattern no.00
00:00-01         01 00 00 00 01 00 00 00
01:00-01         01 00 00 00 01 00 00 00
02:01-00         01 00 00 00 01 00 00 00
03:02-00         01 00 00 00 01 00 00 00
04:00-01
05:01-00         Digi sounds:
06:02-00         01:a0 a7 a0     Speed:05
07:00-01
08:ff-ff TP:00                speed
09:ff-ff PN:00                end of s.
                              start of s.
              repeating ($00-$7f)
           number of pattern ($00-$4f)
```

The first step to do after the editor is open is to load the demo tune from the disk.

Press M and M again, then insert MODULE/DEMO for loading the sample demo part.

Press F1 and you will heart the sample music.

Press M and L and insert MUSIC/DEMO, then the address of the location that activate the music (usually $1000, but you can have music until $7500).

Press F1 and now you will heart all the 4 voices music! Very simple.

This is essentially how you have to use this editor with an external music.

But now it is time to use the editor for making our music.

In the left part of the screen there is the *track* menu. In this menu you have to insert, in the first position, the number of pattern to play, while in the second position the *number of time-1* to repeat this pattern.

So, for example:

```
        01 04                  play pattern 1 for 5 times
        02 00                  Play pattern 2 for 1 times
```

A $FF value means to restart the playing from the beginning, while $FE means to stop the reproduction of sample.

In the right part of the screen there are the locations where insert the pattern values. You have a maximum of 32 values, but you can insert a $FF where you want to finish a pattern before reaching the last position.

A not 00 value you inserted in those locations is the number of pattern to reproduce, while a value of 00 means to not take any sample action (so, eventually you will continue to heard the playing of previous set sample if it is a long sound).

An example:

```
        01 00 00 02 03 FF
```

In this example the sample 1 is played and his execution can go for other two tasks. Then samples 2 and 3 are played, before the pattern is finished.

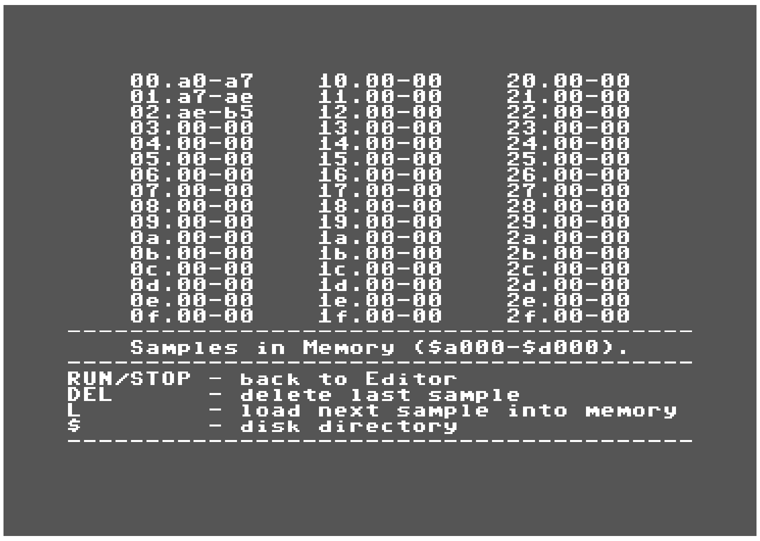
Samples

Now there is the final step: how are samples declared?

Press S for going into sample menu, then press L and insert the name of one sample to load from:

"tom 1"	"noise 1"	"lazer zokk"	"small snare"
"tom 2"	"noise 2"	"backward zokk"	"2nd snare"
"synth tom"	"noise 3"	"double zokk"	"snare 2"
"dx tom"	"noise 4"	"kraftzokk"	"snare reverb"
"electom"	"noise 5"	"laser"	"snare sk.or die"
	"noise 6"	"underwater 1"	"sanxionsnare"
"base 1"			"kraftsnare"
"base 2"	"pueaaa"	"hihat closed 1"	"triadsnare"
"base reverb"	"puuu!!!"	"hihat closed"	
"base sk.or die"	"paaaa!!!"	"hihat open"	"ride"
"sanxionbase"	"koud he"	"dish 1"	"roadrunner"
"kraftbase"	"uuuu!"	"crash"	"scream"
"base x"	"atsyou!"	"triadcrash 1"	"bark"
	"arr!"	"triadcrash 2"	"little dog"

"sanxionchord 1"	"ayeea!"		"burp"
"sanxionchord 2"	"arrh"	"bassguitar 1"	"glass"
"sanxionchord 3"	"klack!"	"bassguitar 2"	"horse"
"chord"	"oouuuooo!!"	"synthbass"	"cork"
"thank you robb"	"paou!"		"bell"
"bigbow"		"guitar accord"	"car"
"nightmare"		"guitar 1"	
"boom"		"guitar 2"	
"touch"		"guitar 3"	
"destroyed egg"		"guitar 4"	
		"guitar 5"	



```
    00.a0-a7    10.00-00    20.00-00
    01.a7-ae    11.00-00    21.00-00
    02.ae-b5    12.00-00    22.00-00
    03.00-00    13.00-00    23.00-00
    04.00-00    14.00-00    24.00-00
    05.00-00    15.00-00    25.00-00
    06.00-00    16.00-00    26.00-00
    07.00-00    17.00-00    27.00-00
    08.00-00    18.00-00    28.00-00
    09.00-00    19.00-00    29.00-00
    0a.00-00    1a.00-00    2a.00-00
    0b.00-00    1b.00-00    2b.00-00
    0c.00-00    1c.00-00    2c.00-00
    0d.00-00    1d.00-00    2d.00-00
    0e.00-00    1e.00-00    2e.00-00
    0f.00-00    1f.00-00    2f.00-00
----------------------------------------
    Samples in Memory ($a000-$d000).
----------------------------------------
RUN/STOP - back to Editor
DEL      - delete last sample
L        - load next sample into memory
$        - disk directory
----------------------------------------
```

After that, you will see that in the table will appears two values (e.g A0-A7) that are the high part of start and ending address of where the sample is loaded into memory.

You can so load up to $2F samples, by filling all the memory from $A000.

One limit of the program is that you can only delete the last inserted sample (by pressing L), but as the sample can have different lengths, deleting one sample in the middle of table should made difficult to fill the empty space without an automatic process that compacts the memory and move the pointers to the correct places.

The editor will fill the memory with $FF if the loaded sample did not reach the end of the memory block (so, volume will go to 15 after the sample is played).

Now return to the main menu and press CRTL+2. Here you can define how the sample inserted into the pattern value is formed:

01: A7 AE A0

Here, A7 is the high part of the starting address of sample in memory ($A700), AE is the high part of ending address of sample ($AE00), and $A0 is the speed of how the sample is played (we see later what physically this parameter means).

With this system, you can have one sample formed by more sequence of memory data. Suppose that we have load this three sample into memory:

A0-A4 Sample 1
A4-A7 Sample 2
A7 A9 Sample 3

If we have this definition:

```
01: A0 A9 A0
```

The sample played by inserting 01 value into patterns, is the sequence of Sample1, Sample2, Sample3.

At this point, the last things to know about this editor is that pressing SHIFT+A..I you can set the global tune speed.

Self made samples

If you think that the samples that are present in the disk are not enough for you, maybe you can add your own sample to the editor.

All you need is a *wav* file containing the sample (stereo or mono, better into mono for skipping a conversion from stereo to mono) of the sound you want to add.

Then you have to convert the file into 8 bits mono at 6KHz. This is achieved by reducing the bits from 16 to 8 and down sampling the frequency from 44KHz to 6KHz.

If you use Linux, simple give this command:

```
sox sample16.wav -c1 -b -v .8 -r 6000 sample8.wav
```

In this example, a 16 bit 44KHz wave file is converted into 6000Hz 8 bit unsigned, reducing even the volume to 80%. Maybe you should manipulate more the signal adding some low pass filters during the conversion, as the down sampling can introduce some spurious upper sound.

If you use other systems, maybe your player can save the wave file into the format we need.

At this point the wave file must be converted into 4 bit and it must be in C64 format (so, in 4+4 format: low nibble, high nibble of 2 samples). A good solution is to use the dos program S64_ENG (www.filety.prv.pl by Reiter/Apidya) that performs the task easily.

Now, goes into sample menu of DigiOrganizer and loads your sample into the editor and test if it sounds good.

Inside

Now it's time to see the internal of the editor by looking at his reverse engineering source. Please, take in mind that all code is copyrighted by Polonius and Padua group, so contact them for making a businesses use of it.

The editor is initialized by a call to $9000 and then at each IRQ, $9003 will be called. This is how all editor normally need to be used. So, your routine for 4° voices music may be like this:

```
Init:
          JSR  $1000          ; init of 3 voices music
          JSR  $9000          ; init of sample volume voice

Play:
          JSR  $1003          ; play of 3 voices music
          JSR  $9003          ; play of sample volume voice
```

During the init part, DigiOrganizer will activate a NMI (Non Maskerable Interrupt) from CIA #2 at timer B together with timer A. In particular, the speed of NMI generation (=speed of sample as the parameter you can set into the editor) is governed by the low byte of timer A.

The other important thing to know is that it sets the volume to 8 (a middle value) and sets the low pass filter flag governed by $D418. This means that if you use filter into music (not recommended by Polonius's note) only the low pass filter may be activate (as it is reseted at each NMI when playing a sample).

I don't know why this flag is set, as apparently it has no effect. Maybe you can hack the DigiOrganizer code for not forcing it if you want to use full filters into your music.

Last thing: make sure that your 3 voices player did not modify the volume during IRQ otherwise some very high noise sound will appears while playing the sample!

As you can see from the source, the most important part of the code is the NMI routines of the player.

The player use three routines:

- nmi_null
- nmi_sx
- nmi_dx

nmi_null simply did not do anything and is so activate at the beginning or as soon as a sample is completely played.

nmi_sx takes the left nibble of the sample and outputs it to the volume. Then it activates *nmi_dx* in the next NMI call.

nmi_dx takes the right nibble of the sample and outputs it to the volume. Then it sets the address for the next sample and the *nmi_sx* for the next NMI call. If sample is finished, *nmi_null* is activated.

The other parts of the code are very simple and essentially is for follow the flow of track/pattern/sample values and all is called by IRQ routine during the play phase.

Finally, the code present some data part that I use in the SidWine3 demo disk music, so look at the source of the music at http://digilander.iol.it/ice00/tsid/sidwine3 if you want to experiment with this data.

The code

```
  processor 6502
  .org $9000
      JMP  Init
      JMP  Play

  .byte "DIGI-ORGANIZER BY POLONUS/PADUA", $21

Init:
      sei
      lda  #$36
      sta  $01                  ; 6510 I/O register
      lda  #$E0
      sta  $21
      ldy  #$00
      sty  $20
clear:
      lda  ($20),y
      sta  ($20),y
      iny
      bne  clear
      inc  $21
      bne  clear

      dec  $01                   ; 6510 I/O register
      lda  #$70
      ldy  #$00
      sta  $DD04                 ; Timer A #2: Lo Byte
      sty  $DD05                 ; Timer A #2: Hi Byte
      sty  $DD06                 ; Timer B #2: Lo Byte
      sty  $DD07                 ; Timer B #2: HI Byte
      lda  #$11                  ; activate timer A, forced
      sta  $DD0E                 ; Control register A of CIA #2
      lda  #$51                  ; activate timer B, TOD allarm, count CNT+
      sta  $DD0F                 ; Control register B of CIA #2
      lda  $DD0D                 ; Interrupt control register CIA #2
      lda  #$82                  ; NMI with timer B
      sta  $DD0D                 ; Interrupt control register CIA #2
      lda  #$18
      sta  $D418                 ; Select volume and filter mode
      ldy  #>nmi_null
      sty  $FFFB
      lda  #<nmi_null
      sta  $FFFA                 ; Not Maskerable Interrupt (NMI) vector
      lda  #$00
      sty  curNTRep              ; actual number of repeat for this track
      sta  curTrackInd           ; actual track index
      sta  curPatPos             ; current pattern position
      jsr  resetDelay
      cli
      rts

curSpeed:                        ; actual values of timing delay (speed)
  .byte $03

curTrackInd:                     ; actual track index
  .byte $04

curPatPos:                       ; current pattern position
  .byte $06

tmpLow:
  .byte $00

tmpHigh:
  .byte $95

curNTRep:                        ; actual number of repeat for this track
  .byte $01

SPEED = $05

Play:
      dec  curSpeed              ; actual values of timing delay (speed)
      bmi  makePlay
      rts

makePlay:
      lda  #SPEED
      sta  curSpeed              ; actual values of timing delay (speed)
      lda  curPatPos             ; current pattern position
      bne  readPatValue

readActualTrack:
      lda  curTrackInd           ; actual track index
      asl
      tax                        ; =*2

readTrack:
      lda  track,x               ; read track value pointed by x
      tax
      cpx  #$FE                  ; is stop playing ?
      bne  testContinue
      lda  #<nmi_null
      sta  $FFFA                 ; Not Maskerable Interrupt (NMI) vector
      rts

testContinue:
      cpx  #$FF                  ; is play from beginning?
      bne  trackVal
      inx                        ; next track
      stx  curTrackInd           ; actual track index
      jmp  goToRead

trackVal:
      lda  curTrackInd           ; actual track index
      asl
      tay                        ; =*2
      lda  track+1,y             ; read number of repeat for this track value
      and  #$7F
      jsr  setRepeat
      lda  #$00
      sta  tmpHigh
      txa                        ; track value
      asl
      rol  tmpHigh
      asl
      rol  tmpHigh
      asl
      rol  tmpHigh
      asl
      rol  tmpHigh
      asl                        ; track*32
      rol  tmpHigh
      nop
      nop
      nop
      nop
      sta  tmpLow
      lda  tmpHigh
      clc
      adc  #>pattern
      sta  tmpHigh
      sta  pInd+2
      lda  tmpLow
      sta  pInd+1
      lda  curPatPos             ; current pattern position

readPatValue:
      tax
pInd:
      lda  pattern,x             ; read pattern value at x position
      bne  performeSample
nextPattern:
      inc  curPatPos             ; inc current pattern position
      lda  curPatPos             ; current pattern position
      cmp  #$20                  ; end of 1 pattern
      bcs  setForNextPT
      rts

;=================================
; set for next pattern/track
;=================================
setForNextPT:
      lda  #$00                  ; reset pattern position
      sta  curPatPos             ; current pattern position
      dec  curNTRep              ; actual number of repeat for this track
      bpl  skipTInc
      lda  curTrackInd           ; actual track index
      clc
      adc  #$01
      and  #$7F
      sta  curTrackInd           ; actual track index
skipTInc
      rts

performeSample:
      ldy  #<nmi_null
      jsr  testEndPattern
      asl
      asl
      tay                        ; pattern val*4
      lda  digiSound-4,y         ; high pointer of start of sample
      sta  sampAddSx+2
      sta  sampAddDx+2
      cmp  digiSound-3,y         ; high pointer of end of sample
      bcc  notModify
      clc
      adc  #$01
      jmp  setHighE

notModify:
      lda  digiSound-3,y         ; high pointer of end of sample
setHighE:
      sta  sampleEnd+1
      lda  digiSound-2,y         ; speed
      sta  $DD04                 ; Timer A #2: Lo Byte
      lda  #$00                  ; low pointer start from 0
      sta  sampAddSx+1
      sta  sampAddDx+1
      lda  #<nmi_sx
      sta  $FFFA                 ; Not Maskerable Interrupt (NMI) vector
      jmp  nextPattern

;=================================
; NMI for null sample output
;=================================
nmi_null:
      sta  tmpVal+1
      lda  $DD0D                 ; Interrupt control register CIA #2
tmpVal:
      lda  #$04
      rti
```

```
;=================================
; NMI for output the left nibble
; of the sample
;=================================
nmi_sx:
      sta  $F8
sampAddSx:
      lda  $A1F2                  ; read packet sample
      lsr
      lsr
      lsr
      lsr                         ; left nibble
      ora  #$10                   ; set low pass filter
      sta  $D418                  ; Select volume and filter mode
      lda  #<nmi_dx               ; set NMI for dx nibble
      inc  sampAddSx
      bne  skipInc
      inc  sampAddSx+1
skipInc
      sta  $FFFA                 ; Not Maskerable Interrupt (NMI) vector
      lda  $DD0D                 ; Interrupt control register CIA #2
      lda  $F8
      rti

;=================================
; NMI for output the right nibble
; of the sample
;=================================
nmi_dx:
      sta  $F8
sampAddDx:
      lda  $A1F1                  ; read packed sample
      and  #$0F                   ; right nibble
      ora  #$10                   ; set low pass filter
      sta  $D418                  ; Select volume and filter mode
      inc  sampAddDx+1
      beq  fixHighDx

      lda  #<nmi_sx               ; set NMI for sx nibble
      sta  $FFFA                  ; Not Maskerable Interrupt (NMI) vector
      lda  $DD0D                  ; Interrupt control register CIA #2
      lda  $F8
      rti

fixHighDx:
      inc  sampAddDx+2
      lda  sampAddDx+2
sampleEnd:
      cmp  #$A7                   ; test if sample is finished
      bcs  pauseSample
      lda  #<nmi_sx
      sta  $FFFA                  ; Not Maskerable Interrupt (NMI) vector
      lda  $DD0D                  ; Interrupt control register CIA #2
      lda  $F8
      rti

pauseSample:
      lda  #<nmi_null
      sta  $FFFA                  ; Not Maskerable Interrupt (NMI) vector
      lda  $DD0D                  ; Interrupt control register CIA #2
      lda  $F8
      rti

resetDelay:
      bit  curNTRep               ; actual number of repeat for this track
      lda  makePlay+1             ; selected SPEED
      sta  curSpeed               ; actual values of timing delay (speed)
exitAbove0:
      rts

;=================================
; Set the number of repeat for
; this track
;=================================
setRepeat:
      sta  numRep+1               ; number of repeat for this track
      lda  curNTRep               ; actual number of repeat for this track
      nop
      nop
      bpl  exitAbove0
numRep:
      lda  #$01
      sta  curNTRep               ; actual number of repeat for this track
exitRTS:
      rts

testEndPattern:
      sty  $FFFA                  ; Not Maskerable Interrupt (NMI) vector
      cmp  #$FF                   ; end indicator of pattern
      bne  exitRTS
      pla
      pla
      jsr  setForNextPT
      jmp  readActualTrack

goToRead:
      stx  curPatPos              ; current pattern position
      jmp  readTrack

  .org $9200
track:

  .byte $00, $01
  .byte $00, $01
  .byte $01, $00
  .byte $02, $00
  .byte $00, $01
  .byte $01, $00
  .byte $02, $00
  .byte $00, $01
  .byte $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF
  .byte $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF
  .byte $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF
  .byte $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF
  .byte $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF
  .byte $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF
  .byte $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF
  .byte $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF
  .byte $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF
  .byte $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF
  .byte $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF
  .byte $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF
  .byte $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF
  .byte $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF
  .byte $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF
  .byte $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF
  .byte $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF
  .byte $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF
  .byte $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF
  .byte $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF
  .byte $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF
  .byte $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF
  .byte $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF
  .byte $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF
  .byte $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF
  .byte $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF
  .byte $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF
  .byte $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF
  .byte $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF
  .byte $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF

  .org $9300
digiSound:
  .byte $A0, $A7, $A0, $A0
  .byte $A7, $AE, $A0, $A0
  .byte $AE, $B5, $A0, $A0
  .byte $A0, $A0, $A0, $A0,  $A0, $A0, $A0, $A0
  .byte $A0, $A0, $A0, $A0,  $A0, $A0, $A0, $A0
  .byte $A0, $A0, $A0, $A0,  $A0, $A0, $A0, $A0
  .byte $A0, $A0, $A0, $A0,  $A0, $A0, $A0, $A0
  .byte $A0, $A0, $A0, $A0,  $A0, $A0, $A0, $A0
  .byte $A0, $A0, $A0, $A0,  $A0, $A0, $A0, $A0
  .byte $A0, $A0, $A0, $A0,  $A0, $A0, $A0, $A0
  .byte $A0, $A0, $A0, $A0,  $A0, $A0, $A0, $A0
  .byte $A0, $A0, $A0, $A0,  $A0, $A0, $A0, $A0
  .byte $A0, $A0, $A0, $A0,  $A0, $A0, $A0, $A0
  .byte $A0, $A0, $A0, $A0,  $A0, $A0, $A0, $A0
  .byte $A0, $A0, $A0, $A0,  $A0, $A0, $A0, $A0
  .byte $A0, $A0, $A0, $A0,  $A0, $A0, $A0, $A0
  .byte $A0, $A0, $A0, $A0,  $A0, $A0, $A0, $A0
  .byte $A0, $A0, $A0, $A0,  $A0, $A0, $A0, $A0
  .byte $A0, $A0, $A0, $A0,  $A0, $A0, $A0, $A0
  .byte $A0, $A0, $A0, $A0,  $A0, $A0, $A0, $A0
  .byte $A0, $A0, $A0, $A0,  $A0, $A0, $A0, $A0
  .byte $A0, $A0, $A0, $A0,  $A0, $A0, $A0, $A0
  .byte $A0, $A0, $A0, $A0,  $A0, $A0, $A0, $A0
  .byte $A0, $A0, $A0, $A0,  $A0, $A0, $A0, $A0
  .byte $A0, $A0, $A0, $A0,  $A0, $A0, $A0, $A0
  .byte $A0, $A0, $A0, $A0,  $A0, $A0, $A0, $A0
  .byte $A0, $A0, $A0, $A0,  $A0, $A0, $A0, $A0
  .byte $A0, $A0, $A0, $A0,  $A0, $A0, $A0, $A0
  .byte $A0, $A0, $A0, $A0,  $A0, $A0, $A0, $A0
  .byte $A0, $A0, $A0, $A0,  $A0, $A0, $A0, $A0
  .byte $A0, $A0, $A0, $A0,  $A0, $A0, $A0, $A0
  .byte $A0, $A0, $A0, $A0,  $A0, $A0, $A0, $A0
  .byte $A0, $A0, $A0, $A0,  $A0, $A0, $A0, $A0
  .byte $A0, $A0, $A0, $A0

  .org $9400                      ; probably spurious bytes
  .byte $A0, $A7, $AE
  .org $9430                      ; probably spurious bytes
  .byte $A7, $AE, $B5

  .org $9500

; group of 32 values of one pattern
pattern:
  .byte $01, $00, $00, $00, $01, $00, $00, $00
  .byte $01, $00, $00, $00, $01, $00, $00, $00
  .byte $01, $00, $00, $00, $01, $00, $00, $00
  .byte $01, $00, $00, $00, $01, $00, $00, $00

  .byte $02, $00, $00, $00, $02, $00, $00, $00
  .byte $02, $00, $00, $00, $02, $00, $00, $00
  .byte $02, $00, $00, $00, $02, $00, $00, $00
  .byte $02, $00, $00, $00, $02, $00, $00, $00

  .byte $03, $00, $00, $00, $03, $00, $00, $00
  .byte $03, $00, $00, $00, $03, $00, $00, $00
  .byte $03, $00, $00, $00, $03, $00, $00, $00
  .byte $03, $00, $00, $00, $03, $00, $00, $00

  .org $A000
; here goes all the digi. high page is filled with $FF for complete the samples.
  .incbin samples.bin
```

Conclusion

DigiOrganizer is very simple to use and I think you can add digi to your tune in an easy way.

The possibly of test the final result of the four voices tune inside the editor is a good characteristic and the disk contains lot of samples that probably give you all you need from the beginning.

Maybe this editor could be more powerful if there was the possibility of move the loaded samples into memory and if we could give a mnemonic name to the loaded sample for better managing them.

.

SIDin 6 end