Commodore Free

Issue number 10
July 2007

FREE to download magazine dedicated to all Commodore Computers available FREE at the end of each month
from www.commodrefree.com

--- Forwarded message ---

I regret to advise the Commodore community that Jim Butterfield has passed away. Jim died at 1:30 AM on June 29 after battling cancer which infected many parts of his body.

His family advises that there will not be a funeral as such but a commemoration of Jim's life is planned in the next month or two.

At the moment that is all the detail that I have to report.

We have all lost a truly wonderful friend and teacher.

Ernie Chorny

Editor

Very sad news this issue with the passing of Jim Butterfield on the 29th of June 2007, For many Jim was an inspiration taking complex information and changing it into a format many people found easier to digest. I am sure he will be missed by a great many people.

Jim`s Work should not be lost, we should pool our efforts and create a website with all his works in one place. Other younger users may not have read or met Jim and should not have to go without his words of wisdom.

I try to contact users and manufacturers that I would like to read about as a Commodore user**,** for example last month was an interview with worldlam about potential purchase of CMD and why he is buying up so much Commodore equipment. Some people felt the interview wasn't justified and a waste of time. I disagree because it raises the question about CMD and the problems Maurice is having in manufacturing the products. Maybe Maurice should sell the hardware and concentrate on the software side.

Many people feel angry to Maurice, I myself feel some anger that Maurice took my money knowing full well he could not provide the goods in the time scale he promised. Also Maurice won't answer any of my emails, and requests for a refund of my money.

We need good companies supporting Commodore but having a company take money and then have problems providing the goods is just having a negative effect on the community.

I was ready for some hate mail from the Worldlam interview, although it wasn't realy hate mail more that people felt there were others who should have been interviewed instead.

Ok interview them and send me the questions and answers session, I will print the text in the magazine remember the Commodore Free team is just ME, and it's a lot of work to produce in just 1 month, also remember the amount of work that goes into asking questions, some users agreed to interviews and I would write the questions but never receive answers for whatever reason, this is fine but it takes me more time, I need your input and articles

Thanks
Nigel
www.commodore free.com

Can you help
If you are working on a project, hardware or software based for any of the Commodore machines please can you let me know, so I can include it in the magazine.

Want to help but don't know what to do and don't think you could write a review, just send a link, don't worry if I may have seen it or not, send the link so I can contact the creator for more information

Thanks

Contents

Commodore Free is a FREE to download magazine in PDF TEXT D64 image and Html normally produced at the end of every month

NEWS

The Internet for Commodore C64/128 Users by Gaelyne Gasson is now available as a PDF download from the Commodore Central Market, which now is part of our online store. We actually have two PDF versions available. One is an exact page-for page copy of the 3rd edition for those who want the book completely as it was written and formatted. The other PDF version has been edited to include 'PDF bookmarks' for chapters and subheadings, and references to other chapters have been made into clickable links. Also URLs that are still valid have also been made into links.

TIFCU is not just for Commodore users these days. Several years ago we moved our servers to Linux, and for the first few months we found TIFCU to be invaluable for looking up how to do basic things on our system. We've heard from several other people who've found it useful for use with their Linux machines as well.

The price is $10.00 Australian (about $8.50 US) and once payment is received you can download the PDF file immediately from the store.

We're running a special offer on the TIFCU book and Homestead Plus Membership. If you purchase a Homestead Plus Membership ($35 AU or about $29 US), you can receive the book at half price. Homestead Plus is the name of our Telnet memberships as it better reflects the fact it's used by members of the Homestead mailing list. It's also quite OK to have a membership without actually using the Telnet services. Some folks have done this as a way to donate to our cause of supporting Commodore computing and naturally we are pleased with all our Homestead Plus Members whether they make use of it or not. :-)

The special offer for TIFCU at half price also applies to Homestead Plus Membership renewals paid for via our online store.

We've moved the Commodore Central Market into the same shopping cart system that we use for other items we have for sale (gameboy items, phone covers, webcams, breath testers and other various items). The C= Central Market has it's own area which can be found at:
http://shop.vcsweb.com/index.php?main_page=index&cPath=24

One of the features of our new online store is that people can write reviews of products, and we're looking to add more reviews to our various C= Central Market items. If you've purchased something from the market in the past and write a review of it, we'll give you 10% off your next purchase. Just visit any item in the store, click on the 'Reviews' link and the next page will have a link to submit your review. It will ask you to login or to sign up as a store member before you can submit your review - this will make it faster to place an order and allow us to give you the 10% discount.

Please come have a look at the new C= Central Market today! The link is:
http://shop.vcsweb.com/index.php?main_page=index&cPath=24

All the Very Best,
Gaelyne & Rod Gasson

High Voltage SID Collection Update #47 Date: June 07, 2007

Resulting Version: 47
Previous Version: 46

As usual, the update and the all-in-one packages are available from http://www.hvsc.c64.org After this update, the collection should contain 34,127 SID files!

This update features (all approximates):

1127	new SIDs
19	fixed/better rips
8	fixes of PlaySID/Sidplay1 specific SIDs
8	repeats/bad rips eliminated
773	SID credit fixes
700	tunes assigned a sidmodel flag
15	UNKNOWN demo tunes identified
29	UNKNOWN game tunes identified

Main Composers featured in this update:
(Artists marked with NEW are either completely new to the HVSC or they get their own directory in this update)

A-Man - venturing into Pollytracker domain now!
Bart
Richard Bayliss
Bernhard Burgstaller (NEW)
Chantal Goret (NEW)
CRD - kindly donated his complete collection to us
Dexter (NEW)
Eco
Fanta - listen to his Desert Dream conversion!
Fox
Gop
Greg
Gregfeel
Harlequin
Heinmuck
Hukka
Image
MAC2
Mac / Radical
Merman
Moogle Charm (NEW)
Nastiness Inc.
Omoroca (NEW)
Pernet
Q-Man
Raze
Rio
Sax
Sharp
Skam (NEW)
Slayer
STP Sound System
Tonid (NEW)
Topaz
Vintaque
Zeta (NEW)

HVSC News

Motion joined the HVSC Crew

Steppe retired from the HVSC admin post. Rambones will take over, good luck!

The new directory structure requires a new update tool. It got updated for Linux and Windows, so we feel the majority of users won't have a problem. For the exotic platform users: The source code of update tool 2.8.4 ison the HVSC website in the Downloads section. If you manage to compile it on your specific platform, feel free to send it over! And by the way: You can still run update #47 with update tool 2.8.3.It will complain heavily that the /Hubbard_Rob/ directory is not where it expects it to be, thus assuming you did something fundamentally wrong. Just ignore the warning, nod away the next "y/n are you sure you're sure?" question and it will work anyway.

SID related news bulletin
STIL Bonus released For all you STIL junkies out there curious about some of the module tunes mentioned in the STIL, the HVSC team now offers the STIL Bonus:

http://www.prg.dtu.dk/HVSC/STIL_Bonus/

The STIL Bonus is a collection of MODs, XMs, MP3s, and other assorted tunes referenced in the STIL. This collection was previously available under a different name (MCOT 64) but on a much smaller scale. The collection is now 63MBs compressed, and offers some very memorable game and scene tunes from the Amiga, PC, and other sources.

The collection was put together in order for the HVSC team to verify several STIL entries for correctness. We then thought it would be a nice collection that the public may also enjoy. For some of you, you may have never heard these tunes. For others, it may be a trip down nostalgia lane. You may even find that you actually prefer some of the SID versions while for other cases the modules/MP3 will be preferable.

PSID64 v0.8 is out!
The popular .sid->.prg converter saw another update. New features include:

* Built-in support for Exomizer.
* Added support for a joystick in port two to control the player.
* Added clock to indicated running time.
* Added -i option to override the initial song to play.
* Added support for SID tunes written in BASIC.

Grab it here: http://psid64.sourceforge.net/

Ebay News

Commodore FREE I managed to find this on ebay

Commodore 64 Disk-Rob Hubbard and Martin Galway Music

You are bidding for a Commodore 64/128 disk comprising of VERY RARE music demos by two of the leading artists from the eighties, Rob Hubbard and Martin Galway.

Side A is a very very rare music collection of Rob Hubbards most famous tunes. You will be hard pressed to find this demo disk anywhere nowadays, very rare and highly collectable, I doubt you will find this on any rom collection, this demo is not even included on the TOSEC Commodore set which includes 1000's of games, demos, compilations etc.

Simply load the disk into your drive and press keys A-Y, some of the tunes on this disk include:

Crazy Comets
Monty on the Run
One Man and His Droid
Thing On A Spring
Zoids
Bump Set Spike

and much much more

Side B includes a great collection of Martin Galway tunes from yesteryear, again this demo disk is very hard to come by. Again, simply load in to your Commodore and press keys A-O , Some of the tunes include:

Highlander
Ccean Loaders 1-3
Parallax
Comic Bakery
Hyper Sports
Wizball
Rambo Loader

and much much more.

This disk is in full 100% working order. Bid with confidence from a trusted eBayer!
Keep an eye on my auctions for more of this type of material in the future.

Postage will be free if purchased with buy it now! (UK ONLY)

Commodore Free
Selling price was £4.80 + 1.50 postage costs for what looked like a new disk with colour picture of rob and Martin on the Cover, but it looked like it was just produced, couldn't you just download the sid files Free. Expect a large amount of these disks to appear soon. To the buyer it must have been worth it, but I remember years ago downloading some of these compilations, and somewhere I must still have around 10 disks with music files on, with a nice front end press A for XXX B for YYY etc.

QVC

Qvc in the Uk is a shopping channel, it featured a pc special with Commodore, the program started with an about Commodore brand (of old) and theCommodore 64 and Amiga machines, The presenter confessed to owning an Amiga

Then the presenter and Commodore Representative went thorough the machines specifications and different cases, the unit was available with monitor keyboard and speakers or just as a base unit, They have a retro design that looks like a Commodore 64, very neat looking case, Maybe they should sell just the case with no internals

IDE64 V4 Prototype

As you can see in the picture the IDE64 V4 prototype is ready. Although there are still some work need to be done before it will be finished, we would like to present one of the major improvement which is increased performance.

The IDE64 V4 is faster by 16% than IDE64 V3.x and even 37% faster than RamLink while loading the same amount of data.We would like to thanks to all C64 users (who asked us about IDE64 availability) for the patience. We are sorry IDE64 is not available at the moment.

posted by Josef at 5/21/2007

Commodore Scene VGA project terminated

Known as C=VGA

www.commodorescene.org.uk

Alan has recently informed me he has terminating the Commodore VGA project. The project was to design a self contained unit to connect Commodore 64 and 128 to standard VGA monitors, the unit would also allow the use of 80 columns and all colours on LCD and CRT monitors

Alan released a statement to me

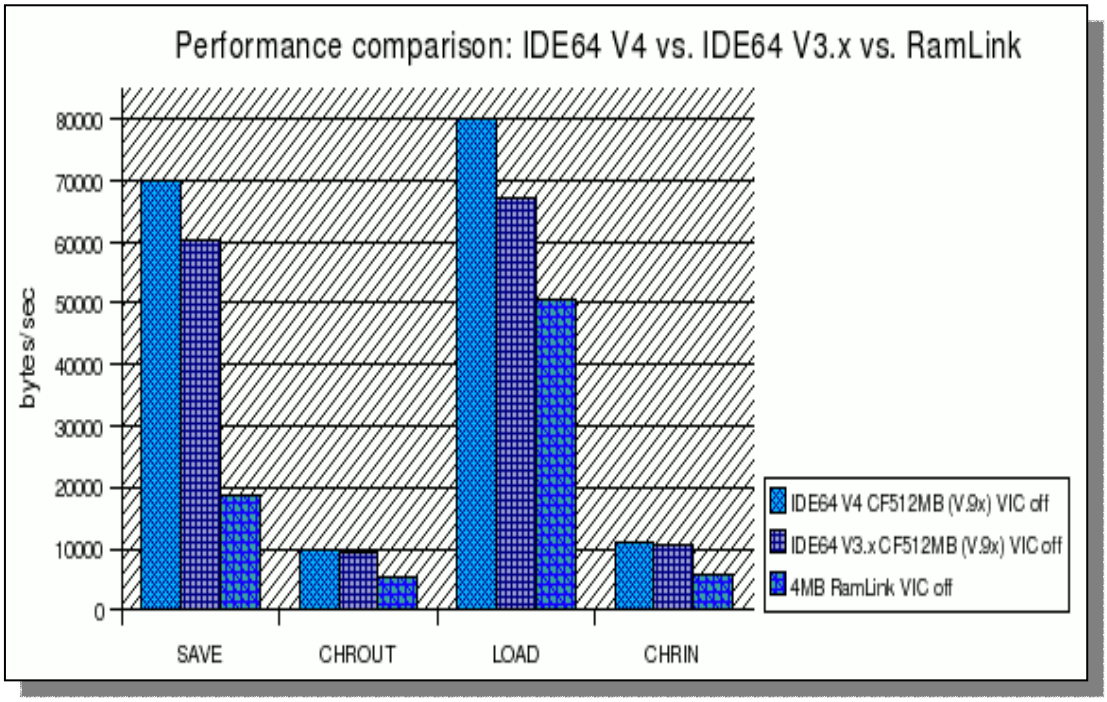
“I didn't get anything although I did see the working prototype many years ago. I've closed the project on my web site so its common knowledge, so please pass on this information. BUT, I am still going to try this new device(s) as a matter of urgency and I am ordering the parts today so I would appreciate any mention of the C=VGA closing also mentions that I am actively sorting out an alternative which I hope to have completed by the end of July.”

Well the meeting in Florida didn't take place and so I have had to make the decision to close this project. All money will be refunded.

However, that is not the end of the Commodore/VGA story just yet. Please pop along to http://home.comcast.net/~kkrausnick/c128-vga/c128-vga.html and see what is happening there. I am ordering the components to make the same set up myself and if all is well I will post a components list and installation instructions up here during July

So closes another project, Alan has said to “watch this space though” Alan has said will be refunded. Lets hope Alan can arrange another suitable alternative

.



Programming The Great Adventure of Creativity and Logic by Dave Moorman

Come with us as we enter a strange world of symbolic logic. Process Logic, to be exact. If you do those logic problems found in puzzle magazines, you know Classic Logic. If you found Geometry enjoyable in high school, you know Proof Logic. And if you studied philosophic logic in college, you are aquainted with Symbolic Logic.

Process Logic is a combination of Proof Logic and Symbolic Logic -- plus three wonderful additional features. Like Proof Logic, you will be arranging statements and commands in a particular order -- not to prove some truth, but to affect some action in the computer. In the process, you will use Symbolic Logic to manipulate values.

The three additional features are
1. symbolic value holders(called variables and arrays),
2. loops
3. conditional commands. It is the ability to make conditional changes in the flow of logic that gives a computer its ability to "think."

The C-64 includes a built in BASIC interpreter. Computers are controlled with three types of language. At its very heart, the computer processor recognizes certain values as "instructions." This is built into the machine itself, and is called Machine Language. EVERYTHING the computer does is really done by means of ML.

ML is nothing but numbers, that is, numeric values. Remembering such values and the tasks they perform can be extremely difficult for humans. We need at least some easily recognizable code "words" to remind us about what is going on. The ML programmer writes these "words" and the computer uses a program to ASSEMBLE that code into ML -- which is what the computer actually understands. The code the programmer writes is called ML, or more correctly, Assembly Source Code.

But a computer can be smarter than that. Assembly ML source code has a one-to-one relationship with the code the computer understands. Each instruction does one and only one very small task. Keeping it all straight can be quite frustrating.

However, the computer can be programmed to read words, numbers, and other characters and translate them into complex groups of ML instructions. Such a language is called Compiled. The program that translates Compiler Code into ML Code is called a Compiler. A compiler compiles an entire program or routine at a time. If the programmer has made a mistake the compiler cannot understand, it reports errors -- but only after chugging through the whole source code. So the programmer writes, compiles, debugs, recompiles, executes, rewrites, recompiles, etc, etc. This is an arduous task, to say the least. It was even more frustrating back in the 1960's when the programmer had to punch cards with each line of the program and take the "batch" to the computer room. The operator would run the batch and return a paper print-out to the programmer in a few hours. Or days!

At that time, computers (big mainframes) were finally becoming fairly fast and powerful. A terminal could be directly connected to the computer so the programmer did not have to wait. But the computer then did a lot of waiting -- for the programmer's input. The concept of time-sharing was developed, where the computer could switch between many different terminals, running different programs at apparently the same time.

To take advantage of time sharing and to provide a language that was easy for students to learn and use, BASIC (standing for Beginner's All Purpose Symbolic Instruction Code) was written (invented) in 1963, at Dartmouth College, by mathematicians John George Kemeny and Tom Kurtzas. The commands and math the programmer typed looked enough like English, making reading the code relatively easy.

But the big advantage of BASIC was that it was -- and is -- an Interpreted Language. During the user's tiny slice of processor time, a single BASIC statement would be read, turned into the ML Code necessary to execute the command, and processed. Then the processor turned to another terminal and program to process. On the BASIC program's next turn, the next BASIC statement would be interpreted and executed.

The great thing about an interpreted language like BASIC is that the program runs until an error occurs. Then it stops and delivers an error message. The programmer can fix the error and rerun the program. This made BASIC very interactive. The programmer did not have to get everything right before seeing how at least SOME of the program performed. In December of 1974, the January issue of Popular Electronics published news about the first home computer -- the Altair 8800. Two Harvard students, Paul Allen and Bill Gates saw the magazine -- and their future. They dropped out of college and rushed to Albuquerque, NM, where the Altair was being built.

They realized that these home computers needed an "operating system" -- a simple way for users to interact with the machine. Bill Gates wrote Altair BASIC using a mainframe computer with an emulator that made it act like the Intel 8008 microprocessor used by the Altair. His BASIC included ML code to read the keystrokes and put the BASIC program into memory. Other code would read BASIC commands and jump to ML routines that performed them. The whole thing fit in just 4 kilobytes of memory (which doesn't seem like much today, but was rather expensive at the time).

Gates and his newly founded company -- MicroSoft -- went on to write BASIC for nearly every home computer. BASIC 2 used about 16K of memory, but was remarkably powerful. Most anything a programmer wanted to do could be done in BASIC. True -- it was slower than straight ML. But is was easy to learn, faster to write, and more-or-less portable between different makes of computers.

When in 1978 Commodore Business Machines produced the Personal Electronic Transactor -- the PET -- they turned to Microsoft for BASIC. Commodore CEO Jack Tramiel bought BASIC 2.0 outright for $10,000 from cash-strapped Microsoft.

So, in the fall of 1981 when Commodore designed the C-64, they already owned the BASIC 2.0 operating system. The C-64 has color video and other features for which BASIC 2.0 had no commands. But that was OK. Game designers would certainly use fast ML for their code. And BASIC 2.0 has commands which can directly read or write information to places in memory that will control these features.

On the up side, the C-64 was designed to be modified with ML code. Though BASIC 2.0 was in Read Only Memory (ROM) and could not be changed, certain critical jump locations were in Random Access Memory (RAM -- which can be altered). By changing the jump addresses, a programmer could add new commands to BASIC and perform all sorts of miracles the designers never dreamed of. The designers did include "paddle controls" for then-popular games like Break Out. These controls proved perfect for adding a mouse.

All in all, a C-64 was a fantastic machine in 1982 when it was unveiled at the January Consumer Electronics Show in Las Vegas. Its capabilities -- especially as a "game machine," and its incredible price that dropped to less than $200 in 1994, kept it in production through 1992. Over its decade of manufacture, some 27 million units were sold, making the C-64 the "Best Selling Computer of the 20th Century," according to the revered Guinness Book of World Records (2000-2001).

In the late 1990's, PC programmers who loved the C-64 began writing emulators to allow a PC to run C-64 programs. The best of these, the Versatile Commodore Emulator, continued to be improved until it brings an almost perfect C-64 to the world of Windows. With VICE, all the great games published over the years on LOADSTAR are now at your finger tips. And to top of a remarkable (if often ignored) history, a C-64 Direct-to-TV game joystick was marked in 2004 through QVC, the shopping channel. Over 200,000 unites were sold between Thanksgiving and Christmas.

Thanks to the designer -- Jeri Ellsworth -- the computer inside the joystick is a real, honest-to-goodness C-64. With nine wires soldered to the credit-card-sized board, a user can connect a PS2 keyboard, Commodore disk drive, and an external power supply.The Commodore 64 - more than any other first-generation, 8-bit computer -- has proved itself as THE computer for gamesters and hobbyist programmers all over the world.

INSIDE the C-64
Every computer has three essential parts --

1. A processor which executes ML instructions and does math and logic operations.
2. Input/Output capabilities -- for keyboard, mouse, joystick, printers, and disk drives.
3. Memory -- "itty-bitty boxes" called bytes which can each hold a value of 0 through 255.

Today's computers can handle up to 8 bytes at a time, making them incredibly fast. Such speed is necessary for processing sound recording, photo-quality images, and real-time videos.

The first generation computers (such as the C-64) had processors that could handle only one byte at a time. A byte is composed of eight bits -- little switches which are on or off, 1 or 0 -- hence these are called "8-bit computers." A two-byte value is used to point to a particular byte in memory, which means that an 8-bit computer is limited to 256 x 256 or 65536 bytes of memory. One kilobyte is actually 1024 bytes, so 65536/1024 equals 64. Thus the name Commodore 64. But the C-64 has more than 64K of memory. The ROM which holds all the ML code to make BASIC wois "banked" on top of RAM. By "flipping" certain bits in the computer, an ML programmer can set ROM aside and use the RAM "underneath."

As mentioned before, even though BASIC 2.0 is powerful, ML programmers have created a number of BASIC extensions and ML modules to add features for BASIC programmers. In 2004, LOADSTAR featured DotBASIC which adds 72 commands to BASIC 2.0 -- including full mouse control and Event Driven programming. Other modules play music and sound effects in the background, enable easy access to bitmap graphics, and even let ordinary BASIC programmers operate sprites (movable screen objects) and do split screen effects.

THE LIMITATIONS
The 6510 microprocessor in the C-64 operates at 1 megahertz (millions of cycles per second) -- which is slow compared to the 8+ Gigahertz (billions of cycles per second) Pentiums now on the market. However, 1Mhz is still 1,000,000 clock cycles per second (not slow, really) and 6510 ML instructions are quite efficient compared to Pentium instructions. The screen is comprised of 60,000 pixels -- 320 x 200. In multi-color mode, two bits determine which of four colors will be displayed as double-wide pixels (160 x 200). The result is a bit grainy.

Text characters are all 8 x 8 pixels in size, and can display the character "cell" color or the background color. In multi-color text mode, double-wide pixels can present the character color or one of three "universal" colors. The font includes 256 characters, but the programmer is not limited to the two built-in fonts. With a font editor, one can design characters as any 8 x 8 combination of pixels. Moreover, the programmer has eight 24 x 21 pixel sprites -- movable objects -- that can be designed and displayed anywhere without disrupting the screen.
The whole screen can be nudged, pixel by pixel, in any direction, enabling smooth scrolling effects -- especially when combined with split screen capabilities.

Sound is limited to three synthesized voices. The synthesizer has only Attack, Decay, Sustain, and Release envelope parameters, but does include various filters and resonance settings. Waveforms include noise, sawtooth, triangle, and adjustable pulse. And, with some cleverness, 4-bit recorded sound can be recorded and played by the computer.

So while the C-64 has certain limits, it is crafted in such a way that truly capable programmers can accomplish most anything computational. I have seen real-time three-dimensional displays (like DOOM, only very low resolution), the MGM Lion roar, and hundreds of other truly amazing sound and video effects. But most important, anything one can do on any computer can be at least MODELED on the C-64. The model may be rough, but the concepts, skills, and personal satisfaction in accomplishing effects are the same as with a big computer.

The day may come when you will want to tackle C, C++, C#, Java, Java Script, Perl, or Visual Basic on a PC. EVERYTHING conceptual and logical learned on the C-64 will apply to any other computer or language.

The C-64 is where one starts -- as did thousands of today's professional software designers. And hundreds of hobbyist programmers still find enough challenge to sit up all hours of the night hunching over their C-64s, fixing just one more thing!

ON TO BASIC 2.0
When you turn on the C-64 (or launch VICE), the screen displays some title information, then presents the word READY.

READY? Ready for what?

Ready for anything you want to do! Type:

PRINT "HELLO"

and press <RETURN>. (On VICE, the double-quotes are <Shift-2>, and <RETURN> is the <ENTER> key.)

The computer immediately complies -- printing to the screen:

HELLO

Type: ? 5 + 7 * 10

(VICE: the plus is the <=> key, the asterisk is the <]> key.) (Remember to press the <RETURN> which tells the C-64 to go ahead and do it.)

75

The question mark is short for PRINT. In the first example, you printed a STRING, a group of characters in order. You marked off the beginning and end of the string with double-quotes.

In the second example, you printed numeric values, multiplied and added according to mathematic rules (multiply and divide are performed first, followed by addition and subtraction). Both of these examples are examples of Immediate Mode. The C-64 immediately responded to your commands when the <RETURN> was pressed.

Now Type:
```
10 ? "HELLO"
20 ? 5 + 7 * 10
```

(pressing <RETURN> at the end of each line).

Nothing happened -- at least not obviously. But inside the C-64 a lot has taken place. Type:

LIST

and the lines appear again. You have written these lines in Program Mode. The difference between Immediate Mode and Program Mode is very simple ---

If a NUMBER comes first in the line, the line is put in Program Memory. You can look at the program with LIST. If a command comes first, the line is in Immediate Mode and processed immediately.

To run your program, Type:

RUN

Wonderful! You have written your first program!
The number you use at the beginning of a Program line determines where that line occurs in the program. Type:

15 ? "WORLD"

and LIST

```
10 PRINT "HELLO"
15 PRINT "WORLD"
20 PRINT 5 + 7 * 10
```

Since every program line must have a number and will be ordered by its number, it is a good idea to use 10's as you start writing your program. Then, if you want to tuck something in between two existing lines, you can -- just like you just did.

VARIABLES

A variable is a "box" that contains something. Each variable has a name, comprised of one or two letters or a letter and a number. Longer variable names are OK (unless they contain a BASIC command word), but the computer will not "see" more than the first two characters.

Let's wipe out your first program. Type:

NEW and LIST

It's gone. Now Type:

```
10 A1$ = "HELLO"
20 A2$ = "WORLD"
30 A1 = 5
40 A2 = 7
50 A3 = 10
100 ? A1$ + A2$
110 ? A1 + A2 * A3
120 END
```

We have two types of data -- string and numeric. So we have two types of variables -- string and numeric. A string variable has a dollar sign after the one or two characters. A numeric variable doesn't. String variables cannot be part of a math formula -- but a plus sign will string two or more strings together. A numeric variable cannot be strung onto a string -- but they can be part of a math formula.

The C-64 has the greatest programming interface ever put on an 8-bit computer. If you want to change a program line, all you have to do is put the line on the screen (with LIST), move your cursor up to it, type your changes onto the line, and press <RETURN>. To list one line, include the line number after the LIST command:

LIST 100

Other list possibilities include:

```
LIST -100   list everything up to line 100
LIST 100-   list line 100 and everything after
LIST 30-50  list all line from 30 to 50
```

As a list is scrolling, you can slow it down by pressing the <CTRL> key (<TAB> for VICE), or stop it by pressing <STOP> (<ESC> for VICE).

List line 100:
```
100 PRINT A1$ + A2$
```

and change it to
```
100 PRINT A1$ + ", " + A2$
```

Press <RETURN> then RUN the program.

Play around with this program for a while! Try all sorts fo things. You can print strings and numerics together on the same line:

```
100 PRINT A1$ + ", " + A2$; A1 + A2, A3
```

Use a semicolon to separate stuff you print. Use a comma to tab items to columns.

Try this:
```
90 T = A1 + A2 * A3
95 ? T;
```

and run the program. Or do this:
```
80 A1$ = A1$ + A1$
85 A2$ = A2$ + A1$
```

Try every combination of variables and print you can think of. It is YOUR program. And there is nothing you can do from the keyboard that will hurt a C-64! NOTHING!

ABOUT ERRORS

You have surely seen some errors by now. Two things about errors:

1. The program stops short at any error, and

2. The C-64 always says READY. after an error.

List the line and try to figure out what is wrong. SYNTAX means that something you typed is incomprehensible to the stupid machine. Fix it! Then try again.

[This is just the beginning! Do take time to play with all the concepts presented. Only with play will you learn how to apply the commands to your program.]

Interview with Mario Misic
The big book of Amiga Hardware

http://amigahardware.mariomisic.de/index_e.html

Q - Please introduce yourself to our readers

Dear readers, I'm a 39 years old and I work as the Managing director at a Tyre shop. Since the age of about 10 or 12 years old, i have used Commodore computers and have learned to love them.

Q - What was your first experience with Computing and especially the Commodore Brand?

My first experience with computers was in the Middle of the 80's and the first computer I played around with was a VC20 later I used a Commodore 64, before eventually using the Amiga. Unlike most people, I don't play games on these machines. From the start, i asked myself: "why has the Amiga (workbench)-window 4 corners and not six, eight or even more?" "Why are they rectangle and not round?" "Why must I use a machine in a certain way," "Why can't use the machine in a way, which I WANT?"

Q- What Commodore machines do you own and which of these are still actively used by yourself

Within my Commodore-time, I first used the VIC20, then later the C64, Then moving to a C128 before I came to the AMIGA. All machines have fascinated me for a long time. I learned about an absolutely new technology and I was very interested and keen to learn, all about how this new technology worked. Now, in the 21st century, no other computer has reached the so called "AMIGA feeling."

Q - Please tell our reader about the Big Book of Amiga Hardware project, what is its main aim

The Big Book of Amiga Hardware (BBoAH) was created by Ian Chapman over 8 Years ago. Over the years, Ian tried to find some Users, to host a mirror of the BBoAH. But no one had the power to keep a mirror running for long. A few years ago I set up a German mirror for the BBoAH and i was the first one to mirror the website, Additionally to Ian's main BBoAH I established a mirror and provide this "step by step" guide with an additional part in the German language The main aim of the BBoAH is, to support all interested people with information about every piece of hardware created for AMIGA computers. Without becoming a commercial project.

Q - How many people contribute to the project will you accept information from anyone

Over the last 9 years, hundreds of Users have contributed the BBoAH with different kinds of information about AMIGA related hardware. So many people have supported the BBoAH, that it's impossible to name them all here. Some people have sent us Images. Others sent us further information, documentation or special jumper settings, manuals, install introductions or driver disks. Without this support of the AMIGA community, the BBoAH, today, would not be the resource it is.

Q - So anyone can send information, if our reader

Thinks he has some value to add to the project how Would he contribute?

Everyone is welcome to send contributions to the BBoAH. Behind the BBoAH there is a small team of people, who will try to verify the contributions and do the updates on the various items of hardware. We try to give a response to each contributor - if his suggestion becomes part of our database. But don't be angry, when you don't receive a reply. Maintaining the BBoAH is a lot of work and time, more time than you can imagine, and sometimes one of the contributors could be forgotten.

Q - So there images of the products as well as text

We like to receive as much information as possible for each item of hardware. We try to collect all information about Amiga hardware. This could be: Images, hardware descriptions, Jumper settings, install disks and much more.

Q - Was it the projects intention to provide things like driver disks and installation software

The main intention of the BBoAH is, to provide the AMIGA Community with all information they would need. This will include as stated earlier driver disks, installation guides and manuals. Until just one year ago, you may remember, a lot of those files were provided by the BBoAH, for download, but For legal reasons, the download area of the BBoAH had to be closed. In the last year some copyright owners "attacked" the BBoAH and forced them to close the download area. However we continue to collect all related suggestions. But we can't name a date, when download area becomes live again.

The BBoAH Team is annoyed about copyright owners, who actually support the AMIGA platform in NO WAY, but "attack" the BBoAH with legal and copyright. The BBoAH was and is still a hobby project and we don't have plenty of money to pay lawyers to defend the BBoAH against legal issues. The Users of the BBoAH have no idea, how complicated the background of the BBoAH is.There are a lot of complications we have to consider. These things we try to keep away from our users.

Commodore Free *– Copyright yes I understand you're problems I have tread on this path myself*

Q - Do you have images of the boxes the items came Packaged in, is this something you would like to see for the project

At the moment, we don't collect images, of boxes or packages. Pictures of Boxes only have an individual background and do not really affect the Hardware information and hardware use.. The BBoAH won't be a museum. It will be a guide for those users, who use the AMIGA's day by day.

Q - What is documented, is it just make model serial Number or are there things like instructions and Installation guides etc
In past the BBoAH provide the hardware support with as much information as was possible. But since we've touched by legal reasons we've come together, to come closer to the will of the right owners. We don't want the BBoAH getting in danger and having to be closed because of legal reasons.

While Ian Chapman owns the BBoaH, the english law was considerably. After Ian has given the BBoAh into my hands, German law will take affect to the BBoAH. For this reason i made a relationship with the geman publisher "Amiga-Future". Amiga-Future is a German Software publisher and also a German AMIGA related print magazine, the magazine has been running for a long time. My intention in this way was, that a professional publisher with his Knowledge could help with the running of the BBoAH.

Q - How are the entries verified does someone look over them before they are placed on the website

Suggestions as they arrive will be verified by a lot of Amiga experienced users worldwide, before they become an official BBoAH-Update. The actual BBoAH team is supported by some well known german hardware developers who have been in the AMIGA market for many years. Because of this, the BBoAH team has a lot of knowledge, but cannot know all things. To ensure that BBoAH has correct entries. People are welcome to amend information, So the BBoAH never will be complete and everybody was pleased, to Change incorrect information.

Q - Is there a downloadable version of the project or is it just a web only, was the project intended to be a commercially printed book

Ian Chapman established the BBoaH As a HTML-based project at first for many years until the early 2006. The BBoAH until this date is available as part of the "kicktart Archives on DVD" In 2006 the BBoAH has changed it's structure from a HTML based to a database provided site, which will provide the BBoAH with a lot of more features. Actually this way provides the "Sort by Manufacturer" or the "Sort by connection" -feature. Also the multilingual BBoAH become available because of that. At the Moment, the BBoAH is available in english and german language. Because of the possibilities we try to provide the BBoAH in other languages, too. If someone would like us to provide the information in another language please contact us, your mail is welcome every time.

Q - Have you come across any problems while compiling the list

Very often we get mails with new/additional information. To keep them all up to date just in time alone is the biggest "problem" we have. A massive problem in the near past was, to convert the known HTML based BBoAH into a MySQL based structure. Ian Chapman solved this problems by himself, and most users would not have noticed, that the BBoAH "structure behind was changed".

Q - Is there any hardware you particularly want to know the details of

We've a lot of hardware we don't know anything about. For this reason, we've crated a category called "Mystery Corner" in which we introduce unknown hardware to collect information, until the mystery's solved by the community. In past, all our "Mysteries" have been solved by some of our readers.

Q - what do you think personally of the current state of Amiga, are things finally starting to look up again for the machine

The current AMIGA state is quite complicated.
It's not possible to say where the AMIGA will go in future. I have a dream that the AMIGA would again reach the status, that it had in the past. But I'm old enough to let this dream be a dream. Today, is another time and the technical side is another matter all together time moves on and a machine from The early 80's would need updating, for the AMIGA to have a real chance of a comeback.

Q - What in your personal view is an amiga, for example is it just the software (operating system)

In my mind, the AMIGA is NOT hardware and NOT software. AMIGA is more a feeling, which is much more hard to explain like the feeling of LOVE. Amiga was and is a feeling you have to live to experience. The Amiga is, for those people, who are in my generation, a machine which came out at the right time in the right place.

Q - when will the project finish, will this project run indefinitely, and how will you know when all hardware is documented

I don't think that the BBoAH ever will be finished.
There is so much very old hardware, which is not listed in our database, because no one has any information about this item. From time to time we get new info about interesting old hardware AND complete the database with the new info.

Also the AMIGA community worldwide develops more new and interesting hardware. These items are also listed in the database we need more information for them. For example I'd like to name the A500-Clockport-card for example. The idea for this item was born in a german Amiga community. It takes a lot of months to discuss this item and the users wrote down their wishes for the device. Also a lot of german AMIGA hardware developers are users of this forum. At last, one of the hardware developers produced the hardware for the community. As you can see, you can see no end of the BBoAH.

Q - What question would you have liked me to ask

As a german, I've heard nothing about your great magazine. I feel this is harming the community; we need to establish much more communication between AMIGA and Commodore users all over the world. Especially: I would like to ask what is the thinking of your British readers about the "German krauts"? Is this a justified prejudice?

That's a really silly question, i know. But it was a question I'm very interested in, because we German AMIGA'ns have fewer contacts with British AMIGA users than we would like to have.

Q - If you were given 1 million GBP pounds what would you spend the money on

If i have 1 million GBP, i'd like to spend them in the same way as i'll spend the money, which Ian has collected with BBoAH donations as I announced some month ago. If anyone had an absolutely new idea for any amiga related hardware, the BBoAH would like to be involved in the project, helping the idea. If the most of the People prefer that idea, the collected money will be transferred to this project.

Thank you very much for this interview and I wish you many success for your magazine in the future.

Kindest regards,
Mario Misic

http://amigahardware.mariomisic.de/index_e.html

Interview with Frantic Codebase 64

http://codebase.c64.org/

Q - Please introduce yourself to our readers

In the c64 demo scene I am known as (not necessarily well known) Frantic. Member of a demo group called Hack'n Trade and sole organiser of the C64 parties known as LCP (Little Computer People) from 1998 to 2005, and then together with other people, organiser of BFP (Big Floppy People) last year, and also this summer, in July. I live in southern Sweden in a city called Lund. I spend my days as a PhD student in Linguistics /cognition /interaction (working with gestures and such things), being with my wife and kid, and coding C64.

Q - What is your first experience of Commodore

I've had a C64 since I was a kid. I think I bought it 1987. Playing lots of turbotape games and so on. Later on I bought an Amiga 500 when that was hot and fresh, and it was first on the A500 I started creating my own things, like music and some simpler code.

Q - How did you learn to program

I didn't start coding "seriously" until later on when I moved from A500 to PC, and eventually got quite bored with the sterile feel of the PC computers. That's when I kind of turned back to the computers of my childhood with a new take on them. This time around I din't play games on them but rather all the other things that I never did back then and never really understood how to do until I got a little older.

Q - what advice would you give would be programmers

Take your time to think about what you are interested in REALLY. For example.. If you think about coding some RPG game for the C64. Are you just longing for that perfect RPG that no one else never did, or are you actually longing for travelling the road that leads to this game? That is, the actual coding part of it. If the answer is no to that, I think you better stay away from coding. Although you get a kick out of finished products of course I think the main drive behind coding necessarily needs to be fondness of the activity of sitting by your computer and just code.

The same thing applies to the process of becoming a coder. Again you can't just long for the "finished product", that is, being a skilled coder. You have to like the road leading there too. Beginning with simpler things and moving one step at a time. Reading some docs, some tutorial, trying out some assembler, and so on.

Q - From various users feedback, it seems actually getting the time to sit down and read about coding is difficult with real world (jobs, children etc) can you comment

Well, in one sense I don't this has much to do with me or with C64 coding per se, but of course people have "real lives" to. But you gotta decide what you want to do in life. Do you want to spend time coding c64 or not? Or do you rather focus on something else? (Little of everything = nothing done at all.) If you want to do that, it's up to you to make it happen by prioritizing and so on, and you can't blame the wife/work/kid for that.

Q - Do you still own any commodore machines

Yes.. 6-7 C64 (among them the original machine I got as a kid), my old A500 and a A2000. I also had an example of that rare C65 machine, but sold it since I rather needed the money than a rare machine. I am satisfied with the common ones... C64 primarily.

Q - Please tell us about the website and project "C64 Codebase" what do you hope to achive

The codebase is a wiki (a site users edit themselves, like wikipedia) on the topic of C64 coding. I think a place like this has been lacking before, and thus I decided to set it up when a discussion on the CSDb forum made it clear for me that a lot of other C64 coders also thought that was a good idea and actually felt inspired to add content to such a site. ...which they did too. I am quite satisfied to see that so many different people actually involved themselves to add some coding routines/articles already.

Of course there is the website called the The Fridge, which is very nice and a bit similar to the codebase wiki. There is also some forums and various coding articles spread all over in C= magazines and so on. All these forms of information structures have their pros and cons, but I think a wiki have some possibilities lacking before. For example it is a better way to store focused information, compared to forums which are nice and very useful but tend to spread information all over different threads/posts and interleave the relevant information with a lot of small talk. The wiki concept is also a bit more flexible than "static" magazine articles which are not improved upon after being published in case someone finds errors in it. Wikis got all this. Focused information spots, but still the dynamics of forums and other editable media. Also, the overall information structure on a wiki is editable, which makes it possible to make sure that it won't just grow into a mass of information where it is impossible to find what you look for. I do not have a clear goal with the site other than trying to keep it in shape. The content is supposed to come from all of us, rather than just from me. (Even though I of course encourage people to add things every now and then.) I will probably put up a forum there, so discussing specific articles/sources become possible. I think that is needed. It will not be a general coding forum though. I think the coding forum on CSDb is more suitable for that.

Q - How many people are working on the project

Just me, concerning the site itself. However, I guess something like 15 people have contributed contents so far. Also thanks to Icon for the web server hosting and to Slaygon for the URL.

Q - Is this similar to "the secret society of commodore coders website"

I guess there is some similarities, but obviously one of them is a forum and the other one is a wiki. Forums are needed too of course.

Q - Will this website teach people how to code

It will do whatever people make it do. So far there is no coding tutorial there from step A to Z, but on the other hand there is a good collection of selected links on the external links section that may prove valuable for beginners. As the wiki grows I think everyone will

find something of value there, beginner or expert...

Q - Is the website purely assembler and machine coderoutines or are Basic programmers welcome Basic

programmers are welcome too, even though I guess most serious programming on the C64 is actually done in machine code for obvious reasons.

Q - Do you think there is really a need for people to Learn Basic anymore

Not really.

Q - Some sample routines are available on the website are these copyrighted or can anyone use them

The contents of the site may be used in your own productions. People wouldn't add the contents there if it was secret or non-useable for others. The wiki concept is all about sharing.

Q - the items on the website have they been contributed by the authors or did you contact them or they contacted you

The wiki concept builds on users contributing material themselves, and that is what they have done. However, in some cases I have asked people for stuff, and in some cases I helped a little with the process of adding the material.

Q - Who can contribute, lets say our reader has something how would they contribute to the project

Everyone who feels they have something C64 coding related that at least one other person in the world may find interesting is very welcome to add it. I don't see it as a huge problem if people start adding a lot of contents not interesting for the broad masses. That is better handled by having a good structure on the site, than not adding the stuff at all. With a good structure it is easy to find what is there and to find what you look for anyway. So, don't hesitat. Go there and contribute! It may really be anything from small snippets to complete programs or articles about this and that.

Q - I notice most of the example are turbo assembler, is there a preferred assembler

Historically it is the most used assembler on the C64, but in these days more and more people use PC cross assemblers of course. This has had the effect that most turbo assembler sources work with minimal modifications in other assemblers, as a kind of prototypical middle point of asm syntax. (ACME use a ! instead of . as the beginning marker of directives such as .byte and .word, and KickAssembler uses // for comments instead of ; and so on. So the syntactic distance between ACME and KickAssembler is larger than the syntactic distance between Turbo Assembler and these two.) However, apart from that, I think that actually quite a bunch of the sources on the site ARE written in other assemblers like ACME, CA65, KickAssembler and others, so I am not sure I agree with you.

Q - What subjects are covered in the coding is the site designed for anything c64 related

Yes, for example it is not only intended to cover demo coding. Tools, games and other things are also welcome. But, as said before, the contents depend a lot on what people contribute, so it is open ended in that sense.

Q - Do you follow the Commodore demo scene

Yep I do, and as I said before I am the organizer of the previous LCP parties and one of the organizers of BFP, so I guess I am quite involved in the demo scene.

Q - Some coders want to hide there code only giving out various demos why share code with others?

I can understand this. It is part of the competitive aspect of the demoscene I guess, which is kinda fun. However, I think the kind of tricks and code that you find on codebase is more of the kind that is in the frontline of the VIC trickery, and thus most people wouldn't feel like it was "giving away" something secret to add this stuff there.

Q - From looking at some of the recent Commodore demos like the crest "krestage 3" it appears there is still a lot to learn about the machine would you like to comment

Yes, I guess there will always be more to do/find out. However, at the same time I think that most of the new tricks uncovered tend to be less useful for general purposes than tricks discovered longer ago. For example, opening the borders are a lot more useful than adding two extra grey pixels to a sprite as in Krestage 3, even though its a nice hack.

Q - Will Codebase 64 look into some of the various VICbugs, that are used in new demos

Once again.. It is up to the users what information that will end up on the wiki. In case people are interested in it and want to share/discuss it, I guess it will appear there.

Q - Do you think anyone can write computer code,

Most should be able to learn at least a bit.

Q - So what next for the website more code snippets and hints or are other changes due to happen

Hopefully the contents of the site will continue to grow in the same speed as it did until now, and I'll add a forum there too, so people can discuss specific articles/sources there. Ask questions or discuss them in general.

Q - Some reader may be put of by websites that are best viewed with this and that browser can you commenton codebase website design

Basically it is just the default look of the DokuWiki engine, slightly modified colorwise and a logo was added. It should work good in all mozilla browsers and also in internet explorer, and I guess that covers 99% of the internet users. Although the wiki has some layers and such things it is mostly text based contentwise, so I think it should look the same in all browsers as it for all practical purposes.

Q - final 2 questions, if you were given an unlimited amount of time and money what would you create, this can be none computer related

I think I would just take it quite easy. :) Continue doing roughly what I do, but in a improved way. ;) But I dunno. It's hard to say really. I guess a lot of ideas would pop up if things were as you say. Maybe I would save the world!

Q - If you won 1 million Uk pounds what would you spend the money on and why

I would not buy a lot of things. I would start working half time and such things instead. That is quality for me. I already got most of what I need I think.A dishwasher would be nice. I don't have that. ;)

Q – Have you any comments you would like to add

Thanks to everyone who contributed to the codebase so far, and to the rest of the world, go there you too and add what you've got lying around hidden on your harddrives so that others can benefit from it.

HEX files Part 5

By Jason Kelk
www.Oldschool-Gaming.com

Righto, time for some more code writing entertainment! Now, where were we?

Ah yes, the challenge from the previous instalment; your mission, if you decided to accept it was to change the text colour to light blue and the sprite colour to dark blue. The text colour is altered by changing the LDA #$0F on line 25 of the source to LDA #$0E or indeed any colour you want. To change the sprites look at line 62. It reads LDA #$0C and to make them purple use LDA #$04. All sorted now? Good, lets get back to dissecting the remaining code, starting from line 71;

```
main        lda #$fc
rashold     cmp $d012
            bne rashold
```

Now our friend the raster has been introduced to us a couple of installments back, and the principle remains the same; the C64 projects the screen to the TV a line at a time and this loop waits for line $FC, which is just off the bottom of the standard screen. From here onwards the term "frame" will denote one complete scan of the screen by the raster, so one frame is a 50th of a second.

```
            ldx scrlcount
            inx
            stx scrlcount
            cpx #$08
            bne dontmove
```

Right, we've already discussed using labels as places to keep numbers. scrlcount is our smooth scroll counter, it's counting how many frames have passed up to a maximum of eight to delay our scroller since moving it every frame would be far too fast to read. It's also used to work out what value should be in $D016 later on, but I'll explain that when we get there.

```
            ldx #$00
scrlmove  lda $05e1,x
            sta $05e0,x
            inx
            cpx #$27
            bne scrlmove
            ldx messcount
            lda scrolltext,x
            sta $0607
            inx
            stx messcount
```

Well, this section has covered before too and it's basically the same scroll shifting routine we had moving so quickly previously. This time, however, we are reading from our own message in the computers memory at wherever the label scrolltext is. That gets defined later, don't worry.

```
            ldx #$00
            stx scrlcount
```

Because we've just finished a "move" of the scroller we now have to reset our counter so that the machine will wait another eight frames before doing it again. Okay, now earlier we branched to dontmove if we were not going to move the scroll, so lets see where that leads us.

```
dontmove txa
            eor #$07
            sta $d016
```

Okay, this is just a little bit of trickery to set the smooth scrolling up for our message. TXA we have already covered, but EOR is new. EOR (or Exclusive OR) is what is known as a logic gate. There are three regularly used gates, ORA (known as just an OR gate, but called ORA because all 6510 commands are three letters long, so we add an A for accumulator), EOR and AND. ORA works like this:

```
            LDA #$64
or in binary %01100100

            ORA #$C7
or in binary %11000111

            Leaves A as #$E7
or in binary %11100111
```

Why? Well, look at the first column of binary numbers. The first two are a 0 in the top row and a 1 in the second and the "rule" is that if either the first OR the second number (or indeed both) is a 1 the answer, in the third number in the column will also be 1. If both numbers are a 0 the answer will be a 0, just like the fourth and fifth columns.

AND works on a similar principle, except that the first number AND the second number must be a 1 for the outcome to be a 1, otherwise the answer is 0. If only one of the numbers is a 1 then the answer is a 0, meaning that if the ORA above is replaced with an AND the binary answer will be %01000100.

Now for our little friend EOR. The result of our above example after using EOR instead of ORA would be %10100011 because EOR works by a different means. Again, it compares the columns separately, but if the second number is a 1 it takes the first number and toggles it, changes it from a 0 to a 1 or vice versa. If the content of the second number is a 0 nothing changes, and the content of the first passes straight through. So the result of EOR #$07 on our counter, which runs from $00 to $07 is to basically invert it so that it goes from $07 to $00 and we can then put this into $D016 for the smooth scrolling.

Okay, so now we've sorted out the scrolling and the smooth scroll, it's time for the sprite positions.

```
            ldx #$00
            ldy sineposx
setsprx     lda sine_curve,y
            sta $d000,x
            tya
            clc
            adc #$20
            tay
            inx
            inx
            cpx #$10
```

```
                bne setsprx
```

Right, this is a fairly normal copy loop (as we've done before) but with an unusual twist. Before I explain how it works, I'll just quickly explain sine and cosine curves a bit more, which are used when demo programmers want to make things swing back and forth. At $0E00 and $0F00 are two tables of data, each 256 ($100) bytes long. If you were to examine the data you'd find that it contains the co-ordinates for a sprite, starting from the right hand side of the area it covers, accelerating towards the middle and decelerating at the left hand side of the area, then moving back in the same way.

The above loop reads the curve at $0E00 using the Y (which contains another counter) and puts it into $D000. It then adds a value of $20 to the Y, increments X twice (so that it only writes to the sprite X values, $D000, $D002 and so on) and then, if it hasn't reached $10 and therefore done all eight sprites goes back.

Why do we add $20? Well, it's to make all six sprites read from different points on the curve to produce that swirling effect. Try changing that value to $00 and they all appear with the same X position.

```
                ldx #$00
                ldy sineposy
setspry         lda sine_curve_2,y
                sta $d001,x
                tya
                clc
                adc #$1c
                tay
                inx
                inx
                cpx #$10
                bne setspry
```

This is basically the same loop as we've just looked at, except that we are using sineposy, reading from $0F00 (the vertical sine table, slightly different to the horizontal one) and writing the values to $D001, $D003 and so forth for the vertical sprite positions. The ADC #$1C is different to the previous loop merely to make the curves a bit more interesting. It can quite happily be $20 or indeed any other value, so why not try playing with it and its fellow to see what you can do?

```
                lda sineposx
                sec
                sbc #$03
                sta sineposx

                lda sineposy
                sec
                sbc #$02
                sta sineposy
```

Okay, this just moves the counters around for the sprite movement. Again, the $03 and $02 can be altered to change how fast the sprites swing around, a value of $00 will cause them to stop dead and a value of $FF will make them go backwards on one axis slowly.

```
                jsr $1003
                jmp main
```

As explained last issue we are using an external music routine. Every frame we have to call the routine to keep the music playing and here's the JSR call for the tune we're using. And the JMP completes our loop, as we go back to main and wait for raster position $FC on the next frame.

```
                * = $2000
message         .scrl "welcome to the scrolling message!    "
                .scrl "this little demo-ette was coded for   "
                ...
```

And to finish off this block of code, here's the previously mentioned label message and another new assembler directive to go with it. The .scrl command simply writes the text in quotes to the memory (at $2000, as specified by the * command above) in a format our routine can use, the letter A is represented as $01, B is $02 and so on. The text included in the source (of which only a part is reproduced here) is exactly 256 bytes long because the scroll routine can't handle any more or indeed less.

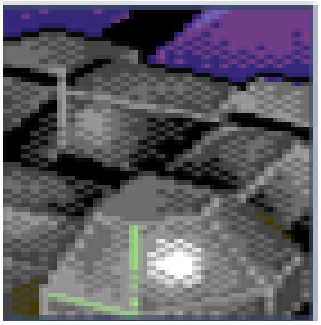
Right, all done and this time I don't have a challenge for you, just some "play" suggestions. The values in the two sprite setup loops ($20 and $1C) and the mover ($03 and $02) can be altered.

Why not try putting different values in and seeing what happens to the sprites. Some interesting ones to try are replacing $20 with $82, which causes each sprite in turn to read its position from opposite ends of the curve, or replacing $03 and $02 with $04 and $05. Go on, have a play with the numbers and I'll see you next time with something new to look at! As always, email me if you have any queries, comments or suggestions.

The source code for the routines above (they're the same as Hex Files 4) can be downloaded here

www.Oldschool-Gaming.com/files/c64/hex_files/part_6_files.zip

www.Oldschool-Gaming.com



“Potrace” Open source tracing application Frequently Asked Questions

My first name for this program was "pbm2eps", which was very lame and also inaccurate, because both the input and output file format might evolve over time. So I started looking for a cooler name.

Since this program does tracing, I wanted a name with the word "trace" in it. I did a Google seach for each combination "atrace"..."ztrace" of a single letter plus "trace". Believe it or check for yourself: every single one of them was already taken by some other software!

Most of them were something boring, like the name of a function in some obscure statistics package written in Fortran. If I remember correctly, the only exception was "ytrace", which brought up a foreign language porn site called "sexo con perros".

I am not sure what "ytrace" might mean in some foreign language, but I have a pretty good idea what "sexo con perros" means, and so I didn't want to take any risks giving this name to my program.

So I started looking for 2-letter combinations. It so happens that "potrace" didn't yield any hits on Google, except a few sites that had misspelled the word "podrace" (as in pod-racing) from Star Wars Episode I.

This was okay, I thought, and would not attract any trademark infringement suits. So I settled for the name Potrace. I particularly like this name because it can also mean "polygon-based tracer", which actually describes quite accurately what Potrace does.

By the way, it is pronounced "po-trace", not "pot-race". If it conjurs up mental images of people running around a track with cooking pots, then you are not pronouncing it correctly.

Question: Is there a mailing list? How can I be notified of new releases?

Answer: There is currently no Potrace mailing list. However, Potrace has a project page at Sourceforge.net. This page provides a discussion forum, as well as a mechanism for reporting bugs. To be automatically notified of new Potrace releases, go to the Potrace project page on Sourceforge.net, find the section on "Latest File Releases" and click on the little envelope icon under "Notes/Monitor".

Basic usage and errors

Question: Can you show me the options used to generate the examples pictures on the Potrace website? I tried to do it myself, but failed.

Answer: All the examples were generated with the default settings. The EPS files were generated by the following command:

```
cat file.pbm | potrace > file.eps
```

but the following also works:

```
potrace file.pbm
```

The input to Potrace in each of these examples was a PBM file. But since most browsers can't display PBM, I posted the images in PNG format on the website, and I also scaled them. You need to shift-click on the left-hand side image to download each original PBM file.

The greymaps were generated by this command:

```
cat file.pbm | potrace -gx2
```

This generates a PGM file. Again, for posting on the web, I have further translated this to the PNG format by piping the output through pnmtopng.
For experts only: If you want to know exactly how I processed these files, you can look at the Makefile that I used.

Question: I installed and ran Potrace, and it did not work. I got the error message: sh: line 1: compress: command not found. What am I doing wrong?

Answer: This error occurs only with Potrace version 1.3 or earlier. When this happens, you should either install the newest version of Potrace, or else run your existing version with the -c option, and everything should work fine.

Question: How can I improve the quality of the output of Potrace?

Answer: The most effective way to do so is to improve the quality of the input! This seems obvious, but not everybody thinks of this. Scan your image at a higher resolution. Scan your image in greyscale, and use a program such as mkbitmap to generate a high-resolution bitmap to trace.

In some cases, it also helps to decrease the resolution of the input image. If the resolution of your input file is incredibly high, the output generated by Potrace might be larger than necessary. This also sometimes happens if the input is speckly or noisy. Rather than trying to simplify the vector image, it often helps to just downscale the input bitmap.

Question: How can I reduce the number of nodes in the output of Potrace?

Answer: Potrace already makes an attempt to reduce the number of nodes, and this behavior is controlled by the --opttolerance parameter (or can be turned off entirely with --longcurve). However, I have decided not to sacrifice quality for quantity, and Potrace will not reduce the number of nodes beyond what it considers tolerable.

For many applications, the problem of reducing output file size can be solved by reducing the resolution of the input image. This sounds too simple to be true, but can give surprisingly good results. One of the most common "beginner's mistakes" when using Potrace is to start with an image that contains too much detail or noise. Blurring the input image can also help, particularly with grey-scale images.

3. File formats

Question: Why can Potrace not read/write PNG files? It is such a popular file format.

Answer: In keeping with the old unix philosophy "write programs that do only one thing and do it well", I am trying to avoid writing a front end for every possible graphics file format known to mankind. Instead, I provide support for a small number of generic formats which are easy to convert to and from.

The PNG format is relatively complex to implement, and there already exist many excellent programs for converting between PNG and the PNM format which Potrace can read. See the next question.

Question: What programs can read/write PBM files on Linux? Gimp does not seem to help me... How can I convert PNG to the formats that Potrace can read?

Answer: Gimp, unfortunately, cannot write PBM files (although it can read them). However, Gimp can write PNM files, as well as BMP files. Both file formats are understood by Potrace version 1.1 or later. Note that the files have to be bitmaps, i.e., they should only use two colors black and white. All other colors or grey values will be converted to black and white before Potrace begins its processing.

A good program for converting anything to PBM is the convert command line utility of ImageMagick. It is invoked simply as "convert inputfile outputfile". It is pre-installed on many systems, and can otherwise be obtained from www.imagemagick.org. The output format is determined by the filename extension, so for converting to pbm, you simply type:

convert image.png image.pbm

Another very useful such set of command line utilities for converting different image formats are the "netpbm" utilities, see netpbm.sourceforge.net. Again, these tools are preinstalled on many popular systems. They allow you to convert between file formats on the command line. Particularly useful are pngtopnm and pnmtopng, but you can also convert to and from other formats using e.g. giftopnm, anytopnm and so forth. You can use these commands in a pipeline, for instance

cat file.png | pngtopnm | potrace > file.eps

Question: I am working to add Potrace support to another program, but the output of Potrace leaves me a little stumped. How can I parse the output of Potrace into some format which my program can use?

Answer: To get parseable output from Potrace, I suggest using the -q and -c options. The default output is optimized for size, not readability. With 'potrace -qc', you should get exactly what you need.

If you use the output from Potrace for processing by another program, I also recommend increasing the output resolution, e.g., -u=100 or -u=1000, to reduce the effect of roundoff errors. Note that the output coordinates correspond to input (bitmap) coordinates, multiplied by the unit length, then rounded to integers.

4. Color

Question: Can Potrace handle color images?

Answer: The short answer is "no". Potrace can only handle 2-valued images at the moment. It does not matter whether the two colors are called "black" and "white" or "on" and "off" - however, there can be only two of them.

Question: Will color support be added to Potrace in the future?

Answer: Maybe.

Question: How can I work around the lack of color support?

Answer: There are many ways in which Potrace can be useful in processing color images, with some extra work. For example, you can trace an image to SVG format using the --svg and --opaque options, and then use e.g. Sodipodi or Inkscape to color it manually.

Or you can extract individual color components from your image using the Gimp or ppmcolormask (part of the netpbm package), trace them separately, and then overlay the pieces to get a multicolored image. You can get pretty good results for posterized images. I have used a command line similar to this:

cat img.gif | giftopnm | ppmcolormask #641b1b | potrace

Recent versions of Inkscape have a built-in Potrace engine that can handle color images via color quantization or multiple scanning, thanks to the great work of Bob Jamison and the Inkscape team.

Another interesting application of Potrace to color images is described in the fascinating article Automatic Generation of Stained Glass from Scanned Photos by C. Scott Ananian.

Graphical user interface

Question: I don't like command line utilities. Is there a graphical user interface for Potrace?

Answer: Yes, there are now several graphical user interfaces (GUI's) for Potrace, contributed by various other people. There are GUI's for Linux and for Mac OSX. I have not tried them myself, so I cannot say if they work or if they have all the features you want. I suggest to try them out. Please look under GUI's and related software on the main Potrace web site.

BEFORE

AFTER

Interview with Peter Selinger
Potrace creator

http://potrace.sourceforge.net

Q - can you please introduce yourself to our reader

My name is Peter Selinger. I am a professor of mathematics at Dalhousie University in Halifax, Canada. I develop open-source software in my free time.

Q - How did you get involved with Commodore machines and do you still actively use Commodore machines

My brother had a C-64 when I was 12 or 13 years old. I spent the next few years programming it in Basic and assembler. It was the first and only Commodore machine I ever used.

Q - Can you tell our reader about Potrace, what it does and how it differs from similar applications?

Potrace is a utility for raster-to-vector conversion. This means, it inputs a bitmapped black-and-white image, such as you would get from a scanner or digital camera, and turns it into a scalable vector image using Bezier curves. The output can be further processed with a vector graphics editor such as Inkscape.

There is other software with similar functionality. Most of it is commercial. In my opinion, none of it gives output as nice looking as Potrace's.

Q - What machines and operating systems does the application run on

Potrace was written with portability in mind, so it can run on virtually any system that has a C compiler and some POSIX-like library functions. It was developed on Linux, and runs on every flavour of Unix, Windows, and the Macintosh.

Alfred Faust ported it to the Amiga (OS4) and Matthias Rustler to AROS, the Amiga Research Operating System (which runs on Intel hardware).

Q - Any possibility of a GEOS or Wheels version for Commodore 64 owners

The source code is available; with a C compiler and some patience, I don't see why someone could not compile it for the Commodore 64. Of course, some expanded RAM will be necessary, as the Potrace source code runs to about 260 kilobytes, not including shared libraries.

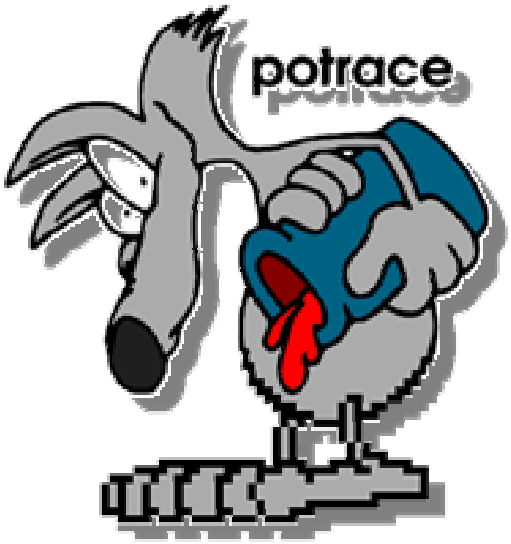
Q - I notice the software is Command line only but other users have added GUI versions, was your intention to produce just a Command line version

Yes. Potrace adheres to the old Unix philosophy of "doing only one thing, and doing it well". This is why I call it a "utility" and not an "application". It has no fancy user interface or add-on functionality. It can be used in batch mode or scripts. Potrace also can only read a handful of image formats. There are other, much better programs for converting one image format to another.

Q - Can you tell our reader who contributed to the project and what they did

Potrace was almost entirely written by myself. Tor Andersson wrote a backend for outputting PDF. Karol Krenski contributed the logo, which he of course created using Potrace. Many people have helped with finding and reporting bugs.

Q - Our reader may not be aware of the "GNU General Public License". Can you explain this does it mean the software is free to use.



For a full explanation of the license, read the file COPYING that is distributed with Potrace. The short answer is: yes, under the GNU General Public License (GPL), you are free to use the software.

However, much more is true: you are also free to modify, recompile, and redistribute it. The primary restriction is that when you redistribute (modified or unmodified) versions of Potrace, you must again do so under the terms of the GPL. This means that nobody can integrate Potrace, or anything derived from it, into a non-GPL program without my permission.

Please note that "free" in this context refers to freedom, not to price. There is lots of software that is offered without payment, but that does not give you any freedom. GPL software is not like that.

Q - What other software have you written, and are you working on any other projects at the moment

I have written lots of software. The most popular items besides Potrace are:

- Ccrypt, a simple and secure command-line encryption program (ccrypt.sourceforge.net),
- Upprint, a printer frontend for n-up and double sided printing. N-up printing means putting multiple pages

on a single sheet of paper at reduced size. Most off-the-shelf software does this very badly, by leaving huge margins and making the text way too small.

Q - I like the Potrace mascot can you tell our reader a little about it.

The mascot was designed by Karol Krenski, a Polish artist who was an early and enthusiastic user of Potrace. You can see some more of his works at

www.sgsp.edu.pl/inne/galerie/krenski/krenski.php

Q - what problems did you have creating the software, was all the code created by yourself or did you manage to find something similar and adapt it to your needs

The largest challenge in writing this software was designing the Potrace tracing algorithm. This is really the heart of the software. I wrote all of it myself, except for the PDF backend.

Q - I see this is a FAQ already but for our reader who desnt read the FAQ`s - Can Potrace scan and convert Colour files as well as mono images, are there plans for the software to convert colour images

No, it cannot convert colour images. Perhaps I will add this ability in the future, but I do not currently have a concrete plan to do so. There is some software, such as Inkscape (www.inkscape.org), which is able to use Potrace on color images, essentially by decomposing the image into a series of bitonal images, tracing each of them separately, and then putting the results together again.

Q - Does the software work equally well on text or complex images

The best results are obtained on handwriting and hand-drawn images such as cartoons. Potrace also works reasonably well on text, provided that the characters have been rendered at a high enough resolution. Complex images are no problem, because Potrace is quite fast even on most large images. One thing that Potrace does not work too well on is noisy images. It is important to prepare the input image carefully to get best tracing results. I wrote another program called "mkbitmap" that can help doing this. It is distributed together with Potrace.

Q - What is the current version of the software and are there further development charges planned

The current version is 1.8. Potrace is quite stable at the moment, which means it works reliably and there is no need to make lots of changes to the software.

Q - How can our reader help with the project

Use Potrace and tell others about it!

Q - is there a forum for people with problems using the software, can users email you directly

For problems, users should first check the Frequently Asked Questions

http://potrace.sourceforge.net/faq.html

and then use the Forum and

http://sourceforge.net/projects/potrace

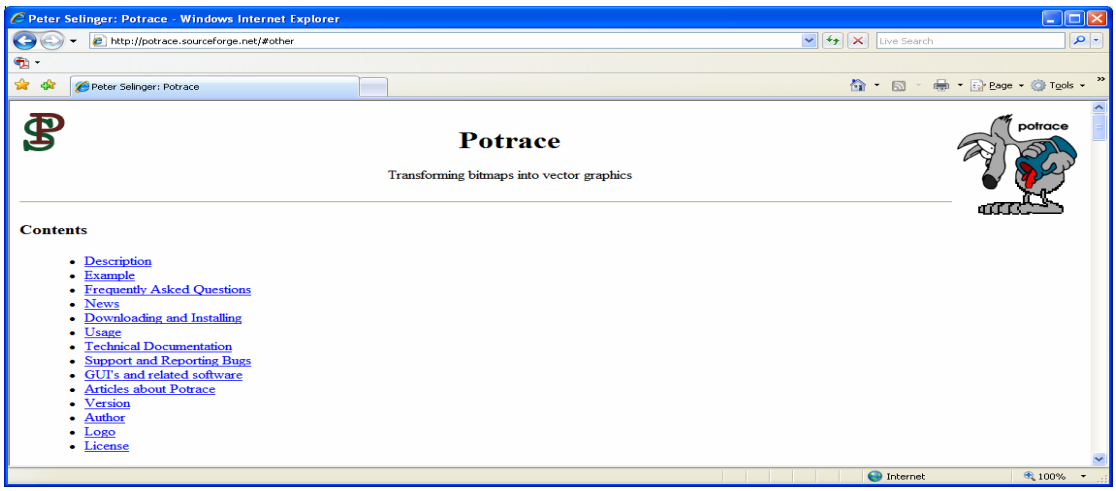
Q - what are the main problems converting the software to other platforms, can you give a quick how to on conversion to our reader, is there any help on converting lets say an Amiga application to linux or even the PC,

My main advice is to write the software portably in the first place. Fortunately for me, within the Unix community, there are some well-established standards for the programming environment that any Unix system should provide. One such set of standards is known as POSIX. Since POSIX environments are also available for Windows and the Macintosh, porting to those platforms was for the most part not very difficult.

Q - I presume that command line versions are quick to convert to other platforms is this why you only personally produced the command line version of Potrace

No, portability was not a consideration in my decision to produce a command-line tool. Actually, I use the command line a lot. If I need to convert 200 images, I would much rather write something like "potrace *.pnm" than going through some graphical user interface where I have to manually click, drag, and drop 200 files. Tracing of images is not fundamentally an interactive process. Pretty much there is an input and an output, and a few parameters that one can tweak along the way. So I never saw the need for a graphical interface. However, others have written such interfaces for Potrace.

See http://potrace.sourceforge.net/#other



Interview with Steve Judd

SLANG 6502 Programming Language
http://www.ffd2.com/fridge/slang/

Q - What first introduced you to Computing and especially commodore

I bought a C64+1541 in 1984 -- in eighth grade -- for $400 of saved up paper route money. I wanted to be a programmer but I wasn't very good at it; all I ever did were a few BASIC programs and eventually a few simple assembly programs that I never got to work. That left plenty of time for playing games, though. I stopped using it around 1988. In 1993 I bought a 128D, for old time's sake and to play games on, and wound up getting interested in 3D programs. This time however assembly language made sense, and it became possible to make stuff work and start having loads of fun.

I probably became interested in computers via video games and computers in stores like Radio Shack; there were also computers at school, Radio Shack Model IIIs. Like a lot of other kids my age, I can say that the C64 is what got me into the work I do today.

Q - Please tell our reader a little about slang what it is and a general History of the project

Slang is a programming language for the C64, and other 6502 computers. It's easy to learn and use, and compiles to pretty decent machine language for speed. It's meant to be used by everyone, from programmers just starting out who want to move beyond BASIC, to high-power assembly programmers who want to take some of the tedium away (writing a game engine, for example). Another hope was that it would encourage more C64 programs to be written, by greatly reducing the development time and effort required.

The project started around Jan 2004, and took some 1-2 years of pretty constant development. The code was developed in modules, for example the parser module, evaluating mathematical expressions, subroutines, etc. The code is all built on top of Sirius, an assembler that I wrote; that is, the compiler more or less translates code into assembly language, using the assembler to generate the code. (One result of this is that assembly language is built-in to Slang.) It looks like the very first beta version was released in Jan 2005, after a year of work. There were a lot of bugs, and missing features, so the program was continually updated for at least a year after that.

The code was written in assembly language, on a SCPU-equipped 128D, using the Sirius assembler. There is something like a half-megabyte of source code, which is assembled to a file that is over 40k large, i.e. over 200 blocks of code (i.e. no graphics or music).

Q - was the software solely produced by yourself

Yes, but a number of people made great suggestions, found bugs, and so forth. Jaymz Julian, Majic Eyric, and Christian Lott come to mind, and I apologize for others I've surely forgotten.

Q - why was the software produced what prompted the project

Hard to remember now, but... the main issue was to make it easier to write C64 programs. There are lots of assembly programmers who never write games, because game engines are such a pain in assembly. There are many more unfinished projects, because the development time is so long and involved. At the other end of the spectrum, there were motivated programmers who had a handle on BASIC but found assembly too difficult.

The idea then was to create a language with a nice C64 vibe, which would be fun to use, easy enough to satisfy a BASIC programmer, and powerful enough to take on the most advanced programming tasks, making program development practical.

Q - you must still be actively use commodore machines, what machines do you own and still uses (you can list none commodore machines)

I'm afraid the 128 is in the closet these days. All we have at home is a Mac which I use for very mundane things like email.

Q - must the software be run on a SCPU equipped Commodore machine

The cross-development version, xlang, runs on Linux or Windows machines, and would probably run on a Mac if someone compiled it. Xlang actually emulates portions of the SCPU/C64 and runs the 65816 binary, as opposed to being a duplicate program.

To develop on a real C= machine, however, a SuperCPU with plenty of memory is required.

Compiled Slang programs run on a plain C64, and do not require a SCPU.

Q - so is slang a new highlevel language like BASIC or is it more a low level language like machine code

Both, actually. That is, it may be helpful to think of a set of layers. At the top layer, Slang is a high level language like BASIC, C, etc., and this is all a user ever has to see to use Slang effectively. All the usual high level features are present (variables, subroutines, logic control, etc.), and there are numerous C64-specific commands, such as commands to control sprites and interrupts.

More advanced programmers, however, can take advantage of lower layers, getting closer to the hardware, writing new commands, and so forth. At the very lowest layer, Slang recognizes assembly commands, which may be mixed with regular Slang commands and variables, for the ultimate in flexibility. The web page contains example programs at these various levels.

It should also be mentioned that Slang can communicate with outside ML programs very simply. For example, many years ago I wrote a set of ML routines for doing 2D graphics, such as lines and circles, to be called by other ML programs. With just a few lines of code I can now call all those routines as if they were ordinary Slang commands. This means that programmers don't have to re-write code

that they've developed over the years, but can just plug it in.

Q - How would our reader start using the application, Does it need "installing" or is it just a case of copying a prg to a disk

Most people would probably just download xlang, unzip the file, and be ready to go under Windows (DOS) or Unix.

Q - the end code can it be run on none SCPU equipped machines or will the code only run on Commodore 64`swith a SCPU

The compiled code does not require a SuperCPU to run. After all, the goal is to get more useful programs written, which means run on a stock C64.

Q - Also is the code only Commodore 64 compatible or could you work on any 6502 processor machine

Most of the code is plain 6502; only a few things, such as printing, or the C64-specific commands such as sprites, require a C64. Thus code may be written for 6502 computers other than the C64.Moreover, because of how the language is designed, it is actually possible to create completely new commands which are specific to other computers. That is, there could be an Atari-Slang, an Apple-Slang, and so forth. Nobody has been interested in this yet, though.

Q - could you give an example of a simple program in basic and then in slang so our reader gets an idea of the syntax - although I will include the tutorial and quick reference from the website (if this is ok with yourself)

(fine by me to include anything -- everything on the web site is freely redistributable)

Here is a short Slang routine, from the example file grdemo.c.s. It uses the grlib graphics routines to plot a sine wave:

```
x1=0
for x=0:60 step .1
  y1 = 100+50.0*sin(x)
  GrPlotAbs(x1,y1)
  x1=x1+1
next
```

As you can see, the syntax is pretty straightforward and should feel familiar to anyone who has written a BASIC, C, or other high level language program.As a second example, here is some code from the example program spritedemo.e.s. It sets up a raster IRQ -- when raster line 50 is reached, the subroutine "Scroll1" will be called:

```
;
; Init scroll IRQ
;
DisableTimerAIRQ      ;shut down CIA timer IRQ
(kernal IRQ)
SetRaster(50)
SetIRQRoutine(Scroll1)
SetCol38              ;use 38 cols to make it smooth
EnableRasterIRQ
...
irq Scroll1()
;irq routine code goes here
```

While this is a fairly simple example, it demonstrates some C64-specific features of the language, and how easy it is to get up and running quickly.

Q - The slang language uses libraries to compile the final code is this because the use of libraries is more efficient, can you explain libraries to our reader and the advantages

Speaking very broadly, a library is a collection of code routines. These are typically commonly used routines -- the "print" routine is an example. One advantage of using libraries is that they usually make code smaller: instead of duplicating a routine many times, a single subroutine is called each time.

A second advantage is that it's easy to update or modify routines: instead of updating the entire compiler, I can simply update the library routine. Moreover, it now becomes possible for others to write their own collections of routines, and share them with others. Another, more technical reason, is that this is one of the ways Slang can be used to write programs for other computers: by writing an "Apple II" library, and substituting it for the "C64" library, it becomes possible to compile programs to other computers fairly easily.There is a more advanced feature, using the linker to create libraries, which are much closer to being "true" libraries, but it's more complicated and not worth going into here.

Q - could our reader write his own library for use with slang, and is there a page to download more libraries for use with slang

Certainly. Anyone can create libraries, or use libraries already created. Every library that I am aware of is available on the Slang homepage.

Q - Where next then for the language? do you consider Slang to be stable so is it just a case of optimisation and adding more features

Slang is pretty stable, and I haven't developed it for some time. The major problem has been getting people to use Slang. To date, I don't know of any finished programs written in Slang, aside from the example programs on the web page. Jaymz Julian had written a pretty substantial program, a Wizard of Wor clone, but it didn't get finished. (Life responsibilities probably took over.) Heck, I wasn't able to get even my friends to try Slang, or the people who asked for it in the first place, and I've received feedback from just two or three people. There just doesn't seem to be much interest, for whatever reason. So, I moved on to other things.

I do still support Slang though, and I'm happy to help out anyone with questions.

Q - Is SLANG free to download,

Yes, Slang is free to download, and programs written in Slang are yours feel free to sell them, or give them away, or whatever, no different than using an assembler.

Q - What support exists for users who need more help

There is an online forum, and a mailing list, and you can always email me directly.

> Q - Thanks for your time

Thank you!

Slang Quick Reference guide

Numbers and misc

```
;                       ;comment
*                       ;comment when in 1st column

a=1234                  ;base 10
a=$1234                 ;base 16
a=%11010                ;base 2

a=a + &                 ;line continuation
  b
a$="Howdy"              ;strings -- " or ' allowed delimeters
a$=!5"hola"!13          ;like chr$(5)+"hola"+chr$(13)
a$=!s"hey"              ;!s -- use screen codes instead of
petscii
                        ;!r –  use regular (petscii) codes
                        ;!n – negate (ora $80) following codes
                        ;!z – do not null-terminate string
```

Slang is CaSe InSeNsItIvE

Variable types and declarations

```
byte, ubyte             ;1 byte, signed/unsigned
int, uint               ;2 bytes, signed/unsigned
float                   ;5 bytes, MFLPT format

byte blah@$d020         ;@-var: locate variable at $d020

int screen(25,40)@$0400     ;Create 2D array, locate at text
screen
int blah(3,3)=[ [1 2 3] &
                [4 5 6] &
                [7 8 9] ]   ;Create 2D array and pre-initialize
values
ubyte str(50)=['..can also use strings to pre-initialize']

blah(2,0) = 1           ;Note that array indices start at 0

byte ^test              ;Create 16-bit pointer to a byte
byte ^^test2(20,10)     ;24-bit pointer to byte array
test=$c000              ;set point location: lda #$c000  sta zp
^test=10                ;set location value: lda #10  sta (zp),y
test2=$123456           ;set array base address
test2(3,5)=10           ;leading ^ not used with arrays

deftype foo ;Compound variable type declaration
  int .a                ;note leading .
  byte .b(10)
  float .x
defend

type foo yak1, yak2     ;Create two variables of compound
type foo
yak1.a = 10             ;...which can then access individual
elements
yak.b(3) = -8
yak1.x = 3.1415926

VarBlock @$c000         ;Define following variables starting at
$c000
  ubyte t1
  int t2
EndVarBlock             ;until the EndVarBlock

int b,c
c=#b+3                  ;Set c to the _address_ of b plus
three
```

Strings

```
byte str(20);Strings are byte arrays, null-terminated
str(1) = 1              ;Treat as byte array
str$="Hey!!"!13         ;Using the $ sign treats as string

if str$(2:4) = "y!!"    ;Can specify substring ranges
  str$(2)="blah"        ;|H|e|b|l|a|h|00
```

Subroutines

note: <- indicates backarrow key; use _ (underscore) on PC/xlang

```
sub blah()              ;Subroutine with no input or output
variables
sub blah2(int x, byte r)    ;Subroutine with two input variables
sub blah3(int z)<-byte s,int t  ;Subroutine with one input
variable (z)
                            ;and two output variables (s, t)

sub blah4(@ax)          ;One input variable, in the .A (lo) and
.X (hi)
                        ;registers. @x @a @xy @yx ->
@(lo)(hi)
endsub                      ;End subroutine (rts)

sub asm chrout@$ffd2(@a)    ;Create subroutine
interface                       ;(for calling
external routines)
Usage:

blah3(10)               ;Call subroutine on its own
a=blah3<-s ;Use subroutine variables in expressions
b=blah3<-t + a + 1
a=blah3(12)<-s          ;can also combine operations
```

Operators

Note: comparison operators can be used with strings

```
=, +=, -=, *=, /=       ;Assignment operator, e.g. a=1, c+=3
++, --                  ;Increment/decrement, e.g. blah++

bitand, bitor, biteor   ;bitwise operators
+                       ;addition, string concatenation
- * / ^ %               ;standard math operators.  Note: int *
and /
                        ;require core library; floats require
BASIC
<< >>                   ;shift operators (note: fixed only, not
float)
< <= > >= = !=          ;comparison operators.  Returns 0 or
1
and, or, eor;logical comparison operators.  Returns 0
                        ;or nonzero (true).  Note: identical to
                                ;bitand etc. but lowest
precedence
```

Functions

```
byte, ubyte, int, uint, float       ;Type conversion
a = int(x)+c            ;Convert x to type int, then add

floating-point functions: (BASIC ROM routines)
abs                     ;absolute value
atn                     ;arctangent (note: will rename as
atan)
cos                     ;cosine
exp                     ;exponential
trunc                   ;truncate
log                     ;logarithm
rnd                     ;random number
sgn                     ;sign
sqrt                    ;square root
tan                     ;tangent

misc functions:
not                     ;logical not (0->1, nonzero->0)

VIC functions: (may be used in expression)
SpriteColSpr(number) ;Check spr-spr collision.  Returns
0/nonzero
```

Flow control, loops

Note: may be followed by a ! to execute the block when _false_:

```
        if!             ;ifnot -- if not true then...
        elseif! while! until! case!
```

May be followed by + to use branch instead of jmp (if+, next+, …)

```
if (expr)               ;standard if-block
  [code]
elsif (expr)
```

```
  [code]
else
  [code]
endif

for x=start:end [step y]  ;standard for-loop.  step is optional
  [code]
endfor -or- next          ;next is alias for endfor

while (expr) / endwhile ;standard while-loop
repeat / until (expr)           ;standard repeat-loop
repeat / forever                ;infinite repeat loop

testall                 ;test all of the following cases
(inclusive test)
testelse                ;test cases like if-elsif-elsif (exclusive
test)
  case (expr)           ;execute following code if (expr) true
  case (expr)           ;multiple case statements may follow
endtest                 ;end statement
```

Printing and I/O commands

```
sprint [string]         ;Simple print (String print) -- prints
strings.
sprintln                ;Like sprint, but adds a CR at the end.
sprint(10,3) "yo!"      ;Print to row 10, column 3

print                   ;Main print command.  Requires core
library
print !5"Hey!"!13       ;strings…
print "x=" x " and y=",y  ;mix and match, commas optional…
print(10,3) "b=" float(b+10*sin(x))  ;expressions, (row,col)
allowed
println                                 ;Print, followed by CR

getchar b               ;read a char, store in b.  Does not
wait for char
waitchar b              ;wait for keypress, store in b
waitchar                ;wait for keypress

Input b$                ;input a string using kernal CHRIN,
store in b
InputStr(maxlen) b$     ;input using core library, max string
length
InputFloat x            ;input float using core library
InputInt y              ;input 16-bit, signed/unsigned,
dec/hex/bin

load "filename",dev,address         ;load file to memory
save "filename",dev,start,end       ;save memory to file
```

Misc

```
MemConfig number        ;Set memory configuration (location
$01, 0-7)

poke                    ;similar to BASIC poke
wait address,value      ;loop until address = value
wait $d012,255          ;wait for $d012=255

waitne, waitge, waitlt  ;wait until address !=, >=, or <= value
waitbitsclear           ;addr AND value (wait for bits to be
clear)
waitbitsset             ;wait for specific bits to be set

fillmem(start,end,fill byte)    ;Fill a range of memory with a
byte

done / donebrk          ;end program (rts/brk)
endsub                  ;end subroutine (rts)
endirq                  ;end irq routine (rti)

put 'filename'          ;PUT the file into code at current
location

saveobj "@0:testcode.o"  ;auto-save object code after
compile
autorun                 ;auto-run program after compile

SetZPTemp address       ;Set temporary ZP stack
(default=$02)
SetEvalTemp address     ;Set temporary results
(default=$0110)
SetStrBuf address       ;Set location for string buffer
(default=$0200)
SlangOut                ;Compiler off (asm only)
SlangIn                 ;Compiler back on
```

Interrupts

```
irq name(varlist)       ;irq is an alias for sub -- create an
IRQ routine
endirq                  ;...but end it with endirq instead of
endsub
jmpkernalirq            ;or exit through the normal ROM IRQ

DisableInterrupts       ;Disable Interrupts (sei)
SetIRQRoutine(name)     ;Set IRQ routine ($0314)
EnableInterrupts        ;Enable (cli)

EnableTimerAIRQ         ;Enable CIA#1 timer A IRQ (system
IRQ)
DisableTimerAIRQ        ;Disable
SetTimerA(value)        ;set CIA#1 timer A value
AckTimerAIRQ            ;Acknowledge timer A IRQ

EnableRasterIRQ         ;Enable the VIC raster IRQ
DisableRasterIRQ        ;Disable
AckRasterIRQ            ;Acknowledge VIC raster irq
SetRaster(value)        ;Set raster value for IRQ to occur at
```

VIC commands

```
UseSpriteStuff                  ;Enables below sprite
commands

SetSpriteX(#,xpos)              ;Set X-position for sprite
number #
SetSpriteY(#,ypos)              ;Set y-position
SetSpriteColor(#,color)         ;Set color
SetSpritePtr(#,block)           ;Sets the sprite pointer to block.  Will
be
                                ;updated in future to something more
useful
SpriteOn(#)                     ;Turn sprite number # on
SpriteOff(#)                    ;Turn it off

SetScrollX(value)               ;Set the fine x-scroll value
(0-7)
SetCol38 / SetCol40             ;Set 38/40 columns
SetCharData(address)            ;Set the character dot
data address
SetVideoMatrix(address)         ;Sets the video (screen)
address
SetVICBank(bank)                ;Sets the 16k graphics
bank to 0-3
```

Linker

```
REL - Assemble as RELocatable file.  Placed at top of code.
ENT - Declare label/variable/subroutine as an ENTry point
EXT - Declare label/variable/subroutine as EXTernal
```

Editor/Jammon (SCPU environment)

```
Ctrl-h from editor brings up all available commands.
SYS54016 will re-enter editor upon reset, etc.
Please see the jammon docs (they're short!) for ML monitor
details.
```

Memory map

```
$0801-$A800     Slang core
$A800-$CE00     Compiler commands
$CE00-$CFFF     Used by slang for temporary results
bank 2:         Temp: code assembled to bank 2
bank 3:         Temp: vartab, storage during
assembly
bank 8+:        Editor text stored here
hi bank:        Used to store slang environment

Upon exit:
- monitor in bank 0 at config.s address, if exited to monitor
- PPPatch at $CE80-$CFFF or so
- Re-entry code stored to $01d300 (SYS 54016 to restart)
- Editor text moved up in memory as high as it can
- Slang environment, etc. stored in highest available bank,
e.g. on a 16M system this will be bank $F5.
```

A simple tutorial: Transitioning to Slang from BASIC

If you're familiar with higher-level languages you can probably skip this part. Some people however may be familiar primarily with programming in Commodore BASIC, and that's who this section is targeted at. This section introduces some of the higher-level language features in Slang you'll need to be aware of, in comparison with BASIC.

This is also a little tutorial on using Slang, so there are lots of examples below to try out. Once you've gone through them you should be well on your way to using Slang like a pro.

Note: If you find this tutorial useful -- or if you find it totally confusing -- or if you find it anything at all -- please write me or post a note to the forum, and let me know. I'd love to hear from you. Heck, right now I'd love to hear from anybody! But feedback is very important, and it helps me to know what works and what doesn't.

Interpreted vs. Compiled
BASIC is an interpreted language, whereas Slang is a compiled language. In BASIC, you write the code, then RUN it, then errors are caught as they are encountered (Syntax Errors, Overflow Errors, etc.). You can break a program while it's running, have a look at some of the variables, change things around, then CONTinue running the program.

In a compiled language like Slang you write the code, then compile it into machine language, then run that machine language program. You don't actually run the code you've written -- the compiler converts the code you've written into machine language, which you then run.

So there are actually two parts to the program, that you can for example save to disk: the _source code_,which is the part you write, and the _object code_, which is what the compiler produces. The object code is what other people will run whenthey run your program. In Slang, source code files end in '.s', as in "spritedemo.e.s", and object files end in '.o', as in "spritedemo.e.o".

As to errors, errors such as Syntax Errors are caught during the _compile_phase, before ever running the program. Other types of errors (called runtime errors) are never caught at all -- in general if there are errors in your program it will keep on running, or may lock up the machine entirely! But Slang is designed to handle this, so that you won't lose your program.

Tutorial lesson #1:
So here is the first thing to try: we're going to write a program that will crash, so you can lose your fear right away of causing some huge problem. To start up slang, load and run the main program (slangb1.9.o or whatever it might be). Once you've got the editor running, type in the following program:

```
sprint "here comes a crash!"
donebrk
```

Then press F1 to compile the program. If there are no errors, you will get a "compile successful" message. Now press F4 to run the program --

crash, right? Go ahead and reset the machine (don't power it down; just press the reset button). Then type

```
sys 54016
```

and you should pop back into Slang. Slang stores itself up in SuperRAM, so you don't need to worry about losing your program because of a crash, and if you want to try something just go ahead and experiment!

Variables
In BASIC variables are created on the fly -- you just say "10 a=1" and off you go. In languages like Slang, you have to declare your variables before using them. Let's take a dumb BASIC program, and convert it to Slang:

```
10 FOR B=1 TO 3:PRINT "BLAH ";:NEXT
20 PRINT CHR$(13)+"PRESS ANY KEY"
30 GET A$:IF A$="" THEN 30
40 END
```

Obviously lines 20-40 are not needed, but I put them in for a reason. Here's a Slang version:

```
byte b

for b=1:3
  sprint "blah "
next

sprint !13"press any key..."
waitchar

done
```

Tutorial lesson #2:
Go ahead and type this program into the Slang editor, press F1 to compile, and press F4 to run. Easy! The first line of this program _declares_ the variable. Part of the reason you have to do this is that there are different _variable types_ available. In BASIC, you can have floating-point variables, integers, and strings, and the name itself tells basic what type of variable it is:

```
a=10        ;numeric (floating point) variable
a%=10       ;numeric (integer) variable
a$="10" ;string variable
```

As you probably know, the only reason to use integer variables is with arrays, to save memory, because an integer array takes less memory than a float. Otherwise, a plain integer variable takes exactly as much space as a regular variable, and is actually slower for calculations because all calculations in BASIC are floating-point calculations.

In Slang, this is not the case. The variable type not only changes how much space is used but also how the variable is manipulated. In Slang, adding integers is much faster than adding floats together, and the integers take less space. For example, if you change the line

```
byte b
```

in the program above to

```
float b
```

you'll find that the program becomes larger, and it's also much slower (although you won't notice that in a simple program like this).

Tutorial lesson #3:
Change b from a byte to a float, compile, and see what happens. The different variable types available in Slang are:

byte - 1 byte, signed numbers in range -128..127

ubyte - 1 byte, unsigned, range = 0..255
int - 2 bytes, signed, range = -32768..32767
uint - 2 bytes, unsigned, range = 0..65535
float - 5 bytes, signed, range = well, the usual BASIC range

In general, the smaller variables are also faster, so you'll often choose the simplest variable that meets your needs. Just a few other things are worth noting. Unlike BASIC, variables can also
have really long names, like:

```
uint ThisIsAVeryLongVariableName

b = b+ThisIsAVeryLongVariableName
```

Note also that variables can mix upper and lower case (the compiler is case-insensitive), and can mix variables of different types (you can add a byte to an int, for example). And once a variable is declared, that's it -- you can't re-define a variable as a different type.

One other thing to notice is that there is only one statement per line. Unlike BASIC, you cannot put multiple commands on the same line.

Loops and such
As is apparent in the above program, the for-loop in Slang is pretty similar to the BASIC for-loop. You can also put a "step" in there:

```
for b=1:10 step 3
```

There are two other loop structures available in Slang (and many other lanauges): while-loops, and repeat-loops. Here's an example:

```
b=1
while b<4
  sprint "blah "
  b=b+1
endwhile
```

This program does exactly the same thing that the for-loop does in the original program: while the expression b<4 is true, it performs whatever
code is in-between the "while" and "endwhile" statements. The repeat statement is very similar:

```
b=1
repeat
  sprint "blah"
  b=b+1
until b>3
```

A repeat-statement always executes at least once, because the expression check is at the end of the loop. The way to think about all these things is pretty simple:

```
while [expression]        While [expression] is true
[code] <----------   Execute this block of code
endwhile                  between the while and endwhile
statements
```

One thing that Slang does not have is a GOTO statement. You can actually do gotos using assembly language, but it turns out that you can handle all the things you might use GOTO for using for/repeat/while-loops and the if-endif structure discussed down below, and your programs will be easier to debug.

Tutorial lesson #4:
Replace the for-loop in the example code with a) a while-loop, and b) a repeat-until loop, but make it print out the text five times instead of three. Compile and run each case to verify that it works.

Ending a program
You'll notice that the program ends with the line "done". In BASIC, putting an "END" in is optional. In Slang, it is required. It tells the C64 how and when to exit the program. If you'd like to see what happens when you leave it out, go ahead and try it! Just remember that "sys 54016" will restart Slang after you reset the computer.

We can now explain the last few lines as well. The command "waitchar" simply waits for a key to be pressed. Without this line, the program would immediately end and return you back to Slang. The "waitchar" would not be necessary if this program were being called from, say, BASIC -- in thatcase, you would probably want control to return to BASIC immediately.

But when running programs directly from the Slang editor, you'll usually wantto put a line like this at the end of the program.

Tutorial lesson #5:
As a test, go ahead and comment out that line (place a ";" before the waitchar), re-compile, and see what happens when you run the program!

Print
The little program above used the "sprint" command. There are actually two print commands in Slang: print, and sprint.

Sprint -- Simple Print, or String Print -- is a simple print command that only prints strings (not numbers). As we shall see shortly, the regular print command contains certain things in a _library_, which requires an extra step, and will make your programs larger. For the moment though we can focus on sprint.

Printing a string in Slang works much the same in BASIC, with just a few little twists. Unlike in BASIC, you cannot embed special characters inside the quotes, like

```
10 PRINT "{ctrl-2}{shft-clr}nice white text on a clear screen."
```

Instead, in Slang, you have to use the character codes directly; in BASIC, you could also do the above as

```
10 PRINT CHR$(5)+CHR$(147)+"nice white text on a clear screen."
```

In Slang, you simply use a ! instead of chr$, so this would look like

```
sprint !5!147"nice white text on a clear screen."
```

The above command has one key difference from the BASIC version, however :in BASIC, PRINT normally prints a chr$(13) at the end of the line, whereassprint will not. It turns out that there's actually two more print commands in Slang, though: println and sprintln. These work exactly the same as print and sprint, but print an extra [RETURN] character at the end.

Tutorial lesson #6:
Go ahead and try replacing "sprint" with "sprintln" in the example program,and see what happens.

Tutorial lesson #7:
Modify the example program to print out the text in cyan. How about light green?

There's one more trick to print and sprint: you can specify _where_ on the screen to print:

```
sprint(0,20) "some text"
```

will print the text at row=0, column=20.

Tutorial lesson #8:
Modify the program to print the text to the middle of the screen. In contrast to sprint, print can print numbers and strings, and you can stick them all together on a single line. The catch is that manyof the routines needed to do this are in a _library_. This library is a file on the disk, containing special routines. In general, libraries can be source code, object code, or a special kind of file (a relocatable file) for use by the linker, which I won't talk about here.

Tutorial lesson #9:
What we are going to use is a simple source code file. Try typing in the following code, and compile and run it:

```
byte b

for b=1:10
  println "b=" b
next

print "press any key..."
waitchar
done

put 'putcore.e.s'
```

The two critical differences here are 1) we are now using print instead of sprint, and 2) the "put" command at the end of the file.

For now, instead of going into detail on PUT, let's just say that you need that line at the _end_ of your code -- after the done statement -- if you want to use the full print command.

Notice that we have put both a string and a number on the print line, similar to BASIC. You can also separate these with commas, if you like:

```
print "b=",b
```

It just depends on which way you think is clearer. You can also print to
any part of the screen:

```
print(b,20) "this is row "b
```

Tutorial lesson #10:
Replace the println statment with the above "this is row" statement, and see what happens.

As you can see, print is much more powerful and flexible than sprint, but the downside is that you need to PUT the core library in there, which increases compile time and makes the program much larger. Experience will help you figure out when you want to use one or the other.

The core library
Above we used the core library, using the line put "putcore.e.s" at the end of the program (whether you use ' or " quotes doesn't matter, incidentally). This is a very important library and is needed for more than just print.

Tutorial lesson #11:
Multiplication and division of bytes/ints requires this library, and you will get an error if you don't include the core library. (Print will also generate an error.) Try the following program:

```
int b

b=10
b=b*2

done
```

Now compile it, and you will get an error at the multiplication. Then add the line

```
put 'putcore.e.s'
```

to the end of the program, compile... and it will work. And since you're including the core library anyways, you can go ahead and print out b if you want to.

Why use a library? If you know BASIC, you know that a command like "print" actually calls a routine in the BASIC ROMs to do its thing. The BASIC ROMs are, for the most part, one big library, but they are in ROM instead of in a disk file.

Sometimes machine language programmers will call these BASIC routines instead of writing their own. Similarly, in a compiled language like Slang, sometimes it makes sense to call a common routine to perform some task. A library is nothing more than a collection of useful routines.

Arrays and Strings
We're just about done here. I'm not going to cover _all_ of the available commands; the goal here is to get across the major commands, and then youcan browse the slangref.txt document to check out other commands. So I'll just touch on the remaining topics at this point. Arrays in Slang work much the same as arrays in BASIC. You declare them
very similarly to other variables:

```
int b                   ;regular variable
int c(20)    ;array
```

This is similar to using the DIM statement, if that helps. There's just one thing to remember: array indices start at 0. In the above declaration,
you can address c(0), c(1), c(2), ... c(19), like

```
c(19) = 1000
```

but

```
c(20) = 1000
```

will be incorrect (and may cause a crash). There are 20 elements total: 0 through 19. If it's too confusing, then one thing you can do is to add one extra element to any array -- like, use "int c(21)" -- and not worry about it.

Strings also are really similar to BASIC, but you're going to have to get one thing straight in your head: strings are really just bytes. There is no "string" type, like byte/int/float; strings are just bytes.

You already know this, from BASIC:

```
d$="a"          ;treating letter "a" as a string
d$=chr$(65)  ;treating "a" as the number 65
```

The letter "a" is really just a number -- 65 in this case. This works the same way in Slang:

```
ubyte d

d = "a"                 ;a "string"
d = 65                  ;a number
```

That's fine for just one character, but that's not a string. In Slang, a string is just a byte array:

```
ubyte d(20)

d$ = "hello";treat as a string
d(0) = "y"   ;change string to "yello"
d(0) = 67    ;change string to "cello"
```

In this example, d is a byte array. You can treat it as a string, or as a list of numbers. As in BASIC, the "$" tells Slang to treat the variable as a string -- I won't go into the details about this, but I think the meaning is pretty clear from the above example.

Tutorial lesson #12
Here's a simple program using strings to try out:

```
ubyte d(20),i

d$="hello"
println "d$=" d$
d(0) = 67
println "d$=" d$
for i=0:5
  println d(i)
endfor

waitchar
done
```

Go ahead and play around with it -- try adding 1 to each element of the string, etc. One important thing to note: when you run the program, you'll notice that the last number printed in the for-loop is a 00. With strings, the very last element will be zero -- this is what tells Slang where the end of the string is. If you overwrite this ending zero, you'll generally get a whole bunch of garbage. So...

Tutorial lesson #13
Change the line

```
d(0) = 67
```

to

```
d(5) = 67
```

and see what happens when the string is printed. This overwrites the ending zero byte, and should in general print out a bunch of garbage. Sometimes,
if you're lucky, there will be another 00 somewhere in the array!

If-elseif-endif
The if-then structure:

```
if a=10
  do something
elseif a=11
  do something else
endif
```

It's similar to the BASIC command, except that you have an elseif or endif command to mark the block of code to be executed, just like while and repeat.

Subroutines
In BASIC, you are familiar with GOSUB -- it calls a subroutine, and then RETURN returns back to the place of the subroutine call. Slang subroutines are more sophisticated, but the idea is the same and overall they are pretty similar.

A subroutine is like a mini-program: you can define variables, have a bunch of statements, etc. The neat thing is that these variables and statements are _local_ to the subroutine -- they "belong" to the subroutine, and do not interact with other subroutines. This allows you to organize your programs efficiently.

Just like a variable, you have to define a subroutine. And you need to end the routine using "endsub".

With a subroutine, you can pass _parameters_ to the routine. For example, the graphics library has a routine called GrPlot, to plot a point. And it requires two parameters: the x- and y-position of the point to be plotted:

```
GrPlot(x1,y1)
```

Subroutines can also pass parameters _back_ to the calling routine. Once you "get" the idea of parameter passing, and of local variables, the rest is a cinch.

Tutorial lesson #14:
Type in and run the following program, which demonstrates the idea of local variables within a subroutine:

```
byte b

b=1
TestRoutine() ;Call the subroutine
println "but in the main code, b is still "b
waitchar
done

sub TestRoutine()       ;Define the subroutine
  byte b          ;create the _local_ variable b
  b=10
  println "in the subroutine, b="b
endsub          ;end subroutine

put "putcore.e.s"
```

You can see what I mean about a subroutine being a mini-program. Within the subroutine you declare variables, have statements, and end it with an endsub. You don't really worry about what other routines or the main program does; the subroutine doesn't "see" them.

So, a few observations: the "sub" keyword defines a subroutine. We _call_ the subroutine by simply typing the subroutine name in the main program. Once the routine finishes, the calling program continues executing where it left off.

The subroutine declares its own variable b. Even though it has the same name as the variable in the main program, this is a _local_ variable; it "belongs" to the subroutine, not to the main program.

Tutorial lesson #15
In this lesson, we'll try parameter passing.

```
int x1,y1

x1=12
y1=20
AddEm(x1,y1)
println "the result was " AddEm<-result
waitchar
done

sub AddEm(int a, int b)<-int result
  result = a+b
endsub

put "putcore.e.s"
```

(As before, <- is the backarrow key).

In this example, the subroutine AddEm takes two _input_ parameters, a and b, and has one _output_ parameter, result. As before, all three of theseparameters are local to the subroutine, so they could have the same name as variables in the main program.

With the return parameter, what we are really doing is making a subroutine variable _visible_ to outside routines. Normally all variables and such inside a subroutine belong to the subroutine, and are not available outside of the routine; this is how to make specific ones available outside of the subroutine. Just as with input variables, there can be a whole list of output variables.

Back in the main program, this is just another variable called AddEm<-result. You can use it in expressions, etc. just like any other variable:

```
x1 = x1 + 2*AddEm<-result
```

So in summary: a subroutine is like a mini-program, with its own variables and statements. But, you can pass parameters into the subroutine, and youcan retrieve variables out of the subroutine, as needed. Subroutines are really helpful in organizing and simplifying programs, so it's well worth taking the time to understand them if you don't already!

Saving object code
Once you've written a program, what do you do with it? What we're going to do here is save the _object_ code, and then load and run it.

Tutorial lesson #16
(always need some multiple of 16, right?)

Compile one of the example programs above (one that works!). Once it compiles successfully, press F7 to enter the disk menu. First, save the source code by pressing "s". Go ahead and enter some filename; the program will automatically append a ".s" to the filename.

Second, save the object code by pressing "o". You'll see the same filename as default, so go ahead and press return. If you now list the directory, you should see two new files: one with a .s, the other with a .o. You can now load and run the object code totally indepently from Slang.

Exit back to the editor. We are now going to exit to BASIC: press shift-ctrl-<- (backarrow, to the left of the '1' key), and you should be back at the BASIC prompt. If you'd like to try your program from here, you can type "sys 4096".

Now reset the machine. Go to the disk directory, and load your object file ,8,1.

Your object code is a machine language file, just like other programs you may own or download. Once it's loaded, type "sys 4096" to run it. (By default, Slang programs are located at 4096, but it is easy to locate them elsewhere, such as 32768 or 49152, or even make it so you can RUN them from the BASIC prompt.)

When you're done, type "sys 54016" to return to Slang.

And that's it! You should now be able to write, run, and save Slang files and be well on your way to writing new programs. If you have any problems or questions, don't hesitate to write me or to post to the forum!

Good luck!

Interview with Nick Coplan
Creator of HDD64

http://www.64hdd.com

Q - Please introduce yourself to our reader

Hi! I'm 38, a mechanical engineer by profession - but the C64 lets me follow my true passion which is electronics and programming. I first got a C64 when I was in high school, aged ~16.

Q - What is it you do for the Commodore world

I'm probably most well known for 64HDD a PC program that lets your C64 use the PC as a hard disk drive. I also provide other tools and hardware stuff.

Q - Please tell our reader about DriveGhost and 64HDD

64HDD is a program which lets you use your PC as a hard disk drive for the C64 and other Commodore computers with a serial disk port such as the VIC20, Plus/4, etc. It emulates the native protocol these machines use to talk to drives like the 1541 so the computer doesn't need to be patched or wedged in order for basic disk commands to work. It also emulates the various Commodore disk drives at a basic level allowing for various disk images to be used, for example D64, D71, D81.

It has some more powerful native modes and can support commands which were part of the CMD product range. It lacks a CPU emulation module and as such cannot run any specialised disk commands which execute code "in the drive". All that's needed is a XE1541 cable to connect the PC and Commodore, the free software download and of course a PC which can be booted to MSDOS (you can do that even to modern PCs with a boot disk).

DriveGhost is a C64 program which works with the CMD drives and 64HDD allowing data to be transferred between these drives. The transfer is at the disk level (rather than file level) and so it is an imaging or “ghosting” system. The idea is to “image” the CMD drive to the 64HDD PC, from where it can be burnt to CDROM as a backup. I’ve found it most useful as a means of backing up my RAMLink and restoring different projects to it either on a partition or whole device basis.

Q - Isn't using a PC for backup and loading applications a little like worshipping the devil

Some find it a wonderful irony to make a "modern powerful machine" act as a slave to a humble 8bit machine from 25+ years ago ;)

Q - Has anyone commented about the use of a PC

A few do and some don't like the concept of having to have the PC hardware (a second computer) next to their C64 setup for space reasons. On the other hand, many have followed my example and build 64HDD into an embedded system - that is a box that needs no screen or keyboard. You can get those IDE2CF adaptors which means your PC box can boot and run off a CompactFlash card making the whole unit quite compact.

Q - Can the software work on anything else Mac/Linux/Amiga

Unfortunately not, and I have no intent to reprogram it to support these operating systems. The problem is that because of the very critical timing protocol, so much of 64HDD's core is dependent on machine specific assembler code.

Q - What started you programming

In a general sense? I guess I've always enjoyed creating things, be that with Lego, wood, metal, design or art. Programming was another way to "create" something. Back when I started coding (the C64 was my first computer) I coded mainly applications, but a few games. I’ve kept programming ever since, but usually moved from project to project once I’ve taken something to its finish.

Q - Why are people still producing hardware/software for the Commodore 64/128 Hardware

It is fantastic that there is still new stuff coming out! The market is not that big, but the users out there I feel want to prove that their C64 is "still up to the task".

Q - Tell our reader about 64HDD Speed-Up Cartridges

These cartridges quite simply allow you to load programs (including compatible multi-part programs) at speeds far greater than the normal 1541/IEC protocol allows. They're a bit like the Epyx FastLoad cart was for the 1541, but specifically for the 64HDD system.

Q - I have been looking to purchase on of the "64HDD Speed-Up Cartridges" but the different configurations confuse me can you explain what the different versions are and what our reader needs to do, for example I think originally you needed to check various chips within the C64 is this still the case.

Basically users have two choices - serial and parallel. The serial speed-up (Turbo/XE) needs no special cable since it works perfectly with the industry standard XE1541 cable. The parallel options are far faster than the Turbo/XE, but either require you build/buy a parallel cable or go for the Pwr/Link option (which has the cable interface built-in).

The only time you need a special version chip of the carts is if you have both a SuperCPU and RAM/Link - since this combo reads the IO area of the expansion port at a faster than normal rate. The cartridges are far easier to install than the internal Kernal ROM upgrade which has been available for a while. There’s a few other options, but that about summarizes the basics…

Q - Commodore Gaming have introduce a new line of PC systems with Commodore logos what are your views on this, could it be a good thing.

It’s always great to see the Commodore name still in use, but I must admit that I think of these being "just a PC". Gone are the days that the Commodore brand had a different OS and hardware architecture that meant you could compare to your friend's Amstrad or Atari.

Q - We know the Commodore 64 had some interesting design workarounds (some would say flaws) also revising models and changing designs and revisions of the main board, can you explain about problems you have encountered getting your hardware and software to work.

The main challenges I've had have been due to the different ROM configurations on the various build boards, and the lack of consistency when it comes to whether these chips are socketed or soldered. This makes it hard to upgrade some C64s. It also some makes internal upgrades very different from machine to machine. From a programming viewpoint, the main problem I had was found years ago with some of my early games and the screen clear bug which is different between kernals v1 and v3.

Q - Did you work with Maurice Randall to produce the GEOS extensions for the 64HDD

The original 64HDD-GEOS drivers were independently developed on the reverse-engineered GEOS code produced by YTM. The GEOS-XP drivers were re-written from scratch. Maurice's articles in the Commodore World magazine provided some helpful GEOS programming advice, but did not directly help with building disk drivers. Maurice has an example of a Wheels disk driver on his distribution, but I’ve not tried any Wheels programming yet.

Q - Maurice had a special version of GEOS for the 64HDD what advantage does the 64HDD provide GEOS (apart from speed)

The only difference between the versions of the drivers on the CMDRKEY website and those on the 64HDD website is that the former version is pre-keyed to the GEOS v2.0 distribution which Maurice provides for free download. Due to GEOS licensing rights, Maurice does not allow these to be hosted elsewhere, but has been kind enough to host them for me. If users want to try the basic GEOS config and have no extra licensed GEOS to key-in its easiest if they try these D64s from Maurice's site.

Q - Can you explain to our reader some of your other hardware/ software products

Other than 64HDD and DriveGhost related… Hmm, well I've built a number of smaller interfaces for things such as controllers, etc. There's the MegaCart – a 1MB cartridge configured to store upwards of 64 Commodore cartridge ROMs (in ROM not transferred to RAM). I've dabbled with DTV, TurboProcessor, IDE64… The www.64hdd.com site also has a bunch of tools I've developed over the years for my needs which others might find helpful.

Q - Some of our reader may like to ask the question "selling all this hardware and software must have made you very rich" how big is the Commodore market

The Commodore market is not very big, its also seasonal – I find people stock up just before the northern winter season so they have something to do whilst coupled up inside . As for the money, near all I've made has been re-invested in my hobby – buying more Commodore equipment! It's a classic case of what comes around, goes around – I've bought things from near everyone selling new Commodore developments/projects as well as heaps from eBay.

Q - Mega Game Cartridge looks like a great idea, although we have seen similar , what is special about your version.

At the time the MegaCart was released it was pretty much a first of its kind. The difference between it and the followers, for example the RetroReplay and MMC64 is that various games are stored in read only memory (ROM) and the menu system. Being in ROM, the CRT images do not need to be patched for the copy protection code embedded into some of them.

Q - Can the Mega Game Cartridge be programmed with your own games, If so how is this done

The MegaCart can be reprogrammed, but must be done using an EPROM programmer. The menu then requires a little bit of reprogramming to setup titles and cartridge settings such as /game and /exrom lines. Not exactly do it yourself. Those that bought the few I made wanted games in the original unpatched format in a format that saved on the storage and wear-n-tear on their originals.

Q - What is the next project then?

Hard to say… there's a lot on the go, but not ready enough to promise. Also, of late family has taken priority. I can tell you about some of the wacky-one-offs which probably won't see "mass production". There's an internal 16MB expansion – being internal means the VIC chip can see all of it, unlike with the SuperCPU in which the VIC is limited to the standard 64k. There was the internal 65816 processor. USB interface (hardware only, never got to the software)

Q - Maurice Randall seem to be over worked and very slow to deliver orders, have you ever wanted to produce "similar" products

I've thought about it, even offered to build them under licence from Maurice – but that didn't suit him at the time. Of the CMD products, it's mainly the SuperCPU which hasn't been matched or superseded by modern day alternatives. Unfortunately for the SCPU, GEOS is the main reason and very little alternative software has taken advantage of it. Wings tried, and Clips looked promising, but was never finished…

Q - Have you met Maurice, and have you asked him about CMD

I met Maurice at the SWRAP expo back in 2001, and have had correspondence with him before and since. I've not specifically talked about CMD, other than to offer support. He's always been helpful.

Q - What is the design process for your hardware projects

Step 1 in the process is to begin with something I want. I don't usually design for what I think will sell as there's no guarantee that the project will ever get to that stage. If what I build for me seems popular to others I consider the options for making more than the 1-off.

Generally selling volumes are low, so hand built is the usually production method. Things like the 64HDD speed-up carts have proven more popular and so specialised PCBs have been made allowing for prices to drop.

Q - How long did let's say the MegaCart take from design to finished product

That was one of the quicker ones… I'm guessing about 3months. The original was pretty quickly built by recycling a SuperGames cart, but the more games / programs I found the more quirks I found about the memory config these programs expect at power-on and how they use this information as a way of copy protection.

Q - Are you ever 100% happy with your finished products or do you think, I wish I had more time to implement xxxyyy or tweak xxxaaa before having the item mass produced

Surprisingly I've had very few recalls or upgrades to existing hardware products, which I guess means they were well proven before release. Its part of the design philosophy of building to satisfy my needs, before offering the item for sale. I guess I must be my own "toughest" customer. Software upgrades are a different thing, and that's why both 64HDD Professional and DriveGhost come with free upgrades. Mostly the upgrades are new features, rather than bug fixes.

Q - After the prototype, who do you contact to create the circuit boards, and how do you know the amount to order

I use a mail order company for the circuit boards, they're reasonably fast and happy to do small production runs. It is hard picking a quantity to order. Order too many means cheaper price per board, but I can't afford to stock pile too many. I usually design the boards to suit a number of purposes by changing a few jumpers. This helps increase their adaptability meaning I can use them for another purpose if something is not a big seller. I usually order 20, and if they get sold, another 20, and so on.

Q - Have you ever prototyped a design then taken it to be mass produced and they have said "we can't make that" or "it will be to expensive"

The mail order place I use has an on-line quoting system so I'm usually able to get that info worked out before going to far… Because of the low batch sizes price per board needs to be weighed against the only other option I would have which is point to point hand soldering. Sometimes hand building something works out cheaper than a simple board.

Thanks for the chance to "talk" to your readers…

Jim Butterfield: The Commodore Guru - An Interview Copyright 1996, 1999 by Jim Lawless

.

This article originally appeared in Commodore Hacking #14.
http://www.radiks.net/~jimbo/art/c642.htm

My initial interest in the Commodore 64 computer began in 1983. At the time, my primary source of information pertaining to the C64 came from Compute! and Compute!'s Gazette publications. One author's name stoodfrom the rest; Jim Butterfield.

I used to turn to Jim's articles immediately when I managed to get my hands on a new magazine. Mr. Butterfield has the rare ability to describe complex subjects in simple terms.

I'm certain that I'm not alone when I credit Jim with having taught me a lot about the inner workings of the Commodore 64. As important as the specifics of writing code for the C64 was Jim's style. He would often write code that was readily portable to multiple CBM machines. His code had longevity and purpose. The solidity of his programs left me with a lasting impression pertaining to how software should be developed. The following interview with Jim was conducted via e-mail.

Q: What was the first programming language that you learned?

A: In about 1963, an assembly language called COGENT for a computer that few people have ever heard of: a Collins Radio C-8401. That was shortly followed by work on an IBM 1401, which had a machine language that was alphanumeric. (Honest! You could keypunch M/L directly!)

Q: Were numbers expressed in Base-36?

A: No. Decimal.

The basic machine had 1000 bytes (not 1K) of (7-bit) memory (core, not RAM!) so addresses ranged from 000 to 999 (and were given in decimal, of course). Expanded machines had 4K, then 16K ... the addresses were slightly more complex in that case. Thus, to move bytes from an area at, say address 123 to address 456 the instruction would be M123456. I AM NOT MAKING THIS UP!!!!

Q: Did you guys have contests to spell out goofy words as part of a program? (I know of a programmer who used to regularly use the return code $0BAD to indicate a problem...)

A: No (the addresses mixed in with the op codes ruled that out), but you could do fun things on a 1401 if the system manager wasn't looking such as play music.

Q: What was the first computer that you owned?

A: Not counting the TUTAC-1, which was powered by rubber bands and was more correctly a logic machine: The KIM-1, a single-board microcomputer made by MOS Technologies, Inc., of Norristown PA. MOS Technologies was subsequently acquired by Commodore.

Q: When did you first encounter a Commodore computer?

A: When Commodore acquired MOS Technologies, the computer that I had owned for over a year became a Commodore computer. Subsequently, an employee of MOS Technologies, Chuck Peddle, convinced Jack Tramiel of Commodore that they should launch a personal computer called "The PET". I got one of those not long after they started production.

Q: Did you have formal training in computer programming?

A: Yes, on that long-ago Collins C-8401. But this was more a process- control machine; it didn't use of any the newfangled (at the time) languages such as Fortran and Cobol. So my training was in machine language/assembler.

Q: What was the first book that you wrote?

A: A couple of enthusiasts and I collaborated on a volume called "The First Book of KIM", a book describing how to do things with the KIM-1 single board computer. That computer was powered by a 6502,by the way; in fact the KIM-1 board itself was designed as a engineering prototype for people who wanted to try out the chip.

Q: Was it similar to the Altair where you had to manually increment an address-counter before you could throw the switches to set the byte at that address?

A: No, the KIM-1 had an operating system in ROM. That's one of the things that made all KIM users "equal" and able to share programs, while the other early micro owners had quite a scattering of stuff.

Q: What COULD you do with a KIM-1?

A: Hey, watch it! That's like saying, "What could you do with a Commodore 64"? Although the KIM-1 came with a hexadecimal keypad rather than a keyboard, and output to a six-digit LED display, you could use those to good advantage AND hook up extra stuff. Play music? Play Blackjack? Hunt the Wumpus? Skeet shoot? Unless you had the budget for a printer, you'd have a hard time doing an accounts receivable, of course. But this is the 6502 we're talking about! And we all know it can do ANYTHING!

Q: What was the last book that you wrote?

A: It's probably the revised version of "Machine Language For the Commodore 64, 128, and Other Commodore Computers". In 1985 and 1986, however, I did produce a "pocket diary" reference guide for Commodore 8- bit computers.

Q: Have you ever written articles or books on subjects that are not computer-related?

A: My first writing experience was a treatise on transistor theory, published by Popular Electronics in August of 1959. Not much else.

Q: Did you write commercial software for any of the Commodore computers?

A: As a general rule, no. All my stuff is public domain. At one time, I had written a simple spell-checking engine that was incorporated into a word processing package for a while.

Q: SuperMon was a tool that I used daily when developing ML routines or exploring the C64. What prompted you to write SuperMon?

A: In the early days of Commodore personal computers, there were quite a few machine language monitors around. They were partly based on some publicly published code by Steve Wozniak (of Apple!), and partly based on the MOS Technology TIM monitor, from KIM-1 days.

Two variants of the basic monitor caught my eye: NewMon, which added several useful features to the basic Machine Language Monitor; and HiMon, which sited the monitor in upper memory where it wouldn't conflict with BASIC programs. I decided to put the two together and generate a self-relocating MLM. That was desirable in early PET/CBM days, where some computers would come with 8K RAM, some with 16K, and others with 32K; you couldn't assume where the top of memory would be.

In those days, almost every Commodore computer came with a small built- in MLM, and the first Supermon was an add-on. Later, as Commodore changed the style of the MLM packages they built into newer machines such as the 128, I went back and modified those earlier versions so that they would work the same across all platforms.

Q: Did you ever expand the mini-assembler in SuperMon into a full-blown assembler development package?

A: No. I hustled Brad Templeton into writing PAL, so that there would be an assembler available for those who needed it. There had been a few assemblers around before that - Commodore had one, and another was the MAE system - but I was sure that somebody like Brad could do better.

Q: Even Superman had to put up with Kryptonite. Describe your worst experience as a software developer / technical writer.

A: My first publication of SuperMon in Compute! magazine had the wrong end-of-address supplied (my fault). I got a LOT of mail and phone calls on that one.

Q: I had heard a rumor pertaining to your software development habits that indicated you would approach a given project with full force. You would focus your undivided attention on it until it was complete. Is this rumor accurate?

A: Possibly. If I have a project under way, it "follows me around" until it's complete; I fret over it and can't put it away until all the pieces are in place.

Q: If so, did you ever change this methodology?

A: Not to any great extent. A half-written program bugs me, and I won't rest until it's finished .I might, however, decide that I'm taking the wrong track, and scrap a program completely in order to start over. This isn't a loss: the first attempt can show you what's really wanted.

Q: Your articles made you seem a bit omniscient. You always had the inside info on the newest CBM computers and always seemed to be able to explain their complexities in a manner that would suggest that you had a lot of time to study them. I don't know a whole lot about your employment during the mid/late 80's. Were you affiliated with CBM? A beta-tester?

A: I had many friends in Commodore Canada, but I never worked for the company, although I did contract work for them on occasion.

The big problem was not getting information from Commodore; it was learning to ignore most of it. Commodore was bubbling over with ideas and plans that never came to fruition. There was no point in writing about projects that never happened (the Commodore music box? the cash register? the videotape/disk storage device?). I took the position: "Don't tell me about it until it's a real product!".

Commodore Canada was an excellent source of information, and I relied on them to keep me from straying too far into technical speculation.

Q: Did you use any high-level languages on CBM computers?

A: BASIC, of course. COMAL, a BASIC derivative language from Denmark, was nicely constructed. Played around a little with C, but that language doesn't fit comfortably into an 8-bit environment.

Q: What was your favorite computer that CBM produced?

A: I don't know that I have a single favorite. The early PET/CBM machines were great "discovery" platforms, where we could investigate these wonderful new computers. The advent of the VIC-20 and the Commodore 64 brought color and sound, which added to the charm of these home computers; but they paid a penalty in slow disk access and screen width limitations. Today, perhaps the Commodore 128 ranks as the best, or at least the computer with most general usability. But it wasn't produced in quantities as great as some of the earlier machines, and so the user community hasn't been quite as furious.

Q: What kind of home computer do you currently use?

A: C128 .. Amiga .. Pentium system. All three.

Q: Who were your influences as related to writing?

A: Nobody specific. Just tried to write it as I would say it.

Q: Who were your influences as related to programming?

A: I've worked with a lot of sharp programmers over the years. Not one I can pick out especially.

Q: If you could relive the CBM glory years, would you do anything differently?

A: I don't think so. On another path, I could have gone for big bucks; but making money carries a responsibility to support and service, and that would have taken the fun out of it.

Q: Is your current job computer-related?

A: I'm currently more or less retired.

Q: If you had not chosen a career in computing, what field of endeavor would you most likely have pursued?

A: Before computers, I worked in electronics and telecommunications.

Q: What are your current hobbies?

A: Reading; travel; films; raising my daughter. (That's a hobby???)

Q: What sort of technical literature do you currently read?

A: Mostly reference material. Current magazines are heavy on the "what's for sale" stream; to my mind, that's not the fun part of computing.

Q: Are you surprised that a sort of "CBM renaissance" has been taking place the last few years (...availability of C64 emulators on multiple platforms and such...the SuperCPU from CMD...).

A: It's a shame that Commodore wasn't able to/interested in keeping the 8-bit line going. It's good to see that is happening.

Surprised? A little. But enthusiasts and user groups have always had a stronger effect than manufacturers are willing to admit.

Q: What is your opinion on the way consumer computing has evolved since the inception of the early PET machines?

A: The average computer user today has a lot less fun than we still have with the early machines. The industry message today is "Buy it and use it, and then turn it off .. don't worry or think about how it all works". That's sure a lot less fun for tinkerers.

Q: What words of wisdom would you care to impart on a new (or revitalized) generation of CBM hackers?

A: Enjoy what you're doing! If it becomes drudgery, you're doing it wrong!

Taken from http://www.radiks.net/~jimbo/art/c642.htm
Commodore Free would like to thank Jim Lawless for permitting the reprinting of his interview with Jim Butterfield

The End !

Commodore Free is a FREE to download magazine covering Commodore Computers available in Text, PDF, Html text, and D64 disk image for use on Commodore 64/128 Computers

www.commodorefree.com