ISSN 0265-5195

Forthwrite

September 2001

Issue 113

news events people reviews projects programming

Treating Data as Source

September
2001
Issue 113

Editorial

In this issue, Dave Pochin returns again with more on Windows and Jenny Brien gives us advance access to her euroFORTH paper.

Welcome to Julian Noble who begins a 3-part article on the use of assembly in Forth.. We also hear that Joe Anderson's report on the remarkable NEAR space probe is being translated for reprinting in the German Vierte Dimension.

There's 4 pages of news and also items on smart cards and 4-bit Forth from the newsgroup.

Look out for a project based on the "Quikwriter" proposal from the last issue.

It's time for the AGM again – your ideas and comments are, as always, very valuable. Why not deliver them in person?

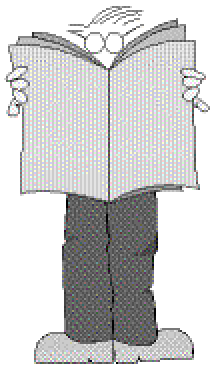
Welcome to new members Glyn James from Kettering and Roland May from Sylmar in California.

Have you noticed the web-site's new look? It uses "css" which makes it much easier to maintain.

PS. Don't forget the monthly IRC session. Our next one is Saturday 6th Oct on IRCNet channel #FIGUK from 9:00pm.

Until next time, keep on Forthing,

Chris Jakeman



Forth News

Application News

Primetime Emmy Engineering Awards for technical achievement included Clairmont Camera of North Hollywood for a series of specialized lenses.

The company makes lenses that impart special effects.

For example, the company's **Squishy Lens**, developed by technician Michael Keesling, is made of a silicon gel and can smear an image.

The Squishy Lens has been used in *Species II*, *What Women Want, Three Kings*, as well as *Star Trek - The Next Generation*, *The X Men* and a handful of commercials.

Mike Keesling writes, "The Squishy Lens was my second big FORTH project using MNI's products. It used a 68HC11, 3 7056 motor drive boards, and an A/D converter board.

The prototype took 6 months to design, including the optical and mechanical design. The software took me about 4 weeks, but 2 of that was tuning up the homing routines and PID coefficients. The code used about 4K of ROM, no floating-point math and less than 32 bytes of RAM.

FIG UK member, Howerd Oakford, has made **PPP.com** available on-line, free for non-comercial use, at http://www.inventio.co.uk.

PPP.com is the latest version of a generic communications protocol debug package supplied by Inventio Software Ltd. It is a DOS-based PC program to analyse, display and create PPP and Internet protocols.

Elizabeth Rather reports NASA's Goddard Space Flight Center is currently using Forth not only on the RTX2010 but also the UT69R000, another RAD-hard space-rated processor for which Forth Inc. developed a version of its SwiftX cross-compiler.

Stephen Pelc of MPE Ltd. discussed the use of Forth in advanced **mobile phones** and wrote “We did a <games> engine for a mobile phone manufacturer and reduced the size of the games on it by an order of magnitude.”

Resources

Glen Paling writes, "Andrey Cherezov at his team at www.delosoft.com has ANS Forth compilers for all Win32 devices including Windows CE. They've devised the compilers so that they're source code compatible on all Win32 devices."

Neal Bridges suggests members check out Kris Johnson's Wiki

http://sleepless-night.com/cgi-bin/twiki/view/Main. It's most heavily used by Quartus Forth developers at present, but there's a section for general Forth material.

Anton Ertl summarised the several places where Forth extensions have been documented.

http://dec.bournemouth.ac.uk/forth/ans/extentions/index.html (note spelling of “extensions”) collects suggestions for the next revision of ANS Forth in 2004.

There is also http://www.albany.net/~hello/comus.htm, which tries to collect common usage.

And of course there is Anton’s own list of proposals at

http://www.complang.tuwien.ac.at/forth/ansforth/proposals.html

In answer to a recent newsgroup enquiry, Hans Bezemer directs the newcomer to

http://forthprimer.siteaddr.com for a 85 page tutorial to the ANS Forth standard.

People

There's an active discussion topic right now about Charles Moore at http://www.slashdot.org which begins "Chuck Moore is, among other things, a chip designer. His latest design, the **25x**, is based on a 5x5 array of X18 microprocessor cores, and could provide 60,000 MIPS with a production cost of about one dollar. And Moore has the chops to back that up: he's been designing tiny, efficient processors for many years."

322 queries and comments had been listed by 28th August and Charles Moore will answer his selection shortly.

Krishna Myneni has published the following example of using Forth for teaching or learning **quantum mechanics**:

ftp://ftp.ccreweb.org/software/kforth/examples/qm4.4th

Apart from providing the numerical computations that illustrate various concepts in quantum mechanics, the use of Forth in this instance seems to me to be a neat example of creating a pseudo-language that is natural and well-suited to a specific application.

Public Forth Systems

FIG UK member, Gary Lancaster, has published a new v3.0 of **CamelForth** for the Z88 at http://www.z88forever.org.uk/camelforth/rom-camel.html

This is a fully ANS-compliant extension of Z80 CamelForth by Bradford J. Rodriguez, with a multitude of extensions for the Z88, an unusual pocket computer.

The new version includes multi-programming support (with simple demo) - a first for the Z88 - and a 6-8 improvement in interpreting source from files.

Tom Zimmer has modified **Win32Forth** following a suggestion from Richard Adams, with code provided by Bernd Paysan, to allow filenames in Win32Forth to contain spaces.

Jih-tung Pai has published a new version of **ppForth**, a Forth for the Palm Pilot, at

http://206.171.116.227/forth/index.html

including view words to look up source and the ability to produce standalone applications with resources.

Brad Eckert reports that a new demonstration system for **Tiny Open Firmware** is available at

http://www.tinyboot.com/eval31.html

Self-installing peripherals really need to be seen in action to be appreciated. The demonstration consists of an 8031 board and one or more 20x4 LCD modules. The CPU monitors the serial expansion bus to allow hot plugging. Add or remove a module, and the system reboots and reconfigures itself in a couple of seconds.

Chuck Moore's **ColorForth** is now publicly available and has generated much interest. In addition to his personal site, reported in the July issue, he is now providing a new site, http://www.colorforth.com, dedicated to ColorForth. (See also the section People below.)

Terry Loveall has assembled a web page for resources on Chuck Moore's **ColorForth**

http://www.users.qwest.net/~loveall/c4index.htm

providing a single starting point for all ColorForth sources, binaries, updates and applications.

Jeff Fox has provided a mailing list for **ColorForth** at

http://www.ultratechnology.com/chips.htm#maillists

John Sadler announces that **Ficl** release 3.00 is now available for download at

http://sourceforge.net/project/showfiles.php?group_id=24441

Release 3.00 changes the programming interface to permit multiple Ficl systems to coexist in a single address space. Thanks to Orjan Gustaffson for contributing these mods.

There are also bug fixes for 64-bit compatibility (thanks to DCS (again) and the FreeBSD mob), and fixes for various bugs in the debugger, parse-steps, and OO support. The linux makefile and the tar.gz package has been tested on the sourceforge compile farm, so it should be trouble free.

Marcel Hendrix, author of the high-performance **iForth**, has made the manual and the glossary available on-line at:

http://www.iae.nl/

users/mhx/i4thmanual.pdf

http://www.iae.nl/

users/mhx/i4thhelp.htm

Comments (by e-mail) would be appreciated.

Commercial Forth Systems

Triangle Digital Systems have now approved the IBM Microdrive for use with their TDS2020F + TDS2020CM2 Data Logger Modules. Its one gigabyte capacity is the largest available on the market in the Compact Flash format.

Stephen Pelc reports that **VFX Forth** for Linux is under development and will be available soon.

MPE's **VFX Forth** for Windows build 3.40.0685 is available for download, as reported in our July issue. A summary of the optimisation results was posted by Stephen Pelc:

```
Primitives using no extensions, test time
(ms) including overhead for VFX3.4, iForth
and SF2.0
  1.Eratosthenes sieve
  2.Fibonacci recursion
  3.Hoare's quick sort
  4.Generate random numbers
  5.LZ77 Comp.
  6.Dhrystone

Total time in msecs:
  1,893 for MPE ProForth VFX 3.40.0686
  5,445 for iForth by M. Hendrix, v1.12.1121
16,103 for SwiftForth 2.00.3
```

Jenny Brien
02866 388 253
jennybrien@bmallard.swinternet.co.uk

Treating Data as Source

Jennifer Brien

Forth has a number of useful facilities for manipulating the interpretation of source text. Unfortunately, they are not always so accessible or useful in the 'interpretation' of other text data. This is the first half of a paper submitted to euroFORTH 2001 proposing a variation of ANS Forth that addresses the problem, and also shows a work-around that is illustrated with a simple and extensible XML parser.

What the Standard Forth Interpreter provides

All Standard[1] parsing words use the current input stream. That portion of the input stream currently in memory (the input buffer) is returned by `SOURCE`, and `>IN` provides an index to the current position in this buffer, which is updated by the parsing words. In theory, the system can be ignorant as to how the input buffer was filled: `REFILL` will fill it from the current input stream, if there is one. Input can be re-read by saving and restoring `>IN`, or by using `SAVE-INPUT ... RESTORE-INPUT` if more than the contents of one input buffer is involved.

Sadly, when analysing a data stream all this abstraction is not available and we have to write its equivalent from scratch, which is very frustrating when the Standard words would provide what we need if we were working with the input stream.

The input buffer (depending on its type) is `EVALUATE`d, `LOAD`ed or `INCLUDE`d, and that's all you can do with it. It has to contain valid Forth source. What we need is a way to use an input buffer with any function, not just the Standard's text interpreter.

Treating any string like the Input Buffer

Many Forths already supply `SOURCE!` to set the position and length of the input buffer. It's simplest and most versatile use is to 'unparse' any parsing word, allowing it to operate on any string:

```
: EXECUTE-WITH         \ ca u xt -- ; execute parsing word with ca u as its input
        >IN @ >R  SOURCE 2>R  >R SOURCE!
        R> EXECUTE
        2R> SOURCE!  R> >IN ! ;
```

[1] The current standard for Forth is ANS Forth (1994) and also adopted by ISO.

Example:

```
    : $CREATE           \ ca u  -- ; create a header using ca u as the name
            ['] CREATE execute-with ;
```

It's a matter of choice whether you regard this as passing a string to a function or a function to a string!

This will work so long as we are sure the original input buffer has not been over-written. As Michael Gassenenko has pointed out,[2] the problem is not with implementing the word itself, but with what happens if `SAVE-INPUT` or `REFILL` is attempted before the original input buffer is restored. Gassenenko suggests that implementors should see the input buffer and the refill buffer as separate entities. All input is taken from the input buffer returned by `SOURCE`. `REFILL` should not depend on the current value of `SOURCE`, but only on `SOURCE-ID`. It should refill a system-supplied refill buffer, and set that to be the `SOURCE`. Changing the input buffer with `SOURCE!` would then cause interpretation to return to the next line of the original stream when `REFILL` is called. It follows, then, that changing `SOURCE-ID` without changing `SOURCE` would cause interpretation to pass to the next line of the new stream when `REFILL` is called. (Jenny goes beyond Michael Gassenenko's proposal with this exposition and by showing that `SOURCE-ID!` should also be added to the standard - Ed)

A word-set for manipulating the input stream

Given assurance of that behaviour, it is possible to specify a complete portable input-manipulating word-set with the provision of just 2 words:

```
      : SOURCE!       \  ca u  -- ;      Set input source spec. Set >IN to zero
      : SOURCE-ID!    \ source-id -- ;   Set source-id
```

Nesting and un-nesting sources can then be done in the same style as `SAVE-INPUT` and `RESTORE-INPUT`:

```
      : SAVE-SOURCE          \ --  xn ..x1 n ;
        SAVE-INPUT >R SOURCE SOURCE-ID R> 3 + ;

      : RESTORE-SOURCE      \ xn..x1 n -- ;
        >R SOURCE-ID! SOURCE! >R 3 - RESTORE-INPUT THROW ;
```

The same definition can get input from any `REFILL`able source (`stackem` and `unstackem` move counted sets of parameters to and from an extra stack).

```
      : INCLUDE-WITH          \  i*x  source-id xt – j*x ;
            save-source stackem
            >R source-id!
```

[2] http://forth.sourceforge.net/word/source-store/index.html

```
        REFILL BEGIN  R@  EXECUTE WHILE REFILL 0= UNTIL THEN
        R> DROP
        unstackem restore-source ;
```

By Gassenenko's rule, the first `REFILL` sets `SOURCE` to the address and count of the input buffer identified by `SOURCE-ID`. Because `INCLUDE-WITH` does not open or close files, it will start at wherever the current `FILE-POSITION` happens to be (i.e. with a newly-opened file, it will start with the first line). The function passed to `INCLUDE-WITH` deals with one line of input per call - though it can deal with more by calling `REFILL` itself - and exits with a flag to say if it requires any more input.

`0 SWAP INCLUDE-WITH` will provide such a function with input from the terminal input device.

Treating the Input Buffer like any string

The input buffer is, after all, just another area in memory. Any string-scanning definitions can work with it, so long as we have a pair of words which convert a `>IN` index into a *caddr u* pair and vice versa:

```
      \ get the as-yet-unparsed portion of the input buffer
      : PARSE-AREA@        \ -- ca u ;
        SOURCE >IN @  /STRING ;

      \ set the portion of the input buffer still  to be parsed
      : PARSE-AREA!      \ ca u -- ; must start within the input buffer!
          DROP SOURCE DROP -  1 CHARS /  >IN ! ;
```

This eliminates the awkward difference between parsing source with `PARSE` and `WORD`, and parsing strings with other pattern-matching words, when you are left with the *caddr u* of a still-unparsed string. Use `COMPARE`, `SEARCH`, etc. to write general pattern-matching words with the stack diagram:

```
      PatternMatch     \ ca u -- ca1 u1 ca2 u2 ;
```

where `ca1 u1 =` *`string-remaining`* and `ca2 u2 =` *`string-matched`* and use them in the form:

```
      PARSE-AREA@  PatternMatch /dosomething/  PARSE-AREA!
```

Where `SOURCE!` is available it could of course be used instead of `PARSE-AREA!` which is in fact just another way of manipulating `>IN`. It can only modify the size of the parse area by changing its start point, but that is all that is required in this situation (and all that can be achieved within the Standard).

```
      : STRING/         \ ca1 u1  u -- ca2 u2
             SWAP - TUCK - SWAP ;
```

This is the reverse of `/STRING` and a useful way to end a `PatternMatch` word, getting the *string-remaining* from the *string-matched* and the length of the original string,

Data as Source in Standard ANS Forth

The effect of `INCLUDE-WITH` can be simulated within Standard Forth by simply `INCLUDE`ing a data file. So long as we know what the first word the interpreter encounters will be, we can arrange for it to read and process the rest of the file. A good example of this technique is the method[3] used by Bernd Paysan to add 'active HTML content':

```
: $>
        BEGIN
          SOURCE >IN @ /STRING
        S" <$" SEARCH 0= WHILE
            TYPE CR
          REFILL 0= UNTIL
            EXIT
          THEN
          NIP SOURCE >IN @ /STRING ROT - DUP 2 + >IN +! TYPE ;

: <HTML>  $> ;
```

The Forth code was embedded in a web page between the tags “<$ “ and “$> “. The page itself began with `<HTML>`. When the Forth interpreter executes `<HTML>`, it calls `$>` to search the input line by line for “<$ ”, printing the lines as it goes. It then resumes interpreting from after the “<$ ” onwards. When the interpreter reaches the ending tag `$>`, it executes it and repeats the search.

The concluding part of this article appears in the next issue of Forthwrite and uses the JenX parser from the previous issue to illustrate this technique.

Jenny Brien has been experimenting with Forth as a hobby since 1986, contributing frequently to Forthwrite, and is only now taking a HNC in Computing. She is also an artist and local historian, and is currently researching the water-mills of Fermangh, of which there were once more than a hundred.

[3] See Forthwrite issue 108, August 2000

From the ‘Net – 4-bit Forth

Tom Zimmer

Tom Zimmer posted this item about 4-bit Forth applications on the comp.lang.forth newsgroup after someone described Forth as “requiring a minimum of 16 bits”. The very first microprocessor was a 4-bit device – the Intel 4004.

“I have told this story before, but what the hay, one more time;

I was hired at Samsung about 10 years ago, to revamp a micro-controller development system for their line of Super8-based 8-bit processors, and for their line of 4-bit processors (yes, 4-bit) called the 56000. I worked with Jerry Boutel, Robert Smith and others to create a development system that ran on F-PC, and used the customized versions of the TCOM compiler. The target and development system for the Super8 processor was a pretty normal call-threaded implementation, with optimizers that produced code which could be debugged remotely at the source level and was eventually burned into onboard MASK ROM. These applications were limited to 256 bytes of RAM, all that was available on the Super8.

The really interesting target though was the 56000 4-bit processor, which had versions with up to, I think, 32k of ROM, and 256 nibbles[4] of onboard RAM. Fortunately, this processor had a couple of interesting features. First it had multiple address spaces, one for RAM, one for ROM and several for I/O, so short addressing could be used for all access to RAM, and I/O. Another interesting feature, was the ability to assign byte tokens to subroutines. The TCOM compiler for the 56000 would automatically assign about 16 of the available 48 tokens to predefined Forth primitives, and left the remaining 32 tokens for assignment to user Forth functions.

“4-bit processors are still viable and available from at least a dozen major suppliers. There will always be high volume products (at least 1 million pieces) that need to do some limited processing at very low cost (under $1). Examples are microwave ovens, bathroom scales and telephones.

We get spoiled with our sub-GHz computers, but the real world doesn't need that kind of processing power for a lot of things. The biggest problem with 4-bit processor is that they tend to be a bear to program. Forth made it more than bearable, it made it fun.”

Tom Zimmer 4-Sep-2001

[4] A “nibble” is 4 bits of data or half a byte. I would love to know who thought of that one – Ed.

The result was that you could generate some truly small programs on the 56000. We added nibble memory operations like N@, and some byte operations like B@. Developing an application truly had the flavor of Forth, even though you were obviously developing for a very specific target processor. The implementation was essentially Forth 83, though portability doesn't really come into play in this environment. If I sound excited, after all these years, I confess I am. It was one of the most interesting development projects I have ever worked on.

Of course Samsung thought everyone would want to program these processors in assembly language, so Robert Smith wrote PASM, a Programmable ASseMbler, which could easily be tailored to any of the processors Samsung built. We did, I believe, have several people that developed their applications in Forth. By the way, the cost of these processors was under a US dollar.

Just my ramblings,

Tom Zimmer"

In a recent poll at www.embedded.com received 1346 responses and listed programming languages for embedded work as:

C	68%
C++	17%
Java	5%
Forth	3%
Other	3%
Ada	2%

his shows that Forth continues to hold its position as a tool for embedded work whose worth is recognised and valued by a minority. (Although you do have to wonder where the assembly-language programmers are. Perhaps they're hidden in the Other category.)

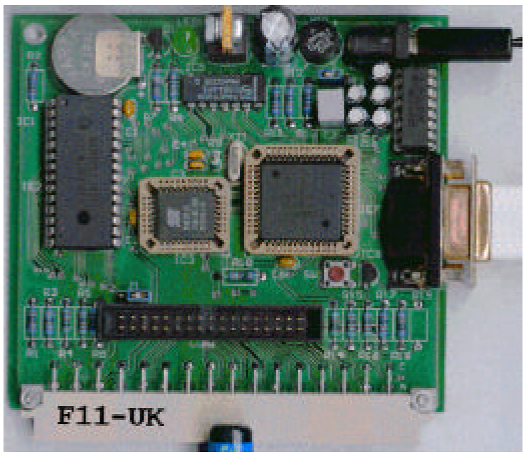
F11-UK

provides everything needed in a professional-quality low-cost Forth controller board.

Use it in industrial or hobby projects to control a wide range of devices using the well-known multi-tasking Pygmy Forth.

Designed for hosting from a Windows or DOS PC, you can test your application as it runs on the F11-UK board itself. The board was developed by FIG UK members to provide an easy way to explore the world of controlled devices – a niche where Forth excels.

The kit includes both hardware and software and is supported and sold to members at a nominal profit through a private company.



All source code provided - 78 pages or so (unlike many commercial systems).

Around 30 pages of additional documentation is supplied including a full glossary of the 300 or so Forth words in the system.

Email mailing list for discussion and limited support.

Software

PC-based PygmyHC11 Forth compiler running under DOS produces code for Motorola HC11 micro-controller.

Code is downloaded via standard serial link from the PC to the FLASH memory (or RAM) on the F11-UK single board computer (SBC).

No dongle or programming adaptor of any kind is required.

Forth running on the SBC is interactive which makes debugging and testing much easier.

Multitasking and Assembly included.

The serial link can be disconnected to enable the SBC to function as a stand-alone unit.

Hardware:

Processor:
Motorola HC11 version E1 - 8 MHz (2 MHz E-Clock).
Memory:
32k x 8 FLASH
32k x 8 battery backed SRAM
512 x 8 EEPROM onboard HC11.
I/O:
20 lines plus 2 interrupts (IRQ & XIRQ).
Analogue in:
up to 8 lines using onboard 8-bit A/D.
Serial:
1. RS232, UART onboard HC11
2. Motorola SPI bus onboard HC11.
Expansion:
Via HC11 SPI serial bus using
2 or more of 20 available lines.
Timer system:
Inputs: 3 x 16-bit capture channels
Outputs: 4 x 16-bit compare channels.
PCB size: 103 x 100 mm.

Price to FIG UK members:	£47.0 plus postage and packing (£2 UK, £4 overseas) plus $25.0 (US Dollars) for registration of 80x86 Pygmy Forth with the author Frank Sergeant.
Delivery:	ex-stock.
More information:	jeremy.fowell@btinternet.com and 0121 440 1809

euroFORTH 2001

The 17th annual euroFORTH conference on the Forth programming environment and Forth processors is being held on November 23 – 26 at Schloss Dagstuhl, near Saarbrücken, Germany.

This annual conference is held in the UK every third year and, after the 1999 venue in St.Petersburg, it returns again to Schloss Dagstuhl. (See Paul Bennett's detailed report in issue 99). The conference language will be English.

For conference details, see http://dec.bournemouth.ac.uk/forth/euro/ef01.html. FIG UK member Bill Stoddart is the Program Chair and invites papers (both academic and business) by 26th August, please, to the Proceedings Editor Peter Knaggs (pknaggs@bournemouth.ac.uk). Topics of especial interest include:

- Forth applications and language extensions.
- Open protocols and standards, including TCP/IP, HTTP, XML etc.
- Virtual machine application and design.
- Stack-based architectures.
- System configuration and Open Boot.
- Other topics likely to be of interest to the extensible language community.

Charles Moore is the Guest of Honour, so this is a rare chance to meet the inventor of Forth on this side of the Atlantic.

Attendance with a single room in the castle for 2 nights and full board costs €350 (£220)5. Discounts are available for students sharing rooms.

Bill tells me that prices have been kept to a minimum to encourage the widest possible attendance. Let's take advantage of that – Ed.

FIG UK – AGM

The Annual General Meeting will be held on Saturday 20th October at Doug Neale's home, 58 Woodland Way, Morden from 2:00pm.

All members are cordially invited to do attend. If you cannot come, but wish to comment on the way FIG UK is going or the direction you would like it to take, write or e-mail Jeremy or myself before the meeting.

Anyone who lives in the London area can get to Doug's house easily by Underground as he is just ten minutes walk from the southern terminus of the Northern Line. You can get directions from www.multimap.com for his postcode SM4 4DS or just phone him on 020 8542 2747.

Some of the topics likely to be discussed are:

- Joint projects such as Keyboard Project
- Ideas for the Web Site
- Finances – see annual accounts on next page
- EuroFORTH 2001
- Promoting Forth and FIG in the UK

Forth Interest Group UK: Revenue Account for year ending 31 March 2001

Income and Expenditure

Y/E 31/3/00			Y/E 31/3/01	
790		Subscriptions	1,293	
3		Interest (net of tax)	8	
	793			1,301
		Printing Forthwrite	593	
		Postage	215	
		Other Printing	26	
		Sundry Expenses	50	
	845			884
	-52	**Net surplus for the year**		**417**

Balance Sheet as at 31 March 2001

Y/E 31/3/00			Y/E 31/3/01	
	579	Accumulated Fund b/f		527
	-52	Surplus for the Year		417
	527			944
Represented by:				
	1,221	Cash at Bank		1,580
654		Unexpired Subscriptions	586	
40		Sundry Creditors	50	
	694			636
	527	**Net assets**		**944**

The accounts were prepared by our Treasurer – thanks, Neville.

Julian Noble
jvn@virginia.edu

A Call to Assembly 1/3

Julian Noble

Institute of Nuclear and Particle Physics
University of Virginia
Charlottesville, VA 22901

This is the first part of a paper originally prepared for the sadly defunct Forth Dimensions magazine.

Introduction

Forth programmers tend to take for granted the assembler that accompanies most Forths[6]. We often eschew assembly language definitions because they are not portable, especially since, in the era of ANS Forth, portability represents an important goal of programming. We therefore resort to the assembler only when running time is of the essence, or when we must access the underlying system at its most basic level---direct control of ports, drives and displays.

However, one of the things members of the Forth community do (besides program in Forth) is attempt to educate their peers who still muddle about with languages of lesser quality. It has for some time seemed to me that in our proselytizing we were missing a good bet, by extolling first the high level features of Forth---its extensibility, abstractive power, simplicity, elegance, etc. etc.. I think it may be better to introduce Forth to the suspicious outsider by way of the Forth assembler.

To make clear why I have taken this position, let me recapitulate what assemblers are and why they exist. In their essence computer programs consist of sequences of numbers, generally in base 2 (binary) or 16 (hexadecimal) format. Since human brains never evolved to use numbers in any base, the man-machine interface suffered from impedance mismatch in the era when digital computers were programmed directly with plug boards or switches. Programming in this fashion, still common in my youth, was arduous and prone to error.

Fortunately today's computers rely on specialized conversion programs called assemblers to translate human-readable representations of the instructions in text form (“assembler mnemonics”) to their numeric equivalents. Good assemblers recognize macro instructions and operations (“pseudo-ops”) that perform such useful chores as referring to variables, constants, or frequently used sequences of instructions by name rather than by address[7].

Even with such tools, however, writing a lengthy program entirely in assembler is not a task to be undertaken lightly. Machine language programs are

[6] That is, all commercial Forths and many public domain ones.
[7] Such assembler directives as macros and pseudo-ops are not actual machine instructions, of course.

hard to get right, hard to understand, and hard to maintain or port to another machine. High level languages were invented to provide a better human-computer interface, providing standardized data structures and operations that encompass most of the user's needs, and translating these to machine language in a standardized fashion. Modern optimizing compilers can generate machine code that executes no worse than 2× slower than the best hand-coded efforts of wizard hackers. Such facts of life have led to declarations that machine language programming is obsolete[8].

What happens when we encounter problems with no reasonable solution in high level code? Memory limitations, a desperate need for speed, or an operation trivial at the level of machine registers, but time-consuming and circuitous in a high level language (e.g., bit reversal in Fast Fourier Transform, or interfacing through ports) lead us - however reluctantly - to exercise our constitutional[9] right to assemble.

“our constitutional right to assemble”

Most modern programming languages permit linking with assembly language procedures that have been assembled separately (that is, outside the compilation process), thereby combining the ease of high level programming with the advantages of assembler. The value of this hybrid approach lies in the fact that most programs spend most of their time executing relatively few instructions. Factoring such bottlenecks into separate subroutines, then hand-coding them, can garner large increases in performance. The usual procedure is:

1. program, test and debug everything in high level code;
2. using a profiler or algorithmic analysis, determine which portions can be rewritten profitably in machine language;
3. finally, endure the tedium attendant on assembling, linking and testing the hand-coded parts.

In many cases, however, Step 3 is so arduous as to discourage even minimal use of assembly language, except out of desperation.

What we really need is a way to test assembly language subroutines in isolation, i.e. to assemble and run them as separate programs. By eliminating the need to compile an “exercise” program, assemble the subroutine and link the two into an executable, we can telescope the compile – test - debug cycle into a single stage. Once we are satisfied with our machine code subroutines they can be (re)assembled and linked to the (compiled) main program once or twice at most.

[8] See, e.g., M. Abrash, The Zen of Code Optimization (The Coriolis Group, Inc., Scottsdale, AZ, 1994) for an eloquent defense of assembly language vs. high level language.

[9] A right enshrined in the US constitution, a privilege not shared in the UK.

Forth offers a shortcut that makes assembly language programming as simple as high-level programming. Although Forth is my first choice for the kind of programming I do (numeric and symbolic), not everyone likes it. Moreover, constraints imposed by management often preclude using Forth in commercial applications. However, for assembling and testing isolated machine code fragments - in fact for rapid prototyping of any sort - Forth is without peer and is worth considering for that purpose, even if the final result must be expressed in C or C++.

Assemblers, cross assemblers and decompilers in Forth are so terse that most programmers used to other languages find it hard to believe they are what they claim to be. In a commercial Forth I use regularly, the traditional (postfix) Forth assembler source code resides in a file about 14 Kbytes long, and adds about 6 Kbytes of compiled code to the system; a more elaborate assembler (for a public-domain Forth) that allows prefix style comprises 31Kbytes of source and compiles to about 8 Kbytes; the source file of a generic Forth cross-assembler for Motorola 680x0 CPUs is about 16 Kb; and the assembler for Intel 80486 and Pentium CPU's that comes with a Windows-based Forth is still a relative lightweight at 85 Kb of source. For comparison, the binary of an ancient 16-bit assembler, MASM.EXE® (v. 2.0), is about 74 Kb long.

The Forth assembler is written in Forth, hence it operates the same way as any other set of Forth words. The words for compiling a new definition from assembler mnemonics, analogous to `:` and `;`, are `CODE` and `END-CODE`. Rather than threading together the addresses of predefined words from the dictionary, the assembler mnemonics actually assemble a new machine code fragment containing the op-codes of the target CPU. To show how this works, I shall illustrate with popular public-domain Forths, F-PC, and its lineal descendent Win32Forth, both the brain-children of Tom Zimmer, that are readily available from the Web site http://www.taygeta.com.

"Forth offers a short-cut"

To a writer, the advantage of public-domain Forths is that they provide access to the machine code of the most primitive kernel words. These serve as convenient examples of the assembler's operation and show how to program simple operations in Intel 80x86 assembler.

This note provides three examples of the development process: STIB[10], a routine that bit-reverses numbers, eg. for use with the fast Fourier transform (FFT); UCASE, a routine to convert all lower-case letters in an ASCII string to upper case, leaving digits and punctuation alone, and a program for computing spherical Bessel functions (a number-crunching application) in which the key subroutine is coded in assembler for maximum speed. In what follows we assume

[10] That is, BITS spelled backwards. Forth names often seem odd to programmers used to the baroque compound names of C functions. Forth's conventions aim toward self-documenting code, with telegraphic word names that express their functionality without lengthy marginal notes. BIT_REV would also work, and may perhaps be less cryptic.

the reader is familiar with the assembly language mnemonics of the Intel 80x86 series of CPUs. Occasionally their operation will be amplified in detail; however the reader is advised to consult a standard assembly language programming manual.

Bit-reversal

The bit-reversal routine STIB may be written in high level Forth as

```
: STIB ( k n --- n')         \ reverse order of bits
   0 SWAP ( --- k 0 n )      \ initialize n' to 0 giving ( -- log2[N] n' n)
   ROT 0 DO                  \ loop k times
      DUP 1 AND              \ pick out 1's bit of n
      ROT 2* +               \ leftshift n' 1 place, add 1's bit
      SWAP 2/                \ rightshift n 1 place
   LOOP DROP                 \ end loop, discard n
;
```

How does this work? The subroutine expects an integer *n* on the stack, in the range

$$0 \le n \le 2^k = N$$

where *N* is the order of the FFT (power of 2). The loop must be executed k = log 2 (N) times, so the loop limits are 0 and *k*. For simplicity, *k* is placed on the stack above *n*, rather than fetched from a variable. To see how the routine performs bit reversal, visualize the (input) integer n in binary notation: a string of 1's and 0's in a field *k* bits wide. For example, if the order *N* of the FFT is 16
then the field is $k=4$ bits wide; the number 7, e.g. is represented as

$$n = 7_d = 0111_b$$

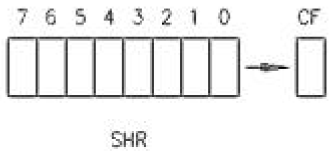
and its bit-reversed form is

$$n' = 1110_b = 14_d$$

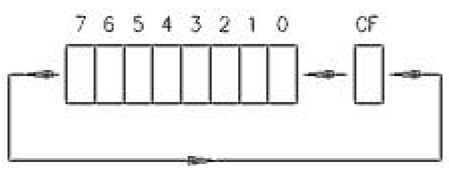
We start with $n = 0$ (all bits are 0); we then shift *n'* one position to the left, adding to it the right-most bit of *n*. Then we shift *n* one position to the right (with its former right-most bit dropping into oblivion), and repeat until done. We simulate the shift operations using integer divide-by-2 (`2/`) for the right-shift, and multiply-by-2 (`2*`) for the left-shift. We keep *n* and *n'* on the data stack (equivalent to temporary local variables that are reclaimed when the subroutine returns control to the main program).

Testing immediately, as is our wont in Forth,

```
 4 7 STIB . 14 ok
14 7 STIB .  7 ok
```



A machine code version that carries out the operations entirely within the CPU's



registers will execute much faster than the high-level code [7]. The logical right-shift (SHR) and rotate-left-through-carry (RCL) instructions are key to an exceedingly simple subroutine. Their behaviors are illustrated here.

Different Forths will require minor differences in how we proceed. Several commercial Forths cache the top of the data stack in the register BX, thereby eliminating some pushes and pops. The public-domain F-PC, on the other hand, leaves BX free. Since we are illustrating with F-PC, our first job will be to obtain the argument n; we therefore POP it from the stack to BX:

```
POP BX
```

Next let us assign the (unused) DX register to the bit-reversed answer; we initialize DX to 0 quickly using bitwise exclusive-or[11]

```
XOR DX, DX
```

Now we shift BX one place to the right using SHR; the rightmost bit, as the Figure suggests, moves from the register to the Carry Flag. Then we RCL the DX register one place to the left; the bit formerly in the Carry Flag becomes the rightmost bit of DX. The leftmost bit (if any) of DX ends up in the CF. But that does not matter, because it will be replaced by the rightmost bit of BX when the sequence is repeated. So the machine-language program (with comments) looks like

```
                    \ initialization steps
POP BX              \ obtain n
XOR DX, DX          \ n' = 0
                    \ repeat following instructions k times
SHR BX, 1           \ logical right shift 1 place
RCL DX, 1           \ rotate left through carry 1 place
```

All that is required now is to arrange to repeat the two-instruction sequence the requisite number of times. For simplicity let us do this using the most elementary looping instruction, LOOP. We must place the number of times the loop is to be executed in the register CX then, at the end of the loop, issue the LOOP instruction which will decrement CX by 1 and loop back to the starting point (which we must label somehow - we will return to this point and describe how it is done), as long as CX is non-zero. That is, it will loop the number of times specified by the integer in CX.

To assemble this subroutine using an assembler like MASM® or TASM®, we would prepare a text file of the form

```
POP BX         ; get n
```

[11] The instruction MOV DX, # 0 would also work, but requires 1 byte more storage.

```
    POP CX          ; get # of iterations
    XOR DX, DX      ; set n' = 0
    HERE:           ; beginning of loop
    SHR BX, 1       ; send 0'th bit of n to CF
                    ; and shift right 1 place
    RCL DX, 1       ; shift n' left and
                    ; move CF into 0'th bit of n'
    LOOP HERE       ; CX=CX-1, loop if CX 0.
    PUSH DX         ; leave result on stack
```

(however, as we shall see below, there will need to be some necessary boiler plate lines that conform to the particular assembler's conventions, as well as respecting the calling conventions of the high-level language we are going to use the subroutine with).

To test the assembly language program with F-PC's intrinsic assembler, we modify it slightly (to conform to the latter's notational conventions), obtaining

```
    CODE STIB       \ reverse bit-order
    POP BX          \ get n
    POP CX          \ get # of iterations
    XOR DX, DX      \ set n' = 0
    HERE            \ beginning of loop
    SHR BX, # 1     \ send 0'th bit of n to CF
                    \ and shift right 1 place
    RCL DX, # 1     \ shift n' left and
                    \ move CF into 0'th bit of n'
    LOOP            \ CX=CX-1, loop if CX 0.
    PUSH DX         \ leave result on stack
    NEXT END-CODE   \ terminate definition
```

An assembler written in Forth is simple because the mnemonics are actually `IMMEDIATE` words that execute during assembly, placing the appropriate operation codes in the parameter field of the word being defined. In the F-PC assembler the LOOP mnemonics (LOOP, LOOPZ, LOOPNZ, etc.) expect a number on the stack---which is actually the address they loop back to (or not, depending whether an appropriate condition is satisfied). This can be supplied by an explicit label or, as in the above example, we may simply say `HERE`, which places on the stack the address of the next piece of code to be assembled; this is the very point we want to loop back to, hence `LOOP` enters the Intel op-code for LOOP, together with that address.

We now enter the subroutine from the keyboard and test the result.

```
    CODE STIB ok
    POP BX ok
    POB CX POB <-WHAT?
```

Oops! A typo, do it again. Just in case, `FORGET` from `STIB` on:

```
FORGET STIB ok
CODE STIB ok
POP BX ok
POP CX ok
XOR DX, DX ok
HERE ok
SHR BX, # 1 ok
RCL DX, # 1 ok
LOOP ok
PUSH DX ok
NEXT ok
END-CODE ok
```

This all looks like it entered correctly---at least the assembler did not burp. The proof of the pudding, however, is in the eating:

```
4 7 STIB . 14 ok
4 14 STIB . 7 ok
```

Eureka! No warts this time.

If an assembly language version of STIB were needed for linking with a BASIC or C program, some minor modifications would be necessary:

- the comments would have to be preceded with a semicolon `;` rather than Forth's traditional back-slash `\`
- the word `HERE` must be converted to a loop-label
- a standard header must be added, and the definition termination also changes.

The final result, suitable for a typical stand-alone assembler, is:

```
Code segment word public 'CODE'  ; define the code segment, assume cs:
Code public STIB                 ; allow any routine to call it
  STIB proc near                 ; reverse bit-order
  POP BX                         ; get n
  POP CX                         ; get # of iterations
  XOR DX, DX                     ; zero n'
HERE:                            ; label beginning of loop
  SHR BX, 1                      ; 0'th bit - CF, shift right
  RCL DX, 1                      ; n': shift left, CF - 0'th bit
  LOOP HERE                      ; CX=CX-1, loop if CX 0.
  PUSH DX                        ; leave result on stack
  RET                            ; return from function call
  STIB endp                      ; terminate definition
Code ends
end
```

Dr. Noble is Professor of Physics at the University of Virginia.
Of special interest to UK members with long memories will be his design for an add-on AMD 9511 32-bit FP math co-processor board for the Jupiter Ace back in 1983. The commissioning company and Jupiter Cantabs both folded about the same time. The whole thing, from design to delivery, only took 3 weeks.

Alan J M Wenham
01932 786440
101745.3615@compuserve.com

Vierte Dimension 2/01

Alan Wenham

Alan provides a look at the latest issue of the German FIG magazine. To borrow a copy or to arrange for a translation of an individual article, please call Alan.

General

Advertisements for both FIG UK and the Dutch Forth group appear in this issue.

Two letters appear from Ulrich Paul, the first of which discusses whether or not Forth is broadly usable (making comparisons with C, availability of standard libraries, strengths and weaknesses). This is all valuable controversial material. The second concerns his word `REORDER` which he built ten years ago and which has found no popularity. This seeks to be a general word which replaces the various stack manipulation words `SWAP`, `DUP`, `OVER`, etc. with one word `REORDER` which knows what to do by analysis of the stack comment eg. `( aabc -- abc )`. Relevant URLs for this are http://www.paul.de/downloadables/index.htm and http://www.paul.de

Obituary for Claude Elwood Shannon

Fred Behringer

behringe@mathematik.tu-muenchen.de

Fred reviews some of the work of this great mathematician, father of switching algebra, reliability theory, and information theory.

A simple PostScript printer driver for bigForth

Bernd Beuster

bernd.beuster@epost.de

In bigForth large parts of the operating system are written in block format. Under Linux, printing of block data is difficult. PostScript data are easier to manipulate and Bernd offers a

Forth program for conversion.

Calculation with guaranteed accuracy

Christoph Poeppe

This originally appeared in September 2000 in the German version of Scientific American.

All combinations of k 1's in an n-bit word in high level Forth tion " is the scalar product. 1000 pairs of floating point numbers are Fred Behringer respectively multiplied with one another and the total added up; it is then easy to cancel out and come to a total error. An extended intermediate register, which finishes the task in fixed point arithmetic, is the answer. The author discusses the vector arithmetic co-processor XPA3233.

All combinations of k 1's in an n-bit word in high level Forth

Fred Behringer

This has already appeared in an English version in Forthwrite 111.

All's well that ends well!

Rainer Saric

Rainer.Saric@t-online.de

The author describes a control program for a process robot in the chemical industry. The robot supplier required too high a fee for development of the software and the author was able to prepare a good functional program with the aid of Forth in a short time and for much less money. His work was not rewarded by payment but the robot manufacturer took him into their employment.

Java Beans

Joerg Staben

The author reports on code recycling. Recycling not by detailed examination and changes in the original text, but visually with the mouse and drag-and-drop through the picking out of standard separately bundled components.

Reviews

Fred Behringer

behringe@mathematik.tu-muenchen.de

Fred summarises the content of Forthwrite 110 and FigLeaf 24.

Content addressable memory and Forth

Ulrich Paul

upaul@paul.de

CAM, consequently associative storage, is only about three times as expensive as usual SRAMs and therefore affordable. One specifies some contents and obtains the addresses wherein the contents reside. How can one apply this in Forth? The author presents a PostScreen language (analogous to PostScript) and makes clear that CAMS, in conjunction with Forth can be well implemented.

From the big Teich

Henry Vinerts translated by Thomas Beierlein

Henry reports on the January meeting of Silicon Valley FIG US. FIG US is not quite dead.

(ANS-Forth) Source code or (Java) component. What now?

Joerg Staben

Investment: Data expenditure, presentation of final data, calculation of expenditure period. Should one very quickly create a small program (in Forth) to do this or should one use a search engine on the Internet to find already completed components? Through ANS-Forth standardisation and with Win32Forth there is no longer a problem. However, with Java beans it may be even easier than with any Forth.

Karatsuba - Pt 1

Martin Bitter

martin.bitter@forth-ev.de

To multiply two double-length numbers together one usually needs four multiplications. Each double-length number is made up of two components and each one must be multiplied by each component of the other number. Of course there will also be a couple of additions. If one introduces specific redundancies and combines specific additive constituents, one arrives, by use of the Karatsuba algorithm, at three multiplications. Of course the number of additions is greater. For very large numbers the advantage is notable. Martin explains the relationship and sets out a Forth program.

Meta riddle. Playing around with the notion of "All"

Fred Behringer

Can this riddle be solved? An answer is expected for VD 4/2001.

“Quikwriter” Project Launch

The July issue of Forthwrite published messages on a requirement for one-handed text input from people with little finger mobility and some ideas which look promising.

Jenny Brien reports that the need is still great and the task is more than one person can manage.

I am pleased to announce that Jeremy Fowell will be trying to turn this into a collaborative project between FIG UK members, involving design, software, hardware and testing.

Jeremy’s first step is to divide the project into manageable tasks and we will use the mailing list that Graeme Dunbar organised for F11-UK to discuss the issues and keep things moving forward. The project may involve the F11-UK control board, so Jeremy will be contacting members that have bought these kits to seek their support. There will certainly be tasks too for members without kits, so bear in mind that this is a deserving cause and please volunteer your services to Jeremy (contact details at back).

Dutch Forth Users Group

Reading Dutch is easier than you might think. And as Forth is an international language, reading Dutch code is easier still for a Forth enthusiast. Are you interested? Why not subscribe to

HCC-Forth-gebruikersgroep

For only 20 guilders a year (£6.30), we will send you 5 to 6 copies of our "fig-leaf" broadsheet 'Het Vijgeblaadje' . This includes all our activities, progress reports on software and hardware projects and news of our in-house products.

To join, contact our Chairman:
Willem Ouwerkerk
Boulevard Heuvelink 126
6828 KW Arnhem, The Netherlands
E-Mail: w.ouwerkerk@kader.hobby.nl

The easiest way to pay is to post a 20 Guilder note direct to Willem.

Chris Jakeman
cjakeman@bigfoot.com

Did you Know?
– smart cards

While other parts of Forthwrite bring you all the news and the latest ideas and developments, the **Did You Know?** section highlights achievements in Forth, both recent and historical (taking care always to distinguish hearsay from attested fact).

A subset of embedded systems that Forth Inc. has tried very hard (unfortunately without significant success) to penetrate is programming smart cards. This is an extremely difficult and arcane art, but one in which unit cost and power consumption are extremely critical.

Consider program size: the cost of enlarging program memory in a card is ~$2/card (depending on a lot of factors). This means the cost of your cards is going from maybe $5 ea to $7 ea, or a 40% increase. Cards are typically issued in quantity, say, 30 million. This should get your attention!

Keycorp, an Australia-based but global company, used Forth to program its cards for a number of years. They gave it up a couple of years ago because the negative press about Forth turned off customers. We are currently working with a group at Atmel on a similar card system.

But over the same period Sun has spent ~$20M promoting Javacard, which is wildly unsuitable (requires cards costing up to ~$30 ea.) but is catching on because of the investment in promotion, Sun's generally strong reputation in the financial community, and Java's general popularity. Mere technical superiority has trouble competing with this!

Source – Elizabeth Rather, Forth Inc

Dave Pochin
01905 723037
davep@sunterr.demon.co.uk

Win32Forth Fonts

Dave Pochin

Dave Pochin continues his series on mastering Windows from Win32Forth. April's issue looked at 6 stock fonts whereas this issue considers adjustments for any font.

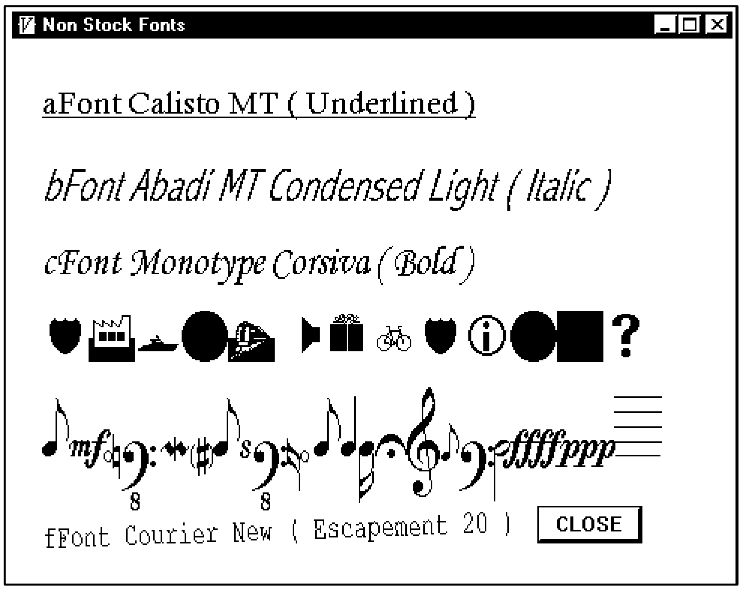
There's a change! A direct title. For me this is heavy stuff, the result of many bad Forth days, visits to the library and a fair measure of sheer stupidity.

To work!

"A font object is created like this;	`Font MyFont`
You then can change its parameters;	`TRUE Italic: MyFont`
Then you can create the font for use;	`Create: MyFont`

Once the font has been created, it must be selected into a DC (Device Context) before any characters can be displayed in the font. You can examine the demo program WINBROWS.F for an example of how to create a font."

This is a quote from one of two entries in the Win32For.prv file dated Aug 23rd 1996, so the file WINBROWS.F is an essential reference, as is the list of fonts available from the `Display/.fonts` menu item in the console window and the file FONTS.F. The listing given below is just an abstraction of the relevant parts of WINBROWS.F and all is fairly straightforward.



aFont Calisto MT (Underlined)

bFont Abadi MT Condensed Light (Italic)

cFont Monotype Corsiva (Bold)

fFont Courier New (Escapement 20)

To re-create these samples, load the code and enter DEMO

So what is the problem ? Regrettably "the devil is in the detail"; there are many unfamiliar terms to master and thirteen parameter settings to consider.

Changing the parameters by trial and error just doesn't work - well not often. There are some rules and guidelines to follow. I don't claim to have found them all or to fully understand those that I have found, but the following notes may save you a bit of time.

Whatever you do, if Windows doesn't like your font, it will substitute what is considered to be a good match.

In the `.fonts` list of Win32Forth, the column headings

```
      ht wide esc ... Cp q fp
```

relate directly to the items

```
      lfHeight lfWidth lfEscapement ... lfClipPrecision lfQuality
lfPitchAndFamily
```

in the record `LOGFONT` in file FONTS.F.

Height, Width.

`lfHeight`, `lfWidth` - Raster fonts are only scalable by multiple integers, the height up to 8x and the width up to 5x, according to the text books. But MS Sans Serif will scale by 20x, go on try it!
Raster fonts will not scale down.
True Type fonts will scale continuously.

Escapement *(esc) (0 - 3600)*

`lfEscapement` specifies the angle in tenths of a degree between the horizontal and the baseline of the printing, as in the last line of the figure.

Orientation *(ornt) (0 - 3600)*

`lfOrientation` specifies the angle in tenths of a degree each character is rotated about the line of printing, so you can invert characters by setting `lfOrientation` to 1800; at least you could if it worked ! The texts say that this parameter is ignored by TrueType fonts, but I haven't had any success with the Non-TrueType fonts either. I suspect that there is a flag that needs setting buried too deep down in Windows for me to reach. However in the file DC.F which has charge of all the printing routines, there is a font created with the orientation set to 900.

Weight *(wt) (0 - 1000)*

`lfWeight` is used to give printing effects like bold. Windows uses many pre-defined identifiers with strange names, most are easy to guess and you can often use them instead of the numerical values. E.g. `FW_BOLD` is 700 and `FW_NORMAL` is 400.

Italic, Underline and StrikeOut *(I U S) (True/False)*

`lfItalic`, `lfUnderline` and `lfStrikeOut` , apart from their normal uses, provide suitable parameters to start experimenting.

Character Set *(set)*

`lfCharSet` - The only values used in `.fonts` are 0 (`ANSI_CHARSET`), 2 (`SYMBOL_CHARSET`) and 255 (`OEM_CHARSET`). It's easy to forget to specify '2' when you want a novelty font like Symbol or CommonBullets.

OutPrecision *(p) (0 - 7)*

`lfPrecision` is used to vary the way Windows matches your font specification to those available.

ClipPrecision *(cp)*
`lfClipPrecision` determines how a character is shown when it is partially outside the area available.

Quality *(q) (0 - 2)*
`lfQuality` - All the fonts listed in `.fonts` have this value set to 1 (`DRAFT_QUALITY`). Try 0 (`DEFAULT_QUALITY`) or 2 (`PROOF_QUALITY`).

PitchAndFamily *(fp)*
`lfPitchAndFamily` is a combined parameter. A look at the listing shows that it is in hexadecimal. The low four bits set the Pitch and the upper four set the font Family. For both the Pitch and the family use either the hex values or the Windows identifiers.
The Pitch part is easy, use either `0x00` (`DEFAULT_PITCH`), or `0x01` (`FIXED_PITCH`) or `0x02` (`VARIABLE_PITCH`), and if using a TrueType font remember to add '`0x04 or`' as in the listings.

The Family part really needs referral to a textbook. Basically the characteristics of each family depend on the design of the font. In the various listings you will find most of the Windows identifiers, such as :-
`0x00 FF_DONT_CARE, 0x10 FF_ROMAN, 0x20 FF_SWISS, 0x30 FF_MODERN, 0x40 FF_SCRIPT` and `0x50 FF_DECORATIVE`.

Getting the PitchAndFamily wrong is a very good way of allowing Windows to use its best match powers. Even trying to '`OR`' all the combinations together doesn't always work either. You can always check your results by looking at the required font in another application, such as a word processor or by using the Character Map in the System Tools file.

And finally, do delete all your created fonts in either `WM_OnDone` or `WM_Close`, see listings.

```
\ FigFonts.F   Listing for 'Win32Forth Fonts'.

\ Refer to files DC.f, Fonts.f, WinBrowse.f and Win32For.Prv
\ Examples of Fonts

anew program

\ Define an Object that is a child object of the Class "Window".
:OBJECT Fontdemo <SUPER WINDOW

ButtonControl Button_1    \ Declare a button

Font aFont      \ Create a object of the class font
Font bFont     \ and another

:M SetScreenFont: ( a1 n1 -- )
          s" Impact"          SetFaceName: aFont
          s" CommonBullets"   SetFaceName: bFont
          ;M

:M SetMyFont: ( font-handle -- )
               SelectObject: dc drop
               ;M

:M ClassInit:   ( -- )              \ Things to do at the start of window creation.
             ClassInit: SUPER   \ Do anything the class needs.
                                \ set the default font type for printing
               SetScreenFont: self
                  24 Height: aFont
                  true Underline: aFont
                  VARIABLE_PITCH 0x04 or FF_SWISS or
                  PitchAndFamily: aFont

                  2 CharSet: bfont
                  30 Height: bFont
                  14 Width: bFont
                  FW_NORMAL Weight: bFont
                  VARIABLE_PITCH 0x04 or FF_MODERN or FF_DECORATIVE or
                  PitchAndFamily: bFont
                 ;M

:M ExWindowStyle: ( -- style )
                ExWindowStyle: SUPER
                ;M

:M WindowStyle: ( -- style )          \ Inherit the style from the class.
                WindowStyle: SUPER    \ See Window.f
                ;M

:M WindowTitle: ( -- title )          \ Title for the window.
                                      \ Example of Forth word z"
                z" Non Stock Fonts "
                ;M

:M StartSize:   ( -- width height )   \ Set width and height of window
                660 180               \ See Window.f
                ;M

:M StartPos:    ( -- x y )            \ Set the screen origin.
                80 100                \ See Window.f
                ;M

:M Close:       ( -- )                 \ Do anything the class needs.
                Delete: aFont          \ Delete the fonts no longer needed
                Delete: bFont
                Close: SUPER
                ;M

:M On_Init:     ( -- )                 \ Add a button. See Controls.f
                IDOK                     SetID: Button_1
                self                     Start: Button_1
                480 140 70 25            Move: Button_1
                s" CLOSE"                SetText: Button_1
                                         GetStyle: Button_1
                BS_DEFPUSHBUTTON OR
                                         SetStyle: Button_1
                Create: aFont
                Create: bFont
                ;M

:M On_Paint:  ( -- )            \ screen redraw procedure

        \ Output the first text string.
        \ Example of the Forth word s" and see the method TextOut: in dc.f
        \ Note TextOut: requires the length of the string.

         Handle: aFont  SetMyFont: self
         20 30 s" aFont AaBbCcDdEeFfGgHhIiJjKkLl"  TextOut: dc

         Handle: bFont SetMyFont: self
         20 80 s" bFont AaBbCcDdEeFfGgHhIiJjKkLl"  TextOut: dc
     ;M

:M WM_COMMAND   ( hwnd msg wparam lparam -- res )
        over LOWORD     \ fetch the identity of the Ok button which is in wParam
        case            \ case .. of .. endof .. endcase is a Forth defined
                        \ switch construction
             IDOK of    \ IDOK is the identity of Button_1
                   Close: self
                 endof
        endcase
        0 ;M


;OBJECT         \ Complete the definition of the new object.

: DEMO          ( -- )
                Start: Fontdemo ;
```

Dave provides downloadable version of this and all his Forthwrite articles at
http://www.sunterr.demon.co.uk/

FIG UK

Membership enquiries, renewals and changes of address to Doug.
Technical enquiries and anything for publication to Chris.
Borrowing requests for books, magazines and proceedings to Graeme.

FIG UK Web Site

For indexes to Forthwrite, the FIG UK Library and much more, see `http://www.fig-uk.org`

FIG UK Membership

Payment entitles you to 6 issues of Forthwrite magazine and our membership services for that period (about a year). Fees are:

National and international	£12
International served by airmail	£22
Corporate	£36 (3 copies of each issue)

Forthwrite Deliveries

Your membership number appears on your envelope label. Please quote it in correspondence to us. Look out for the message "SUBS NOW DUE" on your sixth and last issue and please complete the renewal form enclosed.
Overseas members can opt to pay the higher price for airmail delivery.

Copyright

FIG UK Services to Members

Magazine	Forthwrite is our regular magazine, which has been in publication for over 100 issues. Most of the contributions come from our own members and Chris Jakeman, the Editor, is always ready to assist new authors wishing to share their experiences of the Forth world.
Library	Our library provides a service unmatched by any other FIG chapter. Not only are all the major books available, but also conference proceedings, back-issues of Forthwrite and also of the magazine of International FIG, Forth Dimensions. The price of a loan is simply the cost of postage out and back.
Web Site	Jenny Brien maintains our web site at http://www.fig-uk.org. She publishes details of FIG UK projects, a regularly-updated Forth News report, indexes to the Forthwrite magazine and the library as well as specialist contributions such as “Build Your Own Forth” and links to other sites. Don’t forget to check out the “FIG UK Hall of Fame”.
IRC	Software for accessing Internet Relay Chat is free and easy to use. FIG UK members (and a few others too) get together on the #FIG UK channel every month. Check Forthwrite for details.
Members	The members are our greatest asset. If you have a problem, don’t struggle in silence - someone will always be able to help. Do consider joining one of our joint projects. Undertaken by informal groups of members, these are very successful and an excellent way to gain both experience and good friends.
Beyond the UK	FIG UK has links with International FIG, the German Forth-Gesellschaft and the Dutch Forth Users Group. Some of our members have multiple memberships and we report progress and special events. FIG UK has attracted a core of overseas members; please ask if you want an accelerated postal delivery for your Forthwrite.