ISSN 0265-5195

Forthwrite

January
2002
Issue 115

news events people reviews projects programming

JenX Revisited - A Simple XML Parser

FIGUK magazine:

The End of the Line

The Semantic Web

From the ‘Net - a Non-English View

A Call to Assembly 3/3

A Safer Mini-OOF

Across the Big Teich

Forthwrite Index

January
2002
Issue 115

events

news

reviews

programming

people

Editorial

As usual, this first issue of the year contains a cumulative index to Forthwrite. You will find 12 years of contributions here but new ideas and requests continue to arrive. Jenny's SERVANT concept deserves study (in JenX Revisited) and the Letters section reveals a lack of tutorial material on lists.

At this time, we invite your nominations for the Year 2001 Awards. This is a chance to show your appreciation, so please consider your choice with care.

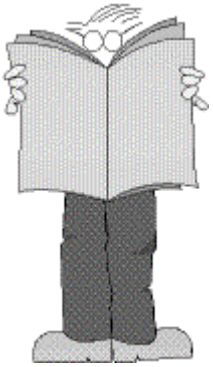
We are pleased to publish our first piece from Henry Vinerts. Henry has been reporting the activities of Silicon Valley FIG for many years. (Last month's meeting was attended by Chuck Moore, Dr.Ting and Neil Bawd – familiar names to Forth users.) We are grateful to Vierte Dimension for granting permission to use Henry's material.

Look out for details of the forthcoming events this year - euroFORTH 2002 and the German FIG Conference.

PS. Don't forget the monthly IRC session. Our next one is Saturday 2nd February on the IRC server IRCNet, channel #FIGUK from 9:00pm.

Until next time, keep on Forthing,

Chris Jakeman



Forth News

Events

euroFORTH 2002

The 2002 event has been provisionally arranged for September in Austria at the Vienna University of Technology.

Non-commercial Systems

New Release for FICL

John Sadler has announced versions 3.01 and 3.02 of this much-respected system. These provide small improvements and bug-fixes. The Forth-Inspired Command Language (FICL) is written in portable C and provides a convenient interactive command line for mainstream platforms, including Windows, and also for specialist platforms.

For more information, see

http://sourceforge.net/projects/ficl/

CGI Scripting

Saul Scudder has made an example of a web-server CGI scripting program. It is free for non-commercial use and runs under Apache for Windows. This is an object-oriented Forth and string variables defined to capture the environment from Apache.

See http://arizona.speedchoice.com/~scudders/Zen_Soft/

Forth Resources

FIG UK Mailing List

The mailing list for the F11-UK board and other projects has now moved to Yahoo at:

http://groups.yahoo.com/group/fig-forth-uk/

We are grateful to Graeme Dunbar and the School of Electronic and Electrical Engineering, The Robert Gordon University, Aberdeen for hosting the mailing list there for several years.

Forth Primer

Hans Bezemer, author of the 4th compiler, reports that the site of the Free Forth Primer Project has changed. to:

http://www.xs4all.nl/~thebeez/ForthPrimer

It remains available from

http://forthprimer.siteaddr.com

but this includes irritating banners.

Neil Bawd's Home Page

This site includes some valuable Forth sources (over 30 items). Macros are used in very powerful ways and there is also a web-publishing system for Forth code. Neil has now added the tools "Alphabetic List" and "Case-insensitive Compare". See

http://home.earthlink.net/~neilbawd/

01905 723037
davep@sunterr.demon.co.uk

The End of the Line

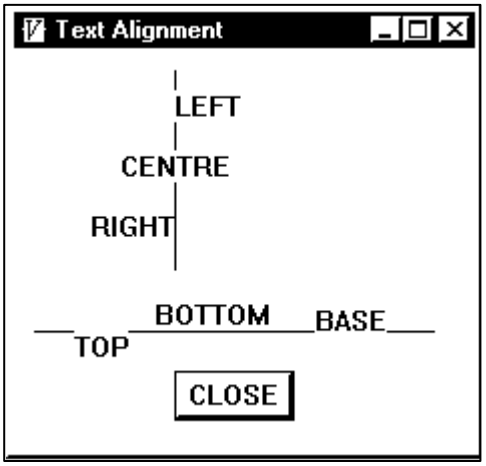
Dave Pochin

Dave has been sharing his discoveries on the use of Win32Forth to tame the Windows monster for an amazing 3 years. In many cases, he has produced examples which cannot be found anywhere else This is probably the last of the series as he moves on to concentrate more on Forth applications. Material supporting this series can be found at his web site http://www.sunterr.demon.co.uk/

When I first downloaded Win32Forth I was overwhelmed by the complexity of some of the example programs. Simple windows and printing seemed fairly easy, but it was essential to extract other simple routines to get familiar with the tools available before trying to tackle any serious project. In time I have slowly built up a series of little test routines, some more successful than others, and some now abandoned and replaced with simpler methods.

Most of these problems have been solved and following the larger examples is a little easier. Of course, I am still finding many little treasures in Windows and the command `SetTextAlign` is one I wish I'd found earlier.

The `SetTextAlign` command is usually described in the Windows texts as `SetTextAlign ( hdc, mode )`, where `hdc` is the device handle and `mode` is the parameter that controls the alignment of the text.



Following the Win32Forth practice of reversing the Windows parameters, the listing below uses the form `mode hdc call SetTextAlign`. Where the mode may be one of `TA_LEFT`, `TA_RIGHT` or `TA_CENTER` to control horizontal alignment or one of `TA_TOP`, `TA_BOTTOM` or `TA_BASELINE` to control the vertical alignment.

When the `SetTextAlign` command is followed by a text output method such as `TextOut: ( x y addr len )` from the file dc.f , the `x` parameter may be the position of the start, or the end, or the centre of the string according to the horizontal mode specified. The vertical alignment parameters work in a similar way.

In the listing that follows, this routine appears in lines like:

```
TA_LEFT GetHandle: dc Call SetTextAlign drop
80 20 s" LEFT" TextOut: dc
```

The default settings are `TA_LEFT` and `TA_TOP`. There are two other parameters available `TA_NOUPDATECP` and `TA_UPDATECP` listed in the Windows texts.

The only use of `SetTextAlign` I have found in Win32Forth is in the basic window class `Generic-Window` in the file Generic.f as part of the method `SetDlgItemAlign:`

I'm sure I could go on and on and on finding many more little snippets of Windows usage within Win32Forth, but the time has come to do some real work and, as you see from the figure, `SetTextAlign` has been a great help when labeling graphs.

Hopefully, these tales of 'daring do' within Win32Forth have helped other beginners. I still find it hard work sometimes, but not quite so frightening with the help of a good Windows API text to show the way.

```
:Object TextAlign  <Super Window

ButtonControl Button_1          \ a button

:M WindowStyle: ( -- style )
                WindowStyle: super
               ;M

:M WindowTitle: ( -- title )
                z" Text Alignment"
                ;M

:M StartSize:   ( -- w h )       \ the width and height of our window
                230 200
                ;M

:M StartPos:    ( -- x y )       \ the screen origin of our window
                10 10
;M


:M SetLines:
     get-dc
      80  10 MoveTo: dc
      80 110 LineTo: dc
      10 140 MoveTo: dc
     210 140 LineTo: dc
     release-dc
;M

:M PrintText:

       TA_LEFT GetHandle: dc Call SetTextAlign drop
     80 20 s" LEFT" TextOut: dc
```

```
      TA_CENTER GetHandle: dc Call SetTextAlign drop
    80 50 s" CENTRE" TextOut: dc

      TA_RIGHT GetHandle: dc Call SetTextAlign drop
    80 80  s" RIGHT" TextOut: dc

      TA_TOP GetHandle: dc Call SetTextAlign drop
    30 140  s" TOP" TextOut: dc

      TA_BOTTOM GetHandle: dc Call SetTextAlign drop
    70 140  s" BOTTOM" TextOut: dc

      TA_BASELINE GetHandle: dc Call SetTextAlign drop
    150 140  s" BASE" TextOut: dc

    \ Reset Default Alignment
      TA_LEFT GetHandle: dc Call SetTextAlign drop
;M

:M On_Paint:
           SetLines: self
           PrintText: self
;M

:M On_Init:  ( -- )      \ things to do at the start of window creation
             On_Init: super          \ do anything superclass needs
                IDOK                SetID: Button_1
                self                Start: Button_1
                80 160 60 25         Move: Button_1
                s" CLOSE"         SetText: Button_1
                                 GetStyle: Button_1
                BS_DEFPUSHBUTTON OR
                                 SetStyle: Button_1
;M

:M On_Done:  ( -- )          \ things to do before program termination
             On_Done: super  \ then do things superclass needs
;M

:M WM_COMMAND   ( hwnd msg wparam lparam -- res )
        OVER LOWORD ( Id )
        CASE
                IDOK OF
                         Close: self
                     ENDOF
        ENDCASE
        0
;M
;Object

: DEMO    ( -- )                  \ start running the demo program
          Start: TextAlign ;
```

From the 'Net - a non-English view

Michael Gassanenko

Have you ever thought how standard Forth words appear to people who's first language is not English? Some words are confusing, some seem comic or meaningless and even offensive (see below).

Marcel Hendrix asked on comp.lang.forth from for comments on learning Forth when "you don't naturally understand what the words mean?" Many thanks to Michael Gassanenko for sharing his reply, printed below.

1. You believe that `ALLOT` is an abbreviation of `ALLOcaTe`.
2. You confuse `QUIT` with `QUERY` and can pronounce neither. (koo-oo-ye... pfui!)
3. `CHAR` gets pronounced as "tschar" (churr)
4. You dislike long words (`SWAP` is meaningless but short, `VARIABLE` is meaningless and long, and could be `VAR`, `CONSTANT` is meaningful but long, and could be `CONST`).
5. When `HEX` is read in Cyrillic, it may be considered as the beginning of a dirty phrase (of 5 letters) meaning "no reason". When a guy tells you that `HEX` at the start of his program means "hexadecimal", you listen to him and think that a more decent sort of man would leave more letters, and that the joke is just silly.

   Two minutes later you forget the end of the word beginning with "hex".
6. All these "GN" (as in `ALIGN`) and "TIONS" (as in `DEFINITIONS`) are tongue-breakers.
7. You try to invent a way to pronounce "y" differently from "i".
8. Somebody tells you that you pronounce everything wrongly, for example, `SWAP` must be pronounced as "swaep" (swep). You do not follow

   this advice because you are used to calling it "swap" (svupp).

9. Each time you see a word like 'throughput' you remember that `THRU` is miss-spelled. You dislike that word.
10. One day someone says that logical “f” is from English 'false', the word means "a lie". That someone pronounces the word very naturally. You are familiar with this word but never tried to read its transcription in the dictionary. You do not really believe him, that it indeed pronounces as "fols", but since then avoid pronouncing any English words in his presence.
11. You give up trying to understand why `DO` and `BEGIN` mean iteration.
12. you understand `ALLOTTABLE` as ALLOT-TABLE .
13. You know that you would not dare to include a word[1] like ANS Forth’s 6.1.0670 ***** into a programming language.
14. You cannot understand the word `ENCLOSE`, neither its name nor its definition can help.
15. Sometimes the operating system switches the code page to that of your native language. It's so stupid...

    Soon you learn by heart that `PYKY` ([give me your/ don't damage the] hand) stands for `HERE`.
16. After you learn to pronounce “th”, you meet a guy that does not understand you, so you have to pronounce `THEN` as tkhyen for him.
17. Just like anyone else in the world, you write software with no means to recode text typed in in the wrong code page, although this should be easy to do. Each time you step on this rake you believe that this will never happen again, neither with you nor with your users.

[1] As an English speaker, I am so used to ABORT being used in a non-biological sense that I use it without thinking of its other connotations. Non-English speakers may not have this convenient amnesia. Now I won’t be able to use the word again without thinking of the offence I might be causing !

The Semantic Web

Chris Jakeman

The start of a new year is an appropriate time to look ahead and Forth users are nothing if not pioneers, always interested in finding better ways to do things. Although this item is not strictly about Forth at all, it looks ahead to potential developments that might involve Forth. Whether they do is, of course, up to Forth practioners like you and me.

The World Wide Web

In the May 2001 issue of Scientific American magazine, Tim Berners-Lee co-wrote an article called "The Semantic Web" [2].

Most of the World Wide Web carries information which is human-readable. Programs to process the information in web-pages currently have limited success. For example, although search engines are more effective than anyone originally expected, the collecting of information is best described as a "hunter-gatherer" activity.

The HTML mark-up in each web page provides formatting information and XML mark-up is being used more and more to provide structure. Unfortunately, the XML mark-up doesn't provide the meaning that programs need to process the information that can be gathered from the Web.

For example, "`<PARTNER>Mrs. Jakeman</PARTNER>`" is valid XML but progams can process it only if we all agree what "partner" means (co-owner?, colleague, marital?, unmarried?).

When he invented the Web in 1989, Berners-Lee intended it to carry more semantics than has become common practice.

If we could find a way for programs to understand the content of the material on the Internet, then they could do a much better job for us. For example, the task of arranging travel to a meeting with you in London - requires an understanding of calendars; mine, yours and the rail company's too.

"The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation." - Tim Berners-Lee

The leading contender for declaring the meaning of Internet material is the Resource Description Framework (RDF [3]), a standard for data about data which operates by declaring the relationships between entities. RDF is mostly written using

---

[2] http://www.scientificamerican.com/2001/0501issue/0501berners-lee.html#further

[3] See FAQ at http://www.w3.org/RDF/FAQ

XML markup and each entity is identified by its URI [4].

The relationships are named and can be simple:

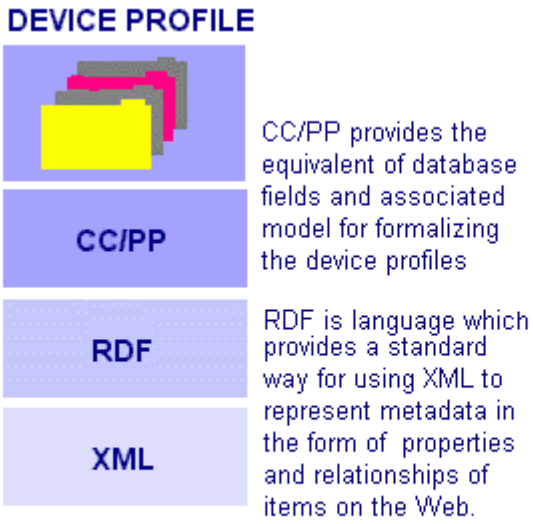
www.fig-uk.org/index.html ***has author*** *Chrs Jakeman*

or more complex:

Chris Jakeman ***has relation to*** *FIG UK, type=Officer, value=editor*

but each relationship is also given a URI. In this way, a program can discover a network of relationships for any entity.

As a final step, these relationships and inference rules between them can be stored in a publicy-accessible RDF Schema or "ontology".Berner's Lee's article explains the value of an ontology with a good example but working systems on the Internet are still hard to come by. RDF, however, is now well-specified and in use.



The device profile and user preferences might be stored in a CC/PP repository. CC/PP is in turn an RDF application.

This notion of discovery is the basis of current efforts to develop useful software agents. In our travel agent example, the program could discover all the services that will get me to London in time for that meeting, find the most suitable one, find out which of my credit cards is creditworthy and then buy the ticket.

Forth and the Semantic Web

How is this related to Forth and small systems? The very successful Open Firmware standard helps computers discover the abilities of peripherals attached to them and load drivers to work with them. In a similar vein, Berners-Lee reports the publication of CC/PP [5], a new standard for interrogating devices, eg cell-phones, to guide the adaptation of content presented to that device.

For example, if a web-server knows the size of the display screen, it can modify its pages to suit. And because CC/PP uses RDF, it is not fixed but readily expandable to cope with features not yet conceived.

As devices become smarter, they will need to find each other, discover what capabilities are available and collaborate to work together. RDF will be at the centre of this work. Maybe Forth will too.

[4] URI or uniform resource identifier. The familiar URL is just a link to a URI.

[5] Composite Capability/Preference Profiles, see http://www.w3c.rl.ac.uk/newsletters/01mar.html

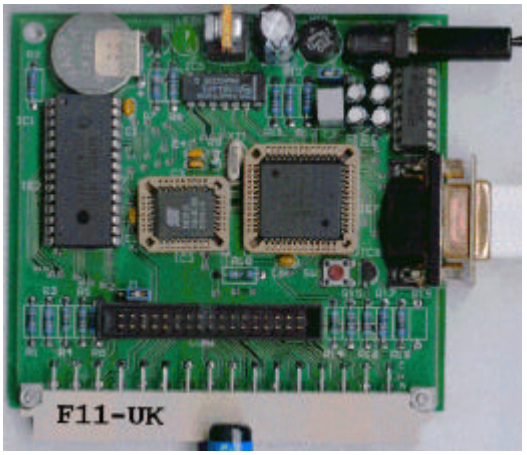
F11-UK

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

jennybrien@bmallard.swinternet.co.uk

JenX Re-visited
- A Simple XML Parser

Jenny Brien

Jenny presented a paper to the November euroFORTH entitled "Treating Data as Source" which was previewed in the July and September issues of Forthwrite. This article re-visits the JenX parser from the July issue as it has been improved substantially in the paper. It also introduces an original and novel construct, SERVANT.

XML files invariably start with "<?xml" - so that's the word that will do the actual parsing. <?xml reads tags delimited, as in HMTL, by "<" and "**>**" and passes them to a one-shot text interpreter that decides what to do with them, ignoring any that it does not recognise[6]. The definition of JenX itself is therefore quite simple. (As one XML file may refer to other XML files, JenX is defined using >R ... R> to make it re-entrant - Ed.):

```
VALUE DOTAG \ holds the execution token (xt) of the one-shot interpreter

: JENX        \ xt ++ ; parse an XML file using this interpreter
    dotag >R  TO dotag  INCLUDE  >R TO dotag
```

<?XML makes use of two "stackpads" on which strings are stacked temporarily. One, TAGNAME, is used for the tag-names which are passed to DOTAG, and the other, SCRATCH, holds any text being processed.

The One-Shot Text Interpreter

A one-shot text interpreter takes a string and performs one action based on the contents of that string, or a common default action if the string is not recognised. It may take the form of a CASE statement but, where the string is a simple word, the actions may be defined in a wordlist and a SERVANT may be used.

```
: SERVANT   \  wid xt ++ ;  defining word for one-shot text interpreters
          CREATE , ,
    DOES>         \ ca u -- ?   ; do associated action
          >R  2DUP R@ CELL+ @
                    SEARCH-WORDLIST IF
            NIP NIP R> DROP
            EXECUTE ELSE
          R> @ EXECUTE THEN  ;
```

[6] In HTML unknown tags are ignored whereas, in standard XML, the reverse is the case. Ed.

Since the default action (supplied by the xt) still has the string on the stack, it can itself be a servant word, and so servants can be stacked in a hierarchy.

E.g.
```
        WORDLIST WAITER'S ....
waiter's ' 2DROP  servant WAITER
        WORDLIST HEADWAITER'S ....
headwaiter's ' waiter's servant HEADWAITER

: CREATION  \ wid -- ; CREATE a word on this wordlist
  GET-CURRENT SWAP SET-CURRENT  CREATE  SET-CURRENT ;

: DEF: \ wid -- ; DEFINE a word on this wordlist
     GET-CURRENT SWAP SET-CURRENT  :  SET-CURRENT ;
```

A Servant Example – dealing with XML entities

(In XML, five special characters known as entities, eg. "<", have a special meaning, so they must be represented in some other way. XML uses ">" to represent "<". Any character can also be specified using its numerical code in decimal or hexadecimal, so "A" can be represented by "A" and also by "A". JenX includes the servant DENT (for defined entity) to place the decoded character on the stackpad. Ed.)

```
WORDLIST CONSTANT ENTITY?

: CENTITY       \ c ++ ; defining
    \ word for single character entities
ENTITY? CREATION
  C,            \ store the replacement
                \ character
  DOES>         \ -- append to
                \ scratch stackpad
  C@ scratch c+ ;

  CHAR < CENTITY &LT
  CHAR > CENTITY &GT
  CHAR ' CENTITY &APOS
  CHAR " CENTITY &QUOT
  CHAR & CENTITY &AMP
```

Words for using string stackpads

```
: STACKPAD   \ u -- ; create a stackpad to hold up
                    \ to u chars
    CREATE
    HERE CELL+ ,    \ pointer to top of stack
    0 ,             \ length of top string
    ALLOT ;

: SEMPTY   \ spad -- ; empty pad completely
    DUP CELL+ DUP ROT !  0 SWAP ! ;

: SPUSH    \ ca u spad -- ; push string onto stack
    SWAP >R
    TUCK @ CELL+ R@ MOVE
    R@  CHARS  CELL+ OVER +!
    R> SWAP @ ! ;

: S1      \ spad -- ca u ; top string on stack
    @ DUP @ TUCK - SWAP ;

: SDROP    \ spad -- ; drop top string from stack
    DUP CELL+ OVER @  U< IF
    DUP @ @ CHARS CELL+   \ length of top
                          \ string + count
    NEGATE OVER +! THEN DROP ;

: SNEW \ spad -- ; push a zero length string
  1 CELLS OVER +!   0 SWAP @ ! ;
```

```
: #>C    \ ca u -- c ; from Leo Wong
         \ convert from ddd or xhhh
         \ to char
  BASE @ >R
  OVER C@ DUP  [CHAR] x  = SWAP
  [CHAR] X  = OR IF
     1 /STRING HEX  ELSE DECIMAL
  THEN  EVALUATE
  R> BASE ! ;

: UnknownEntity \ ca u -- ;
  \ try for digits, else append string
  OVER C@  [CHAR] #  = IF
  1 /STRING #>C scratch c+ ELSE
  scratch s+ THEN ;
```

```
: S+  \ ca u spad -- ; concat with top string
      DUP @  @  >R \ save length of top string
      SWAP >R      \ length of additional string
      TUCK @ R@ MOVE
      R@ CHARS OVER +!
      2R> + SWAP @ ! ;

: C+ \ char spad -- ; append to top string
     DUP @ @ >R
     TUCK @ C!
     1 CHARS OVER +!
     R> 1+ SWAP @ ! ;
```

Not previously published but similar to J.Brien in Issue 89

```
ENTITY? ' UnknownEntity SERVANT DENT
```

Further defining words can be added later to deal with string substitutions and file inclusions. In this respect, a `SERVANT` can be seen as an extensible CASE statement. (`DENTS+` below locates any entities by finding "`&`" and "`;`" characters in an XML string and uses `DENT` to decode them. Ed.)

```
: DENTS+  \ ca u -- ;  append decoded version of string to SCRATCH
      BEGIN [CHAR] &  csplit
        scratch s+            \ append text before entity
      DUP WHILE
        [CHAR] ;  csplit dent \ append decoded entity
        1 /STRING             \ skip over ";"
      REPEAT
      2DROP ;
```

The word `csplit` used above splits a delimited string into the part before the delimiting character and the rest:

```
: CSPLIT ( ca u c -- ca' u1 ca u2 )
\ ca u2 is string before first instance of char c in ca u
```

A slightly more sophisticated version would use a SERVANT that calls DENT to deal with the Standard Entities, reserving its own wordlist for entities it defines itself by reading the XML file's DTD or schema.

How <?XML deals with tags

All handling of actual content is done by the xt supplied as a parameter to `JenX` and stored in `DoTag`. `<?XML` just repeatedly parses to the next "`<`", and places the

entire tag on the SCRATCH stackpad. DoTag is passed the address and count of this string, which will be over-written by the next tag.

```
: TILL  \ c -- flag ca u ; parse string up to char, flag false if char not found
     SOURCE NIP >IN @ - >R PARSE DUP R> = ROT ROT ;

: MACRO [7]  \ Usage: macro <name> <char> <words> <char> (by Wil Baden)
     :  char parse  postpone sliteral
     postpone evaluate  postpone immediate ;

macro NEXTLINE  " WHILE REFILL 0= UNTIL EXIT THEN"
```

When used in conjunction with TILL, the NEXTLINE macro ensures that the intervening code is applied to all input up to, but not including, the delimiting character. If the character is not found before the end of the input stream, then the remainder of the enclosing definition is not executed.

```
: NextTag  \ --  fetch and execute next tag
    Scratch snew
    BEGIN  [char] >  till
       dents+ \ some tags may contain entities - fetch decoded tag to Scratch
    nextline
    Scratch spop doTag EXECUTE ;

: <?xml  ( -- )
    BEGIN
      BEGIN  [char] <  till 2DROP  nextline
      NextTag
    AGAIN ;
```

<?XML ends when NEXTLINE fails – that is, once input from the file has been exhausted – and returns control to JenX.

Recognising valid Tagnames

For some simple XML files, the decoded tag may always be a simple tag name (eg. "<chapter>") , and the function in DoTag need be nothing more than a SERVANT. Each tag's action is described by a normal Forth word of the same name . This can be the case even for more complex files, if the only tags you want DoTag to act on are simple ones. In all cases, the decoded tag will be overwritten by any word called by DoTag which itself uses the Scratch stackpad, if not by the next execution of NextTag.

There are two other cases which you may need to deal with.

---

[7] Wil Baden has written extensively about the convenience of using macros in Forth. See his article in Forth Dimensions July 97 (available for loan from FIG UK Library) - Ed.

Tags with attribute lists

(Eg. "<chapter language="English">") In this case the tagname is invariably followed by white space. `DoTag` may call `WORDSPLIT` to recognise it and pass it on to a `SERVANT`.

```
: white?  ( c -- ? )  BL > 0= ;

: skip-white  \ ca u -- ca1 u1
      BEGIN DUP WHILE OVER C@ white? WHILE
         1 /STRING REPEAT THEN ;

: scan-white  \ ca u -- ca1 u1
      BEGIN DUP WHILE OVER C@ white? 0= WHILE
         1 /STRING REPEAT THEN ;

: WORDSPLIT \ ca u -- ca1 u1 ca2 u2 ; remaining-string first-word
     skip-white DUP >R scan-white 2DUP >R string/  ;
```

XML Processing Instructions and XML Declarations

These start with "?" and "!" respectively and, depending on the application, may need to be dealt with in a batch or individually. In this case, recognition is based on the characters which the string in `TAGNAME` *starts with* and can be checked using:

```
MACRO STARTSOF  " >R OVER R> COMPARE TRUE OF "
```

and a CASE statement of the form:

```
\ ca u from TAGNAME
OVER
  CASE
    S" pattern1" STARTSOF  2DROP action1 ENDOF
    (etc)
    \ pass TAGNAME on to WORDSPLIT or a SERVANT
  ENDCASE
```

Assume for example that you want to ignore comments (which begin with "<!--"). "<!--" does not have to be followed by a space, so defining it as a word won't work. Instead we use:

```
S" !--" startsof doComment endof
```

`doComment` must ignore everything up to "-->" The comment may span multiple lines and may enclose tags. If it does not enclose ">" (which is the most likely case) then `TAGNAME` will already contain the whole comment and we can treat it like any other unknown tag – ignore it. So check for that first.

```
: doComment  \ ca u --
```

```
      + 3 CHARS - S" -->" COMPARE IF EXIT THEN
   BEGIN  parse-area@ S" -->" SEARCH 0= WHILE
          2DROP REFILL 0= UNTIL          \ ignore lines until found or eof
      3 /STRING parse-area! ;            \ parse past -->
```

Matching tags handle content

The actual content of XML files is invariably held between matching tag pairs of the form <*name*>… </*name*>. These may be nested inside other tag pairs, so the tagname is saved for matching on the `TAGNAME` stackpad. `TAGNAME` will at any point contain, in order, the names of all active tag pairs. That allows it to be used to establish context where tags of the same name may be used by different parents.

I have made the assumption that any content in an inner tag pair without a defined handler should be treated as part of the content of the outer pair. That follows naturally from my rule "ignore any unknown tag". The opening tag accumulates content unto the `SCRATCH` stackpad, processing at will, and executing any tags it meets until the matching closing tag. The space used on `SCRATCH` is then freed for other tag pairs. The macros `TILLMATCH` and `GETALL` encapsulate this behaviour.

```
: GETNAME \ ca u -- ca' u' ; the name of the current tag
        wordsplit 2SWAP 2DROP ;

: MATCHED? \ ca u -- f ; true if current closing tag
    Getname OVER C@  [CHAR] /  <> IF 2DROP FALSE EXIT THEN
    1 /STRING DROP TagName s1 COMPARE ;

: OPENTAG \ ca u --  common opening tag initialisation – save name
    GetName Tagname spush  Scratch snew ;

: CLOSETAG \ ca u --  common closing cleanup – return content
    Tagname sdrop  Scratch spop ;

MACRO TillMatch "  opentag BEGIN BEGIN [char] < till"

MACRO GetAll " nextline parse-area@ matched? 0= WHILE
             NextTag REPEAT closetag "

: PRESERVE-SPACE  \ ca u -- ca u ; of content with space preserved
        TillMatch
        dents+                  \ copy decoded string to Scratch
    13 scratch c+               \ add cr
    GetAll   ;

: CONTENT    \ ca u -- ca u ; of content formatted in the default manner
        TillMatch
        BEGIN wordsplit dents+     \ copy decoded string word by word
```

```
        BL scratch c+
        DUP 0= UNTIL
              2DROP
    GetAll ;
```

CONTENT will be the word most commonly called when an opening tag is recognised. If the tag has an attribute list which affects processing, it must be dealt with before OPENTAG is called, or else temporarily saved elsewhere.

A Very, Very Simple JenX Application - Output Text of a HTML file

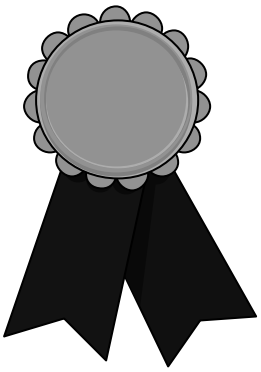
This minimal application, called simply, parses an HTML file using the <?XML parser. It recognises the section <BODY> ... </BODY> printing each line that is parsed from this section. It ignores embedded tags but prints their contents, converting XML entities and preserving white space.

simply does this by adding the word BODY to an HTML wordlist and when the HTML tag <BODY> is met, it prints the content of all tags until </BODY> is met.

The only servant defined in the HTML wordlist is HTMLTYPE – which does nothing more than tidy up the stack. Any tag attributes will therefore be ignored.

```
Wordlist HTML

HTML DEF: BODY
    TillMatch dents+  Scratch spop TYPE CR  Getall  2DROP ;

HTML ‘ 2DROP SERVANT HTMLTYPE

: SIMPLY  getname htmltype ; \ don’t bother about attributes

: <HTML> <?XML ; \ HTML files usually begin with <HTML>

‘ simply JenX filename
```

And that’s all! The application can be refined later by adding more HTML DEF:s to recognise other tags.



Nominations for the FIG UK Awards - 2001

The FIG UK Awards of 2000 were won by Keith Matthews and John Tasgal. These awards are given to encourage effort and recognise achievement.
Please take the time to look back over the past year and send in your personal nominations for 2001.

Free membership

To nominate your candidate, send in a note of who, in your opinion, most deserves an award and why. The recipient of each award will receive a place in the FIG UK web-site's Hall Of Fame, a mention in Forthwrite and ***a year's free membership***.

Achievement

The Achievement Award is given to the member who has made the best contribution towards Forth during 2001. The contribution may be a presented paper, a library of code or an idea which inspires others. Whatever form it takes, the contribution must support the goals of FIG UK.

Forthwrite

The Forthwrite Award is given to the member who has made the best contribution to Forthwrite magazine during 2001. The contribution may be judged on quality of writing, tutorial potential, entertainment value or other criteria which the Forthwrite Team deem appropriate.

The awards are judged by the officers of FIG UK. All who are members on 31st Dec. 2001 are eligible (except the judges).

Julian Noble
jvn@virginia.edu

A Call to Assembly 3/3

Julian Noble

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

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

Spherical Bessel functions

Here is an example of a fairly complex subroutine from a number-crunching application, used for calculating the effect of a 3D wave at any point. It was necessary to code this function in assembler because it was used many times.

If one only needs a single spherical Bessel function, $j_n(x)$, it is usually best just to compute it in terms of $\sin(x)$, $\cos(x)$ and polynomials in $1/x$. However, when more than one is needed, especially functions of high order, the most practical approach is recursion. The obvious method of upward recursion, based on the relation

$$j_{n-1}(x) = (2n+1)x^{-1} j_n(x) - j_{n+1}(x)$$

but, starting with explicit formulae for $j_0(x)$ and $j_1(x)$, is unstable and rapidly loses numerical precision. We therefore employ the downward recursion recommended by Abramowitz and Stegun[8], with starting values (for some large N)

$$j_N = 1, J_{N+1} = 0$$

then apply the relation

$$\sum_{k=0}^{N} (2k+1)[j_k(x)]^2 = 1$$

to obtain the normalization. In Forth this might be

```
\ data structures
10 REAL*8 #CELLS 1ARRAY JBES{            \ holds j0-j9
FVARIABLE SUM                            \ temps to off-load from fp stack
FVARIABLE X

: SETUP   ( F: x --- 0 1 )   ( --- 79)
   X DF!   79 S>F   SUM DF!
   F0.0 F1.0   79 ;
```

[8] M. Abramowitz and I.A. Stegun, Handbook of Mathematical Functions (Dover Publications, Inc., New York, 1965) p. 452.

```
: NORMALIZE
   SUM DF@ FSQRT 1/F
   10 0 DO  FDUP  JBES{ I }[9]  DUP DF@ F* DF!
   LOOP
   FDROP ;

: DO_X=0
   FDROP  F1.0  JBES{ 0 }  DF!
   10 1 DO  F0.0  JBES{ I }  DF!  LOOP ;

: ITERATE ( F: jn+1 jn --- jn jn-1) ( 2n+1 --- 2n-1)
   DUP SF FOVER F*                  ( F: jn+1 jn jn*[2n+1] )
   X DF@ F/ FROT F-                 ( F: --- jn jn-1)
   FDUP F^2                         ( F: --- jn jn-1 jn-1^2 )
   2- DUP                           ( --- 2n-1 2n-1 )
   S>F F*
   SUM DF@ F+
   SUM DF! ;

: SPHBES ( F: x --- )
   FDUP F0=
   IF DO_X=0 EXIT THEN
   SETUP  11 39 DO ITERATE  -1 +LOOP
   0 9 DO ITERATE
      FDUP  JBES{ I }  DF!
   -1 +LOOP
   DROP FDROP FDROP                  \ clean up stacks
   NORMALIZE ;
```

Translating this routine to assembler will be the piéce de resistance of this article. It is rather long, and represents the upper limit of what is reasonable to hand code as a single subroutine, in the never-ending search for speed. We shall maintain temporary values and intermediate expressions on the intrinsic stack of the floating point co-processor to minimize transfers to/from the (slower) main
memory. The public domain Forth F-PC does not come with 80x87 extensions to its assembler. Therefore to assemble and test the subroutine we must choose one of the following courses:

- add the necessary extensions to the F-PC assembler (Robert L. Smith has done this in creating the floating point extensions ffloat.seq available on various Forth archives);
- use the Micro-mini assembler described above;
- employ a Forth with a more complete assembler, such as Tom Zimmer's Win32Forth;

[9] This notation was introduced in my book Scientific Forth and has been adopted as standard for the Forth Scientific Subroutine Library Project organized by Skip Carter.

The floating point units associated with Intel microprocessors possess an intrinsic 8-deep stack[10]. Upon entering the subroutine, the on-chip stack must be initialized to contain nothing, which we visualize as

```
st(7) ...
st(6) ...
st(5) ...
st(4) ...
st(3) ...
st(2) ...
st(1) ...
st(0) ...
```

The first steps are initialization, following which the fpu stack will contain x, the argument of the Bessel function(s), as well as the initial values of j n , j n+1 and whatever else may be needed. In fact it looks like

```
st(7) ...
st(6) ...
st(5) ...
st(4) x
st(3) sum
st(2) 2n+1
st(1) jn+1
st(0) jn
```

At each subsequent iteration the stack transforms as

```
st(7) ...
st(7) ...
st(6) ...
st(6) ...
st(5) ...
st(5) ...
st(4) x      ->    st(4)   x
st(3) sum          st(3)   sum + (2n+1)*jn*jn
st(2) 2n+1         st(2)   2n-1
st(1) jn+1         st(1)   jn
st(0) jn           st(0)   jn-1
```

Let us begin with the initialization steps:

```
finit                 \ clear fpu stack
mov ecx, FSP [edi]    \ get fstack ptr
sub ecx, # B/FLOAT    \ decrement by data size
```

[10] The stack notation (87: ---) refers to the 8 - deep fpu intrinsic stack (the Intel fpu began as a separate chip with the designation 8087/80287/80387 before being combined onto the 80486 and Pentium series).

```
js L$2                        \ -> error handler
fld FSIZE FSTACK [ecx] [edi] ( 87: x)
mov FSP [edi], ecx            \ adjust fstack ptr
push ebx                      \ TOS -> mem
push # 4F                     \ put 79d=4Fh on data stack
fldz                          \ fload 0 ( 87: 0 x)
fild dword 0 [esp]            \ 79d - st(0)
fldz                          \ fload 0
fld1                          \ fld 1 ( 87: 1 0 79 0 x)
pop ebx                       \ ebx = 79 ( 87: jn jn+1 2n+1 sum x)
```

The initialization clears the floating-point unit (FPU) stack and moves x from the in-memory fstack to the FPU. (This part is taken directly from float.f's word `fpop`.) Finally, numeric constants are loaded.

Next we consider what happens during each iteration: we must pay careful attention to the FPU stack because there are 5 items on it after initialization. We note we shall need the factor $(2_{n+1}) \times j_n$ in two places: first, to calculate j_{n-1}, and second, to calculate the next term in the sum. To work out the steps, we show the fpu stack after each machine instruction:

```
      FXCH ST(1)  FLD ST(1)          FMUL ST(0), ST(3)
st(7) ...         ...                ...
st(6) ...         ...                ...
st(5) ...         x                  x
st(4) x           sum                sum
st(3) sum         2n+1               2n+1
st(2) 2n+1        jn                 jn
st(1) jn          jn+1               jn+1
st(0) jn+1        jn                 jn*(2n+1)

      FLD ST(0)   FMUL ST(0),        ST(3) FADDP ST(5), ST(0)
st(7) ...         ...                ...
st(6) x           x                  ...
st(5) sum         sum                x
st(4) 2n+1        2n+1               sum'
st(3) jn          jn                 2n+1
st(2) jn+1        jn+1               jn
st(1) jn*(2n+1)   jn*(2n+1)          jn+1
st(0) jn*(2n+1)   jn*(2n+1)*jn       jn*(2n+1)

      FDIV ST(0), ST(5) FSUBRP ST(1), ST(0) FLD1
st(7) ...         ...                ...
st(6) ...         ...                ...
st(5) x           ...                x
st(4) sum'        x                  sum'
st(3) 2n+1        sum'               2n+1
st(2) jn          2n+1               jn
st(1) jn+1        jn                 jn-1
```

```
st(0) jn*(2n+1)/x jn-1                1

FSUB ST(3),       ST(0) FSUBP ST(3), ST(0)
st(7) ...         ...
st(6) ...         ...
st(5) x           ...
st(4) sum'        x
st(3) 2n          sum'
st(2) jn          2n-1
st(1) jn-1        jn
st(0) 1           jn-1
```

That is, the complete sequence of instructions that performs one iteration is

```
   fxch   st(1)         ( 87: jn+1 jn k=2n+1 sum x)
   fld    st(1)         ( 87: jn jn+1 jn k sum x)
   fmul   st(0), st(3)  ( 87: k*jn jn+1 jn k sum x)
   fld    st(0)         ( 87: k*jn k*jn jn+1 jn k sum x)
   fmul   st(0), st(3)  ( 87: k*jn^2 k*jn jn+1 jn k sum x)
   faddp  st(5), st(0)  ( 87: k*jn jn+1 jn k sum' x)
   fsubpr st(1), st(0)  \ this is a sp. error in 486asm.f
 \ ^^^^^^ \ --- should be fsubrp, not fsubpr
   fld1
   fsub   st(3), st(0)
   fsubp  st(3), st(0)  ( 87: jn-1 jn 2n-1 sum' x)
```

Now, how can we test this to be sure it is correct? The beauty of testing an assembly language subroutine within the Forth environment is that no linking step is required. Thus we can assemble larger and larger subsets of the CODE word, testing each portion and `FORGET`ting it to test the next iteration. (Assuming, that is, that we have not caused the system to crash in one of the experiments!)

The word `ITERATE` was built in stages and tested interactively at each stage. The final stage added a `BEGIN .. UNTIL` loop. Many Forth assemblers provide macros for this purpose, but since my aim was to create a subroutine that could be ported easily to another high-level language (given the proper boiler-plate header and footer), I did not use the Forth-specific macro facilities.

Note that at the beginning of an iteration the current value of the Bessel function (not yet properly normalized, of course) gets stored in its proper array element of the array `jbes{`. This is done by computing the base address using the phrase `jbes{ 0 }` which is then added to the offsets indexed by registers ebx and edi. Note that the array index seems to be multiplied by 4 (bytes) as for 32-bit precision. However, at this storage step, the value in ebx is 2*n* because ebx has been decremented once. So in fact the subroutine is written to store 64-bit floating point numbers - vital because the magnitude of the un-normalized functions (not to mention that of the normalization sum) can grow easily past the numbers accommodated in IEEE 32-bit precision.

In fact, the first dec ebx instruction (leaving 2n in ebx) marks the beginning of the loop. The second dec ebx instruction marks the last computational step of the loop. We label the beginning of the loop with the assembler's local label facility (the phrase

`L$1:` and use the Intel jns (“jump not sign”) instruction to loop back to it when the decrement operation has not changed the algebraic sign of the index in the ebx register (that is, while 2*n*-1 $\geq$ 0).

Finally we must clean up the stacks. The exit value of the index (--1) needs to be replaced in the ebx register (which is used as the top of the data stack by Win32Forth) by whatever was on top of the stack before entering the subroutine. This is accomplished by the pop ebx instruction. Since it does not particularly matter when this is done we perform this last. The only number we wish to retain from the fpu stack is the sum so we simply pop off the top three items with three repetitions of the instruction `fstp st(0)` ; then we move the sum to the in-memory fstack (simply copying the code sequence from `fpush` for this purpose); and finally we drop x from the fpu stack with one more repetition of `fstp st(0)` .

Believe it or not, when I added this code and tested the high level word sphbes given in the listing below, it worked perfectly, first crack out of the box. The entire test sequence, including a mistake I had to correct, lasted 15-20 minutes. I do not believe MASM or TASM could come within an order
of magnitude of this time.

With the completion of the spherical Bessel function routine I end this call to assembly. Class dismissed.

Appendix

Here is the complete spherical Bessel function routine, with the assembler-coded iterative loop.

```
FALSE [IF]
Regular spherical Bessel functions j_n(x), n=0-39
(Assembly language version suitable for Win32Forth)
© J.V. Noble 1999. May be used for any purpose as long as this copyright
notice is retained.

Uses Miller's method of downward recursion, as described in
Abramowitz & Stegun, “Handbook of Mathematical Functions” 10.5 ff.

The recursion is
j(n-1) = (2n+1) j(n) / x - j(n+1)

The downward recursion is started with j40 = 0, j39 = 1 .
The resulting functions are normalized using
Sum (n=0 to inf) { (2n+1) * jn(x)^2 } = 1 .

Usage: To calculate j0-j39 say, e.g.,

   3.0e0 sphbes
To access/display a value say, e.g.,
   jbes{ 3 } F@ F. .1520516620 ok
[THEN]

marker -jbes
```

```
include arrays.f
40 long 1 dfloats 1array jbes{
FVARIABLE x
HEX

code ITERATE ( f: x --- ) \ initialization
   finit                \ clear fpu stack
   mov ecx, FSP [edi]   \ move x from fstack to st(0)
   sub ecx, # B/FLOAT
   js L$2               \ -> error handler
   fld FSIZE FSTACK [ecx] [edi] ( 87: x)
   mov FSP [edi], ecx   \ done moving
   push ebx
   push # 4F            \ 79d on data stack
   fldz                 ( 87: 0 x)
   fild dword 0 [esp]   ( 87: 79 0 x)
   fldz
   fld1                 ( 87: 1 0 79 0 x)
   pop ebx              \ ebx = 79 = 2N+1
   ( 87: jn jn+1 2n+1 sum x) \ end of initialization
L$1:
   dec ebx              \ loop begins here
   fst double jbes{ 0 } [ebx*4] [edi]
\  fwait                \ may be needed
   fxch st(1)           ( 87: jn+1 jn k=2n+1 sum x)
   fld st(1)            ( 87: jn jn+1 jn k sum x)
   fmul st(0), st(3)    ( 87: k*jn jn+1 jn k sum x)
   fld st(0)            ( 87: k*jn k*jn jn+1 jn k sum x)
   fmul st(0), st(3)    ( 87: k*jn^2 k*jn jn+1 jn k sum x)
   faddp st(5), st(0)   ( 87: k*jn jn+1 jn k sum' x)
   fdiv st(0), st(5)    ( 87: k*jn/x jn+1 jn k sum' x)
   fsubpr st(1), st(0)  \ this is a sp. error in 486asm.f
\  ^^^^^^               \ --- should be fsubrp
   fld1
   fsub st(3), st(0)
   fsubp st(3), st(0)   ( 87: jn-1 jn 2n-1 sum' x)
   dec ebx
   jns L$1              \ loop ends here
                        ( 87: j0 j1 -1 sum x)
   fstp st(0)           ( 87: j1 1 sum x)
   fstp st(0)           ( 87: 1 sum x)
   fstp st(0)           ( 87: sum x)
   mov ecx, FSP [edi]   \ sum-fstack
   fstp FSIZE FSTACK [ecx] [edi]
   fwait
   add ecx, # B/FLOAT
   mov FSP [edi], ecx
   fstp st(0)           ( 87: x --- )
   pop ebx              ( -1 --- )
```

```
   jmp L$3
L$2:
mov esi, # ' FSTKUFLO body \ error handler
add esi, edi
L$3:
next,
end-code

DECIMAL

: DO_X=0 \ handle the special case x=0
    FDROP F1.0 JBES{ 0 } DF!
    10 1 DO F0.0 JBES{ I } DF! LOOP ;

: NORMALIZE ( f: sum --- )
    FSQRT F1.0 FSWAP F/
    39 0 DO FDUP JBES{ I } DUP F@ F* F! LOOP
    FDROP ;

: SPHBES ( f: x --- )
    FDUP F0= \ x=0 ?
    IF DO_X=0 ELSE ITERATE NORMALIZE THEN ;
```

Correction

Julian Noble writes:

I detect a typo in Part I of "A Call to Assembly", on FW p. 19:

> the phrase to the left of the diagram should be
>
> ```
> 4 7  STIB . 14  ok
> 4 14  STIB . 7  ok
> ```
>
> exactly as on FW p. 22.

Chris Jakeman
01733 352373
cjakeman@bigfoot.com

A Safer Mini-OOF

Chris Jakeman

Bernd Paysan's mini-OOF is unsurpassed and remains the smallest object-oriented extension for ANS Forth. This article adds some safety features.

The mini-OOF is available from Bernd's web site[11]. It provides any ANS Forth with single inheritance, polymorphism, late and also early binding in just 12 lines. See Forthwrite Nov. '99 for a detailed exploration. More extensive packages[12], such as Anton Ertl's OOF[13] also provide data hiding, easier syntax and compile-time checking, but mini-OOF is small, simple and appropriate for applications which can benefit from a little inheritance.

I have been using mini-OOF in the construction of an XML parser and found a couple of weaknesses that this article aims to fix. An XML document contains a tree of nodes with similar but not identical properties. This makes it an obvious candidate for the object-oriented approach. The various types of node can inherit from a common ancestor but, thanks to overloading, each type can respond differently (eg. when told to print itself).

All object-oriented packages for Forth fall into one of two camps. They are either "object method" (like mini-OOF and the packages in Gforth) or "method object" (like the Neon-inspired package in Win32Forth). In the XML application, most of the objects are not named but anonymous and the methods just take the object id from the stack which suits the "object method" arrangement.

Mini-OOF is truly minimal and has no checking at all. If the method being applied to the object on the stack is not appropriate for the class of that object, then your Forth system will most likely crash instantly. After all, it's equivalent to applying `EXECUTE` to some random data.

For example, we could define a top-level class `XMLNode` for general XML nodes and then inherit a sub-class `XMLElement` which is more specialised and supports the method `.AddAttribute`. Eg:

```
XMLNode class
  method .AddAttribute
endclass XMLElement
```

We can make an object of each class using:

---

[11] Paysan's mini-OOF is at http://www.jwdt.com/~paysan/mini-oof.html

[12] Gforth includes 3 optional OOF packages. For a comparison, see http://www.delorie.com/gnu/docs/gforth/gforth_63.html

[13] Ertl's OOF can be found at http://www.delorie.com/gnu/docs/gforth/gforth_65.html

```
XMLElement  new constant anXMLElement
XMLNode     new constant anXMLNode
```

but if we apply the .AddAttribute to these objects, the first will work correctly and the second is inappropriate and will most likely crash:

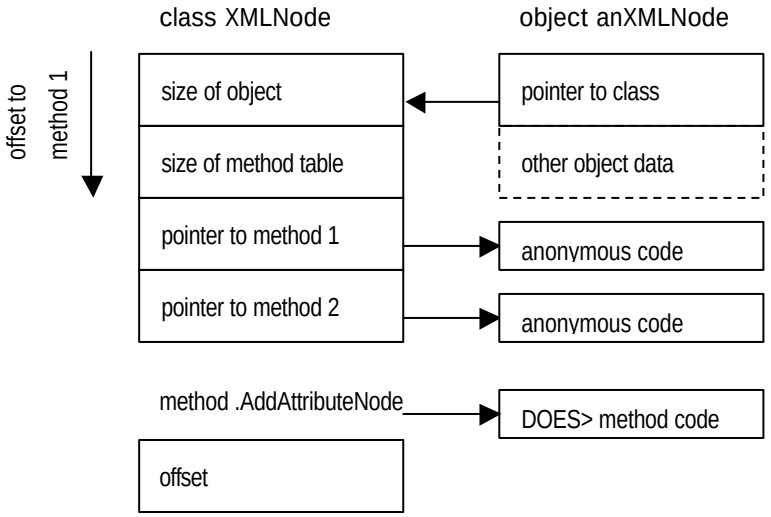
```
anXMLElement .AddAttribute  \ works fine
anXMLNode    .AddAttribute  \ will crash
```

After I fell over this several times, I extended mini-OOF with an optional run-time check inside the method to stop safely if the object is not appropriate for it. This facility can be included during development and, since it incurs a speed penalty, you may prefer to exclude it once testing is complete.

The mini-OOF is unchanged (and, being so short, is repeated here for easy reference). The additional code is a separate file/block which redefines some of the mini-OOF words. It adds a unique signature or key to each method and a copy is kept in the object's class. These two are matched whenever the method executes. You might use the two packages as:

```
include mini-oof.fth  \ From Bernd Paysan
include safer-oof.fth \ After testing, comment out for more speed
```

The mini-OOF builds a data structure like the one here.

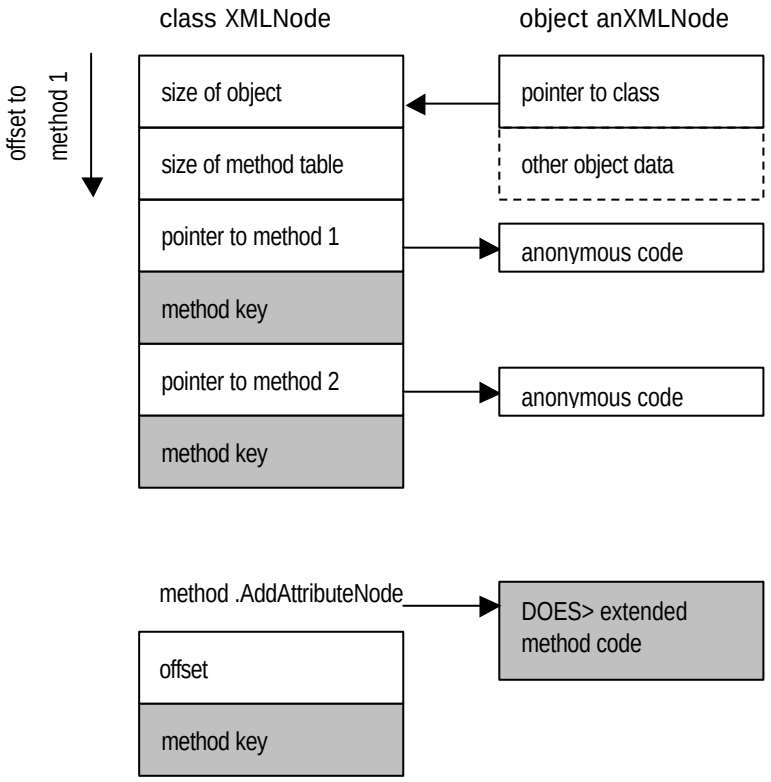


Each object points back to its class and the class holds a table of pointers to its methods (the "vtable" or despatch table). The methods themselves are built using `CREATE .. DOES>`. The `CREATE` part adding the name and saving the offset, while the `DOES>` provides common code for all methods. This code traverses from the object (on the stack) to its class, offsetting down into the method table and back across to execute the code for the current method.

My Safer-OOF builds a similar data structure as shown here but the `DOES>` part of the method provides more complex code to match the method's key with the one in the method table.

As with mini-OOF, we build a vtable containing pointers to a dummy routine (Paysan uses `NOOP` for this, but I think an `abort` is appropriate - see `UndefinedMethod` below).

Remember that the top of each vtable contains pointers to code for the methods inherited from the parent class and ends in pointers for the new methods added in this class. The `EndClass` word finishes by overwriting the top of the vtable with entries from the parent's vtable.



\ MINI-OOF.FTH from Bernd Paysan

```
: Method ( m v -- m' v ) Create  over , swap cell+ swap
  DOES> ( ... o -- ... ) @ over @ + @ execute ;
: Var ( m v size -- m v' ) Create  over , +
  DOES> ( o -- addr ) @ + ;

create object  1 cells , 2 cells ,

: Class ( class -- class selectors vars ) dup 2@ ;
: UndefinedMethod
        true abort" undefined class method called"
;
: EndClass  ( class methods vars -- )
  create  here >r , dup , 2 cells ?DO
    ['] Undefinedmethod ,
  1 cells +LOOP
  cell+ dup cell+ r> rot @ 2 cells /string move ;
: Defines ( xt class "name" -- ) ' >body @ + ! ;
: New ( class -- o )  here over @ allot swap over ! ;
: :: ( class "name" -- ) ' >body @ + @ compile, ;
```

Here is Safer-OOF which re-defines some of the mini-OOF words. The only non-obvious part is that when each method is compiled, it leaves its key on the stack. These are in the wrong sequence for EndClass which uses `roll` to extract them in the reverse sequence.

\ SAFER-OOF.FTH for debugging to check that method is appropriate for class.

```
: Class ( &Class -- &Class Key*m MethodOffset >Vars< )
    dup 2@ >r >r                    \ Save size of vars and methods
    r@ 2 cells /  1 ?do  0  loop    \ Leave a dummy key value 0 for each method
    r> r>                           \ inherited from the parent class.
;
: CheckMethod ( key1 key2 -- )
    <> abort" Method not appropriate for class"
;
: Method ( MethodOffset >Vars< -- Key NewMethodOffset >Vars< )
    create >r here                  \ Key = HERE
      swap                          \ bury key under MethodOffset
      dup , over ,                  \ compile Offset then Key
      cell+ cell+ r>                \ adjust Offset for next method
    DOES> ( ... o -- ... ) 2@       \ -- object key offset
      2 pick @                      \ -- object key offset class
      +                             \ -- object key methodPointer
      2@ >r CheckMethod r> execute
;
: EndClass  ( &Class Key*m MethodOffset >Vars< -- )
    create  here >r , dup ,
    2 cells /  2 -
    0 swap  ?DO                     \ Loop to compile keys, oldest key first
      ['] Undefinedmethod ,         \ Compile default method function
      i roll ,                    \ Add method key (1 roll = swap, 0 roll = no-op)
    -1 +LOOP
                                    \ Overwrite with contents of parent table
    cell+ dup cell+ r> rot @ 2 cells /string move
;
```

In the next issue . . .

Scripting with Forth

Did you know that Windows is fully programmable and the MS Word and Internet Explorer applications too? Any scripting language that works with the free MS Windows Scripting Host (WSH) will do the job and Microsoft themselves illustrate WSH with ForthScript, a lightweight Forth.

Jim Lawless explores scripting and describes a Forth developed for the purpose.

Henry Vinerts
Volvovid@aol.com

Across the Big Teich

Henry Vinerts

This material was prepared for Vierte Dimension by Henry Vinerts, and printed by permission of Forth Gesellschaft (German FIG)

It has been almost a week since our last SVFIG[14] meeting, and I must confess that the longer I wait, the lazier I get about writing another report. So let me "throw" a quick one at you again.

For a change, we had three speakers, but, as usual, Ting filled most of the time. The group grew from about 14 in the morning to over 20 in the afternoon. Except for some opinions about Windows XP and Microsoft in general, we did not dwell on the current subject of terrorism.

Dr. Ting started out with a call to organize a Win32Forth workgroup, to cover the next release, add better documentation, device access, etc.. It appears that Tom Zimmer wants to retire from having any responsibilities for Win32Forth. As I mentioned before, Tom left California for Texas some years ago. John Peters has been in touch with Tom via e-mail and he listed a number of ideas that a workgroup could implement to keep Win32Forth up to date.

Ting admits that the world is finally pushing him from FPC and eForth to the Windows platform, especially in his recent work in Taiwan, where he is studying various ways of inputting Forth with Chinese characters. He is also developing programs of teaching Forth to Taiwanese primary school children. He needed some help in adding sound to such programs and had found it in Doug Dillon, who came prepared to give us a lecture on how to access the sound-card related DLLs with Win32Forth, as well as with Forth Inc.'s SwiftForth.

[14] Silicon Valley Forth Interest Group

There was enough time left before a long lunch period for Ting to give us a very interesting description of the Chinese lunar calendar, which has been running steadily and unerringly for over 4000 years, whereas our Gregorian calendar is but a baby. Of course, Ting has worked out the way (on Win32Forth) to calculate the conversion of the latter to the Chinese calendar, with special emphasis on finding the correct Western date for Chinese New Year. Incidentally, Ting mentioned that there were about 30 people at the recent meeting of the Taiwan Forth Group that he attended.

I have a distinct feeling that Dr. Ting never sleeps. It is amazing how much he has produced for Forth and how varied his interests are. He concluded the day with another example of
something that had caught his attention recently and that he had found worthwhile to study and to talk to us about. That was another Forth system, the creation of a student in Australia. It is downloadable from http://pringle.sphosting.com, and that is all that I will say about it. It still seems to me that, except for Chuck Moore, every creator of his own clever and unique version of Forth will have to remain in relative obscurity and be content
with listening to his own singing or admiring his own brushstrokes. But, isn't it wonderful to labor with enthusiasm, as long as those for whom we are responsible are not running around freezing and hungry?

Mit besten Wuenschen,
Henry

German FIG
Annual Conference 2002

Dear readers of Forthwrite,

Our Annual Conference will be held on **April 19-21** 2002 at **Garmisch-Partenkirchen**, a place widely known for its skiing facilities (4th Olympic Winter Games, 1936), at the base of the Bavarian Wetterstein mountain-range; the hotel itself being 900 metres above sea level. The 2002 conference programme promises to be an interesting one with Invited Guest Speakers are Willem Ouwerkerk, chairman, and Albert Nijhof, editor, of Forth-gebruikersgroep, i.e., the Dutch FIG. Details are also announced on our website at http://www.forth-ev.de/

This year's meeting place is Forsthaus Graseck, a hotel and mountain lodge which combines facilities for both outdoor activities and seminars. A hotel-owned cable-car takes us right to the lodge, 150 metres above the town. All rooms have showers, W.C., telephone, balconies, and ISDN connections.

There are easy rail connections to Garmisch-Partenkirchen via Munich and car-parking at the cable-car station.

Before the formal conference starts, there will be a "free" day, 18th April. Depending on the weather, we will arrange a mountain walking-tour or a visit to a museum. Also within the reach of Garmisch-Partenkirchen are the most enchanting castles of the eccentric 19th-century Bavarian King Ludwig II.

The programme will leave time for ad-hoc discussions and workshops. Three prospective authors have already announced the topics of their papers: **Cross Compilation**, **Lego Writing Machine**, **Forth in OR**. If you wish to present a paper, please send an abstract before 15th March.

If you should need further information, please don't hesitate to contact us at: Heinz.Schnitter@physik.uni-muenchen.de or behringe@mathematik.tu-muenchen.de

Heinz and Ulrike Schnitter (organisers) and Fred Behringer (programme)

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.

Letters

The Magazine Team are always pleased to get feedback and encouragement. Here we have a suggestion/enquiry from Boris Fennema who is new to Forth and my response - Ed.

Boris Fennema

Hi Boris,

> Sent: 10 January 2002 09:17
>
> As a novice Forth hobbyist I am (slowly) learning Forth.
>
> I can appreciate most of its features but I fall down in OO Forth
> and moderately advanced data structures.
>
> I can see how you build data arrays but how whould you operate on
> a singly-linked list ?
>
> What I am getting at is that there are idioms in any language that
> are preferred over others. A novice -> advanced dictionary of
> Forth idioms and data structures would be very useful to me.
>
>
> just a suggestion.

A very welcome one - thanks.

There's quite a lot of Forth material on lists themselves. For example, Forth Dimensions ran a series from Neil Bawd called Stretching Standard Forth which includes Linked Lists (July 97 p20). Dick

Pountain's book Object-Oriented Forth is as much about data structures as about OOF and Chapter 3 is entirely devoted to lists.

Forth provides so much freedom that it can become seductive. I can point you to several fascinating articles about doing clever things with lists - eg. OOF classes to develop lists and trees or rings used to implement
lists, queues and sets. However I cannot find an article devoted to working with straightforward lists using ANS Forth. Neither can I find anything suitable in the on-line tutorials.

Yours is a question that deserves to have an answer so I will pass this on to Graeme Dunbar, our Librarian, and ask him to check the Library; I am thinking especially of the books from Forth Inc. (Forth Application Techniques and Forth Programmers Handbook).

If nothing turns up that fits the bill, then I'm sure we can find a member willing and able to write effectively about the topic for a future Forthwrite. In the meantime, if you have a specific problem I might be able to help you myself. What are you trying to do with your list?

Bye for now

Chris Jakeman

Voice +44 (0)1733 753489

Forth Interest Group United Kingdom
chapter at http://www.fig-uk.org

Forth News Correction

John Peters (japeters@pacbell.net) writes:
"As you can see the Win32Forth fan club http://go.to/win32forth/
was the project of Ryon Root, not me. I am working on improving Win32Forth with the members of the Silicon Valley FIG"

Forthwrite Index

Jenny Brien maintains a set of 3 indexes to Forthwrite on the FIG UK web site and updates them with each new issue. These indexes are sorted by date, by author and by subject *going back to 1990. The subject index is published in the magazine annually (below), with the new entries highlighted.

Back issues of Forthwrite may be borrowed from the Library without charge, so this is a good way to catch up on topics of special interest. If you spot a topic that has not been adequately covered, please drop a line to the Editor.

Forthwrite Subject Index 1990-2001

Subject	Author	Date	Title
algorithms	Hersom, Ed	92-10	Advanced course
algorithms	Charlton, Gordon	93-04	Backwards (psychic programming)
algorithms	Hersom, Ed	93-04	Trees & splines
algorithms	Hill, Will	93-06	Solving with Newton-Raphson
algorithms	Payne, John	93-12	Approximate pattern matching
algorithms	Bennett, Paul	94-06	Fuzz, fibs and forms
algorithms	Pochin, David	94-10	First attempts at Fuzzy Logic
algorithms	Bennett, Paul	95-06	Fractionally angular
algorithms	Charlton, Gordon	95-06	Easter Sunday
algorithms	Ramsay, Chris	99-08	Forth and Genetic Programming
applications	Green, Roedy	90-08	Abundance (database)
applications	Brien, Jack	91-02	Typing tutor (code)
applications	Kendall, Les	91-02	Terminal emulator for PC (code)
applications	Smith, Graham	91-02	Logic gates
applications	Grey, Nigel	91-06	Big Blue on the move IBM CAD (review)
applications	Franin, Julio	92-08	Torsion measurement system
applications	Stephens, Chris	93-08	Seven thousand networked micros
applications	Anderson, Joe	98-07	Forth In Space
applications	Trueblood, Mike	99-11	Radio Clock
applications	Bennett, Paul	00-08	Logging on - statistically speaking
applications	Paysan, Bernd	00-08	A Web-Server in Forth
applications	**Matthews, John**	**01-01**	**Forth as Preferred Development Environment**
applications	**Kendall, Les**	**01-01**	**XML and Forth**
applications	**Wong, Leo**	**01-04**	**Solving a Riddle**
applications	**Brien, Jenny**	**01-07**	**"Quikwriter" proposal**
applications	**Anderson, Joe**	**01-07**	**Forth for NEAR Spacecraft**
applications	**Fowell, Jeremy**	**01-09**	**"Quikwriter" Project Launch**

applications	**Brien, Jenny**	**02-01**	**JenX revisited - A Simple XML Parser**
arithmetic	Jakeman, Chris	90-12	A high-level /MOD (code)
arithmetic	Preston, Philip	91-02	Multi-cell arithmetic (code)
arithmetic	Filbey, Gil	91-04	Tutorial
arithmetic	Haley, Andrew	91-04	Function approx. by Chebyshev series
arithmetic	Filbey, Gil	91-12	Mixed point arithmetic (tutorial)
arithmetic	Payne, John	91-12	Fixed point arithmetic (word set)
arithmetic	Filbey, Gil	92-02	Mixed point arithmetic (tutorial)
arithmetic	Filbey, Gil	92-04	Mixed point arithmetic (tutorial)
arithmetic	Brown, Jack	92-10	Floored v symmetric division (tutorial)
arithmetic	Filbey, Gil	93-02	Floating point
arithmetic	Filbey, Gil	95-02	Cube roots
arithmetic	Bennett, Paul	97-02	From the 'Net - Square Roots (code)
arithmetic	Hersom, Ed	98-07	Quad (Fixed-Point) Arithmetic
arithmetic	Behringer, Fred	00-04	32-bit GCD without Division
arithmetic	Pochin, Dave	00-06	Floating Decimal Fudge
arrays	Jakeman, Chris	90-08	Arrays and records (code)
arrays	Brien, Jack	92-02	Ways with arrays (code)
assembly	Tanner, P.	96-05	Linking machine code modules with Forth
block tools	Filbey, Gil	91-02	Bits and loading blocks (tutorial)
block tools	Hainsworth, Chris	91-02	Editing blocks (tutorial)
block tools	Charlton, Gordon	94-04	One-screen library load (code)
bons mots	Bezemer, Hans	97-08	Th
bons mots	Eckert, Brad	97-08	On Off On? Off?
bons mots	Luke, Gary	97-08	Tally
bons mots	Hersom, Ed	97-11	NVars [H] [D]
bons mots	Payne, John	97-11	3rd Swap@ Sgn #>ASCII
bons mots	Wenham, Alan	97-11	Z
bons mots	Elvey, Dwight	98-01	Setting bits with MASK
bons mots	Wenham, Alan	98-01	Printing binary with .SB U1B. U2B.
bons mots	Hoyt, Ben	98-03	PLACE is to COUNT as ! is to @
bons mots	van Norman, Rick	98-03	MANY for debugging
bons mots	Wong, Leo	98-05	Laying down values with COURSE
concurrency	Charlton, Gordon	91-10	Co-routine monitors (code)
concurrency	Charlton, Gordon	94-04	One-screen concurrent Forth (code)
control flow	Charlton, Gordon	90-04	Universal delimiter (code)
control flow	Brien, Jack	91-02	Extended ANS structures (F83 code)
control flow	Bennett, Paul	91-04	High level FOR..NEXT (code)
control flow	Carpenter, R.H.S.	92-12	Flow-charting method
control flow	Preston, Philip	93-06	Shortcuts and drop-outs
control flow	Brien, Jack	94-06	Extending ANSI control structures
control flow	Brien, Jack	95-06	Portable control structures

control flow	Charlton, Gordon	95-06	Trouble with DO
control flow	Jakeman, Chris	96-05	If and begin - ANS style
database	Filbey, Gil	91-08	FIG UK database (tutorial)
database	Filbey, Gil	91-08	FIG UK database (tutorial)
design	Payne, John	90-12	Simpler Forth (comment)
design	Brien, Jack	91-10	Return stack ENTER ISNOW and aliasing
design	Thomas, Reuben	92-06	Forth lifestyle
design	Hersom, Ed	92-10	NVARS
design	Charlton, Gordon	93-04	Upside down
design	Smart, Mike	93-10	Computer Shopper Programmer's Challenge
design	Matthews, John	94-02	On his September lecture
design	Bennett, Paul	94-08	Taking exception ...
design	Hersom, Ed	94-08	Simple user interface
design	Flynn, Chris	94-10	Numerical input
design	Allwright, R.E.	95-06	Pagination
design	Jakeman, Chris	95-06	From the 'net
design	Telfer, Graham	96-05	The specification method hunt
design	Brien, Jack	99-01	Working with Wordlists
design	Brien, Jack	99-06	Handling Literals
design	Telfer, Graham	99-06	Skeletons - Designing a Recursive Application
dynamic data	Charlton, Gordon	90-04	Dynamic words (code)
dynamic data	Charlton, Gordon	94-06	Work, rest and play
editing tools	Jakeman, Chris	90-02	Search and replace 1/2 (code)
editing tools	Jakeman, Chris	90-04	Search and replace 2/2 (code)
editing tools	Lake, Mike	91-02	Full screen editor in one screen (code)
editing tools	Brien, Jack	95-06	Full screen editor
editorial	Hainsworth, Chris	91-04	Forthtalk and EuroFORML report
editorial	Jakeman, Chris	92-08	Soapbox - "Do it yourself"
editorial	Payne, John	92-12	Fat, thin or inflatable?
editorial	Wilson, R.J.	93-06	Seeing trees in the wood
editorial	Rush, Peter	95-02	Honeywell Forth Bulletin Board
editorial	Jakeman, Chris	96-05	From the 'net - perceptions
editorial	Hersom, Ed	96-07	Why Forth?
editorial	Jakeman, Chris	96-11	Sell-by-date
editorial	Jakeman, Chris	97-02	FIG UK joins the World Wide Web
editorial	Jakeman, Chris	97-02	Welcome Disk
editorial	Brien, Jack	97-08	FIG UK Web Site
encryption	Greenwood, Mike	98-03	File Encryption
exceptions	Charlton, Gordon	91-04	CATCH and THROW (code)
exceptions	Jakeman, Chris	93-10	Portable CATCH and QUIT (code)
exceptions	Jakeman, Chris	93-10	Using CATCH and QUIT (code)
FANSI project	Bennett, Paul	90-06	Time for a new FIG Forth (comment)

FANSI project	Charlton, Gordon	90-10	High-level /MOD using recursion (code)
FANSI project	Charlton, Gordon	90-10	High-level multiply (code)
FANSI project	Flynn, Chris	90-10	Discussion on REQUIRES
FANSI project	Hainsworth, Chris	90-10	FANSI that (proposal)
FANSI project	Bennett, Paul	90-12	FANSI environs (proposal)
FANSI project	Flynn, Chris	90-12	Response to design proposals (comment)
FANSI project	Payne, John	90-12	Response to design proposals (comment)
FANSI project	Charlton, Gordon	91-06	FANSI definitions (code)
FANSI project	Charlton, Gordon	91-08	FANSI bloomers (code)
FANSI project	Payne, John	91-08	Notes on FANSI (code)
FANSI project	Bennett, Paul	91-10	Report on FANSI
FANSI project	Charlton, Gordon	91-12	FANSI vocabularies (proposal)
FANSI project	Brien, Jack	92-02	FANSI (comment)
FANSI project	Payne, John	92-02	FANSI (comment)
FANSI project	Preston, Philip	92-02	FANSI (comment)
FANSI project	Payne, John	92-12	FANSI QUIT
file tools	Brien, Jack	91-02	Loading dependant source (code)
file tools	Jakeman, Chris	93-02	File access, part 1 (code)
file tools	Jakeman, Chris	93-04	File access, part 2 (code)
file tools	Jakeman, Chris	93-06	File access, part 3 (code)
file tools	Jakeman, Chris	93-08	File access, part 4 (code)
file tools	Brien, Jack	95-10	Hierarchical screen filing
file tools	Wong, Leo	98-10	ANS File Words for Pygmy Forth
file tools	Behringer, Fred	99-01	ANS File Words for Turbo Forth - 1
fractions	Charlton, Gordon	90-02	Vulgar words (code)
fractions	Wilson, R.J.	90-04	Rational numbers (code)
fractions	Wilson, R.J.	90-06	Transcendental rationale (code)
fractions	Charlton, Gordon	90-10	Rational approximation (comment)
futures	Jakeman, Chris	94-08	Telescript (comment)
futures	Jakeman, Chris	94-10	Some future directions (editorial)
futures	Jakeman, Chris	96-11	Forth and Java (comp.lang.forth)
futures	Pelc, Stephen	99-11	FIG UK - The Next 20 Years
futures	**Jakeman, Chris**	**02-01**	**The Semantic Web**
graphics	Filbey, Gil	90-04	Plotting spirals (tutorial)
graphics	Charlton, Gordon	92-06	Turtle graphics
graphics	Payne, John	92-08	Flood fill
graphics	Charlton, Gordon	93-08	Drawing a line
graphics	Charlton, Gordon	93-10	Not drawing a line
graphics	Payne, John	93-10	How Bresenham's line drawing alg. works
graphics	Pochin, Dave	00-11	"BLT is not a Sandwich"
hardware	Koopman, Philip	90-10	RTX 4000 (publicity)
hardware	Fowell, Jeremy	92-08	P20 chip, part 1/2

hardware	Fowell, Jeremy	92-10	P20 chip, part 2/2
hardware	Bennett, Paul	96-07	Chuck's chips
hardware	Fowell, Jeremy	99-01	FIG UK Hardware Project
hardware	Fowell, Jeremy	99-04	FIG UK Hardware Project - Progress
hardware	Heuvel, Leendert	99-04	The 'Egel Coursebook
hardware	Fowell, Jeremy	99-08	FIG UK Hardware Project - Progress
hardware	Fowell, Jeremy	99-11	FIG UK Hardware Project - Progress
hardware	Fowell, Jeremy	00-01	F11-UK Hardware Project - Progress
hardware	Fowell, Jeremy	00-04	F11-UK Hardware Project - Progress
hardware	Fowell, Jeremy	00-08	F11-UK Hardware Project - Launch
hardware	**Jakeman, Chris**	**01-01**	**F11-UK Hardware Project - Progress**
hardware	**Jakeman, Chris**	**01-04**	**F11-UK Hardware Project - Progress**
history	Rather, Elizabeth	95-04	The evolution of Forth
history	Rather, Elizabeth	95-12	The Forth approach to operating systems
history	Hainsworth, Chris	99-01	Forthwrite Issue No. 1 revisited
history	Powell, Bill	99-01	The Birth of FIG UK
history	Behringer, Fred	99-11	Swap Dragon
history	Brien, Jack	99-11	FIG UK - The Last 20 Years
history	Jakeman, Chris	00-01	Did you Know? - EasyWriter
history	Jakeman, Chris	00-04	From the 'Net, Forth for Novell
history	Crook, Neal	00-06	The Canon Cat
history	Jakeman, Chris	00-06	Did you Know? - Forth OS
history	Jakeman, Chris	00-08	Computer Conservation
history	Jakeman, Chris	00-08	Did you Know? - Forth v C
history	Jakeman, Chris	00-11	Did you Know? - Open Firmware
history	**Jakeman, Chris**	**01-09**	**Did you Know? - smart cards**
history	**Jakeman, Chris**	**01-11**	**Did you Know? - large Forth projects**
humour	Payne, John	90-12	A program that works the French way
humour	Smith, Graham	95-06	Book titles
humour	Jakeman, Chris	96-05	From the 'net - a drinking song
humour	Allwright, Ray	98-05	A Story of Cowboys
humour	**Gassanenko, Michael**	**02-01**	**From the 'Net - the non-English view**
interfacing	Robinson, Dave	91-08	Mouse handling (F83 code)
interfacing	Bennett, Paul	98-05	Reading the World - 1
interfacing	Bennett, Paul	98-07	Reading the World - 2
interfacing	Bennett, Paul	98-10	Writing the World - 1
interfacing	Bennett, Paul	99-01	Writing the World - 2
internals	Hainsworth, Chris	90-02	Kiss and run (exploring F-PC)
internals	Charlton, Gordon	91-02	A replacement for DO .. LOOP (code)
internals	Flynn, Chris	91-06	Forth engine on 68000
internals	Bennett, Paul	92-10	Top-down development of a Forth system

internals	Bennett, Paul	93-04	QUIT, the story continues...
internals	Preston, Philip	93-12	RatForth - ANS on F83
internals	Preston, Philip	94-02	Ratforth revised etc.
internals	Preston, Philip	94-06	Redefining colon
internals	Preston, Philip	94-10	Simulating EVALUATE
internals	Preston, Philip	95-10	Variables, values & locals
internals	Wenham, Alan	95-12	Meandering Forth
internals	Brien, Jack	97-08	Building a new inner interpreter
internals	Allwright, Ray	98-03	From the 'Net - Minimal word sets
internals	Allwright, Ray	99-04	From the 'Net - Turnkey Apps and Docs
internals	Tasgal, John	00-04	An Introduction to Machine Forth
internals	**Brien, Jenny**	**01-09**	**Treating Data as Source**
interpreting	Jakeman, Chris	95-10	From the 'net - text interpreter
interpreting	Brien, Jack	96-11	Implementing an outer interpreter
interview	Moore, Charles	99-06	1xForth
interview	**Lawless, Jim**	**01-11**	**An interview with Tom Zimmer**
interview	**Morrison, George**	**01-11**	**Charles Moore interview on Slashdot**
library	Hainsworth, Sylvia	91-04	FIG UK library bulletin
library	Jakeman, Chris	96-11	Library assets
library	Hainsworth, Sylvia	98-05	Purchases and current publications
logic	**Behringer, Fred**	**01-07**	**Arithmetized Logic in Forth**
MCFAs	Brien, Jack	90-08	Comment
objects	Jakeman, Chris	94-12	Objects and so forth
objects	Jakeman, Chris	98-11	OOF - A Minimal Approach
objects	Prinz, Friederich	99-01	Counting Fruits the Classic Way
objects	**Jakeman, Chris**	**02-01**	**A Safer Mini-OOF**
performance	Jakeman, Chris	98-01	From the 'Net - Speed Demons
permutations	Charlton, Gordon	90-02	Permutations, a new algorithm (code)
permutations	Hersom, Ed	91-10	Permutations (code)
permutations	Hersom, Ed	92-04	Permutations/combinations
permutations	Baden, Wil	00-11	Permutation by Transposition Sequence ACM 115A
permutations	Jakeman, Chris	00-11	Simple Forth Permutations
permutations	**Behringer, Fred**	**01-04**	**Generating Combinations**
presentation	Brien, Jack	90-02	Locals and more (discussion)
presentation	Matthews, Keith	90-12	Stack diagrams (explored)
presentation	Brien, Jack	91-02	GIST for indexing source (code)
presentation	Bennett, Paul	91-06	Manual documentation (code)
presentation	Charlton, Gordon	93-12	StackFlow
presentation	Brien, Jack	94-10	Readable Forth
presentation	Tanner, P.H.	94-12	Post indentation
presentation	Charlton, Gordon	97-02	From the 'Net - StackFlow
probability	Filbey, Gil	90-12	Probability of common birthdays

probability	Filbey, Gil	90-12	Random thoughts on probability
probability	Payne, John	90-12	Probability of common birthdays
publications	Haley, Andrew	91-12	FORML 87, 88 & 89 (review)
puzzles	Hainsworth, Chris	90-06	Forth brain teasers
puzzles	Charlton, Gordon	90-12	Name that word
puzzles	Charlton, Gordon	91-02	Puzzle answers (code)
puzzles	Filbey, Gil	92-10	Tethered goat puzzle, part 1/2
puzzles	Filbey, Gil	92-10	Tethered goat puzzle, part 2/2
random nos.	Filbey, Gil	93-06	Visualising random numbers on Apple II
random nos.	Jakeman, Chris	93-06	Random numbers
random nos.	Filbey, Gil	93-08	Testing for randomness
random nos.	Payne, John	93-08	More on random numbers
review	Charlton, Gordon	94-10	Riding the wild 'net
review	Charlton, Gordon	95-02	Report from EuroForth '94
review	Bennett, Paul	97-11	EuroForth '97 Conference
review	Wenham, Alan	98-01	Vierte Dimension
review	Fowell, Jeremy	98-05	Forth Programmers' Handbook
review	Jakeman, Chris	98-05	Genetix - The Inside Story
review	Payne, John	98-07	FORML Proceedings 94 & 95
review	Flynn, Chris	98-10	A Hard Day Garbage Collecting
review	Jakeman, Chris	98-10	jeForth
review	Bennett, Paul	98-11	euroForth '98 Conference
review	Wenham, Alan	99-01	Vierte Dimension
review	Anderson, Joe	99-06	Forth for Virtual Reality
review	Wenham, Alan	99-11	Vierte Dimension
review	Jakeman, Chris	00-01	FIG UK 20th Anniversary Reunion
review	Wenham, Alan	00-01	Vierte Dimension 4/99
review	de Ceballos, Federico	00-04	21st FORML Conference
review	Wenham, Alan	00-04	Vierte Dimension 1/00
review	Wenham, Alan	00-06	Vierte Dimension 2/00
review	Jakeman, Chris	00-08	euroForth '99 Conference
review	Wenham, Alan	00-11	Vierte Dimension 3/00
review	Jakeman, Chris	00-11	Forth in the UK
review	**Wenham, Alan**	**01-01**	**Vierte Dimension 4/00**
review	**Ives, Robert**	**01-01**	**"Forth Application Techniques"**
review	**Oakford, Howerd**	**01-01**	**euroFORTH 2000 Conference report**
review	**Jakeman, Chris**	**01-07**	**Gesellschaft 2001 Conference report**
review	**Abrahams, David**	**01-07**	**"Extreme Mindstorms" book**
review	**Bennett, Paul**	**01-07**	**3 Free Forths and an OS too!**
review	**Wenham, Alan**	**01-09**	**Vierte Dimension 2/01**
review	**Wenham, Alan**	**01-11**	**Vierte Dimension 3/01**

review	**Vinerts, Henry**	**02-01**	**Across the Big Teich**
roots	Wilson, R.J.	90-08	Root of rational numbers (code)
roots	Charlton, Gordon	90-10	Square root (code)
roots	Trapp, John	91-02	Square-roots for double/floating point
roots	Brien, Jack	97-11	From the Net - More on square roots
roots	Behringer, Fred	98-03	Square roots once more
roots	Behringer, Fred	98-05	Cubic roots without division
roots	Jakeman, Chris	00-04	Cube Roots Again
roots	Jakeman, Chris	00-04	From the 'Net - Cube Roots
roots	Jakeman, Chris	00-06	From the 'Net, Cube Roots
searching	Charlton, Gordon	90-12	A faster string search (code)
searching	Charlton, Gordon	91-10	A binary search (code)
searching	Hersom, Ed	91-12	Recursive BINSEARCH (code)
searching	Charlton, Gordon	93-02	Shift-AND string search (code)
searching	Charlton, Gordon	94-02	Best string search (code)
searching	Jakeman, Chris	95-06	Linear search
sets	Charlton, Gordon	90-06	Set manipulation (code)
sorting	Charlton, Gordon	90-08	Radix, an extravagant sort (code)
sorting	Charlton, Gordon	90-10	Sorting strings with qsort (code)
sorting	Charlton, Gordon	91-10	Heapsort (code)
stacks	Preston, Philip	92-12	Stocking fillers - stacks & write-only
stacks	Charlton, Gordon	94-04	Stacrobaticus exotica
stacks	Filbey, Gil	94-08	Stacks (tutorial)
stacks	Jakeman, Chris	95-08	Stack manipulation
stacks	Joseph, Neville	95-10	Stack manipulation
stacks	Barr, Stan	95-12	A third stack
stacks	Hersom, Ed	97-11	Multi-precision Stack Operators
standards	Jakeman, Chris	91-06	Portable code (code)
state machines	Charlton, Gordon	90-10	Variables for state machines (code)
state machines	Dunbar, Graeme	98-07	Finite State Machines 1/3
state machines	Dunbar, Graeme	98-10	Finite State Machines 2/3
state machines	Dunbar, Graeme	99-08	Finite State Machines 3a
strings	Leibniz, David	91-02	String stack routine (code)
strings	MacLean, Ruaridh	91-02	High level DIGIT (tutorial)
strings	Charlton, Gordon	91-04	A string pattern matcher (code)
strings	Payne, John	92-04	Text processing
strings	Preston, Philip	92-10	TACK and BLOCKL
strings	Charlton, Gordon	93-04	ANSI and portability - STRLIT (code)

strings	Brien, Jack	93-06	Comment on Blockl & Tack
strings	Charlton, Gordon	93-06	Similarity
strings	Jakeman, Chris	95-12	From the 'net - please
strings	Brien, Jack	96-07	String handling
strings	Jakeman, Chris	97-02	Pattern matching - 1/3 (tutorial)
strings	Jakeman, Chris	97-08	Pattern matching - 2/3 (FoSM with Forth)
strings	Jakeman, Chris	97-11	Pattern matching 3/3 (Rex)
strings	Borrell, Richard	98-03	Deferred Language Translation
strings	Oakford, Howerd	98-11	Multiple Language Programs Made Easy
structures	Brien, Jack	98-01	Building Forth Structures
systems	Green, Roedy	90-08	BBL Forth (review)
systems	Bennett, Paul	92-02	Pygmy Forth (review)
systems	Tanner, Philip	92-04	As in a glass darkly
systems	Hersom, Ed	93-02	Pocket Forth (review)
systems	Tanner, P.H.	93-06	URForth (review)
systems	Payne, John	95-02	A 32-bit Forth for Windows (review)
systems	Stephens, Chris	95-02	Forth for the Transputer (review)
systems	Behringer, Fred	97-08	Forth for the Transputer
systems	Worthington, Thom.	98-01	Aztec - A Forth For Windows '95
systems	Besemer, Hans	98-05	4th - The Alternative Compiler
systems	Jakeman, Chris	99-01	Web Forth Project
systems	Lancaster, Garry	99-04	Forth for the Z88
systems	Jakeman, Chris	99-06	Web Forth Project
systems	Ouwerkerk, Willem	99-08	ByteForth for MCS51 cpu's
systems	Tasgal, John	00-06	An Introduction to Color Forth
systems	Tasgal, John	00-06	The BMP Example
systems	**Zimmer, Tom**	**01-09**	**4-bit Forth**
systems	**Eckert, Brad**	**01-11**	**Tiny Open Firmware**
tools	Jakeman, Chris	90-06	Patch programming aid (code)
tools	Jakeman, Chris	90-10	Run-time operators (code)
tools	Preston, Philip	91-12	ALIAS ALIAS ALIAS (F83 code)
tools	Jakeman, Chris	92-12	Also and -Also (code)
tools	Charlton, Gordon	93-04	Wrong way round!
tools	Bennett, Paul	93-06	+MOD! (LOG?) and commenting words
tools	Brien, Jack	93-10	Utilities for F83 on Amstrad PCW
tools	Jakeman, Chris	93-12	Shell (code)
tools	Bennett, Paul	94-02	Spooling and browsing
tools	Jakeman, Chris	94-02	.Call and Assert (code)
tools	Jakeman, Chris	94-04	Check (code)
tools	Flynn, Chris	94-06	Conditional compilation
tools	Preston, Philip	94-08	More fun with EVALUATE

tools	Charlton, Gordon	94-12	16-bit cyclic redundancy checksums
tools	Franin, Julio	95-02	MC51 Forth debugging
tools	Smith, Graham	95-06	MARK
tools	Jakeman, Chris	95-08	Limit variables (code)
tools	Abrahams, David	95-10	General purpose utilities for F-PC
tools	Stott, Barrie	97-02	Stack checking (code)
tools	Jakeman, Chris	99-06	From the 'Net - Iterative Interpretation
tutorial	Charlton, Gordon	92-04	Two geese and a car
tutorial	Brown, Jack	92-06	An indefinite loop example
tutorial	Filbey, Gil	92-12	Escape codes and printing
tutorial	Filbey, Gil	93-02	A conjuring trick
tutorial	Hainsworth, Chris	93-02	Shallow end
tutorial	Filbey, Gil	93-04	Some old words revisited
tutorial	Filbey, Gil	93-10	Floating point
tutorial	Charlton, Gordon	93-12	Create .. does> ..
tutorial	Filbey, Gil	93-12	Postfix
tutorial	Filbey, Gil	94-02	Editorial & Tu
tutorial	Filbey, Gil	94-12	Floating point
tutorial	Filbey, Gil	95-08	Immediacy
tutorial	Filbey, Gil	95-10	Editorial
tutorial	Telfer, Graham	98-07	Wondrous Numbers
tutorial	Jakeman, Chris	98-11	jeForth Project
tutorial	Pochin, Dave	99-01	Forth for the New Year
tutorial	Pochin, Dave	99-01	Guide to Getting Started
tutorial	Pochin, Dave	99-04	Getting Stuck Into Win32Forth
tutorial	Pochin, Dave	99-08	Figuring it out with Win32Forth
tutorial	Jakeman, Chris	99-11	Clock Challenge
tutorial	Pochin, Dave	00-01	"See Win32Forth scroll the Window"
tutorial	Jakeman, Chris	00-01	Clock Challenge - 2nd installment
tutorial	Brien, Jack	00-04	All you need to know about STATE, IMMEDIATE and POSTPONE
tutorial	**Pochin, Dave**	**01-04**	**Six Easy Fonts**
tutorial	**Noble, Julian**	**01-09**	**A Call to Assembly 1/3**
tutorial	**Pochin, Dave**	**01-09**	**Win32Forth Fonts**
tutorial	**Noble, Julian**	**01-11**	**A Call to Assembly 2/3**
tutorial	**Pochin, Dave**	**02-01**	**The End of the Line**
tutorial	**Noble, Julian**	**02-01**	**A Call to Assembly 3/3**
vectoring	Charlton, Gordon	90-10	Resolving forward references (code)
vectoring	Jakeman, Chris	91-02	Deferred words (code)
vectoring	Preston, Philip	91-04	Forgettable vectors and smart compiling
vectoring	Bennett, Paul	92-10	Vectoring with DOER and MAKE
vectoring	Allwright, Ray	97-11	From the Net - Defer and Is

Chairman	**Jeremy Fowell**,	11 Hitches Lane, EDGEBASTON B15 2LS 0121 440 1809 jeremy.fowell@btinternet.com
Secretary	**Doug Neale,**	58 Woodland Way, MORDEN SM4 4DS 020 8542 2747 dneale@w58wmorden.demon.co.uk
Editor	**Chris Jakeman,**	50 Grimshaw Road, PETERBOROUGH PE1 4ET 01733 352373 cjakeman@bigfoot.com
Treasurer	**Neville Joseph,**	Marlowe House, Hale Road, WENDOVER HP22 6NE 01296 62 3167 naj@najoseph.demon.co.uk
Webmaster	**Jenny Brien,**	Windy Hill, Drumkeen, BALLINAMALLARD, Co. Fermanagh BT94 2HJ 02866 388 253 jennybrien@bmallard.swinternet.co.uk
Librarian	**Graeme Dunbar**	Electrical Engineering, The Robert Gordon University, Schoolhill, ABERDEEN AB10 1FR 01651 882207 g.r.a.dunbar@rgu.ac.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.