Forth-2012: Der neue Standard

M. Anton Ertl*
TU Wien

Bernd Paysan hat den neuen Forth-2012-Standard bereits in Heft 3-4/2014 vorgestellt, allerdings vor allem als Aufzählung von neuen Wörtern. In diesem Artikel beschreibe ich, wozu diese Neuerungen gut sind.

1 Was ist ein Standard?

Es gibt eine Reihe von Missverständnissen über die Rolle von Standards.

Ein Standard für eine Programmiersprache beschreibt eine Schnittstelle zwischen Programmen in dieser Programmiersprache und Implementationen (Compiler, Interpreter) dieser Sprache. Ein Standard-Programm läuft also auf einem Standard-System. Im Forth-Standard haben wir optionale Wörter, was die Sache etwas komplizierter macht: Ein Standard-Programm, das eine bestimmte Menge optionaler Wörter verwendet, läuft auf Standard-Systemen, die mindestens diese optionalen Wörter implementieren. Zum Glück stellen die meisten populären Forth-Systeme fast alle optionalen Wörter zur Verfügung (manche aber erst durch das explizite Nachladen gewisser Dateien).

2 Was kann der Standard und was nicht?

Unter den Standards gibt es welche, die spezifizieren wollen, was in Programmen und Systemen üblich ist (z.B. der Forth-Standard), aber auch solche, die den Stand der Technik weiterentwickeln wollen (z.B. WLAN-Standards). Der Forth-Standard zählt zur ersteren Art, und ist dementsprechend nicht der Platz, wo Innovation und neue Features zu finden sind. Wenn man ein neues Feature im Standard haben will, ist der Weg, es in einem System zu implementieren, dafür zu werben, dass Programmierer es verwenden, und andere Systeme es auch implementieren, und bei Erfolg dieser Bemühungen ist das Feature dann irgendwann üblich und kann standardisiert werden.

Bestimmte Features sind in verschiedenen Systemen verschieden implementiert. Mit Glück kommt im Standardisierungskommittee eine Einigung auf eine gemeinsame Variante zustande (wobei die Systeme zusätzlich ihre ursprünglichen Versionen implementieren), manchmal aber auch nicht. Und in jedem Fall muss sich auch noch jemand finden, der einen Vorschlag zur Standardisierung macht und den (relativ aufwändigen) Prozess dazu durchzieht.

Im Ergebnis enthält der Standard nur einen Teil der Features, die man sich als Anwendungsprogrammierer wünschen würde. Dann kann man systemspezifische Erweiterungen zu verwenden (eventuell in abgegrenzten Bereichen des Programms), aber bindet sich damit an ein System; in einigen Fällen kann man auch um die Abwesenheit eines Features herumprogrammieren; die Frage ist, ob sich das auszahlt.

Auf der anderen Seite gibt es immer wieder Kritik, dass der Standard für kleine Systeme und minimalistische Ansprüche zu groß ist. Programme für kleine Systeme weichen häufig vom Standard ab, sodass der Vorteil des Standards dabei wohl vor allem in einem gemeinsamen Grundvokabular besteht, weniger in der Portabilität von Programmen. Für minimalistische Ansprüche ist der Standard (und vermutlich jeder andere mögliche) wohl ungeeignet.

3 Was ist Forth-2012?

Forth-2012 ist der neue Standard. Er ist eine Weiterentwicklung des als ANS Forth bekannten Standards von 1994 (im folgenden Forth-94). Dabei haben wir großen Wet auf Kompatibilität gelegt, sodaß Forth-94-Programme auch Forth-2012-Programme sind.[1] Forth-2012 ist daher nicht revolutionär, sondern evolutionär.

Das Standard-Dokument in druckbarer Form ist auf `http://www.forth200x.org/documents/forth-2012.pdf` zu finden. Die Änderungen im Vergleich zu Forth-94 sind im Anhang C.7 aufgeführt, werden aber im Rest dieses Artikels vielleicht etwas verständlicher erklärt.

Oft ist aber die Online-Version `http://forth-standard.org/standard/words` praktischer als ein Ausdruck. Dort kann man auch über

*Correspondence Address: Institut für Computersprachen, Technische Universität Wien, Argentinierstraße 8, A-1040 Wien, Austria; `anton@mips.complang.tuwien.ac.at`

[1] mit Ausnahme der Programme, die Wörter und Features benutzen, die schon in Forth-94 als *obsolescent* (veraltend) gekennzeichnet waren und in Forth-2012 entfernt wurden.

Wörter und andere Teile des Standards diskutieren.

4 Entfernte Wörter und Features

Eine Reihe von Wörtern wurden in Forth-94 abgekündigt und die meisten von ihnen wurden in Forth-2012 dann auch entfernt:

- `Tib #tib` wurden entfernt, dafür gibt es seit Forth-94 `source`.
- `Convert` wurde entfernt, dafür gibt es seit Forth-94 `>number`.
- `expect span` wurden entfernt. Ersatz: `accept`
- `query` wurde entfernt. Der passende Ersatz scheint mir `refill` zu sein.[2]
- `Word` hinterlässt jetzt keinen Space mehr hinter dem Wort.
- `Forget` wurde zwar auch in Forth-94 abgekündigt, aber hat noch soviele Anhänger, dass es nicht aus Forth-2012 entfernt wurde.

Forth-94 hat Abfragen von wordsets mit `environment?` eingeführt (z.B. `sßtringënvironment?` zur Abfrage, ob das wordset vorhanden ist. Dieser Mechanismus war allerdings wenig populär und hätte für Forth-2012 erweitert werden müssen (damit man auch die Forth-2012-Varianten der Wordsets abfragen kann). Wir haben uns daher entschlossen, keine Forth-2012-Variante dieser Abfragen hinzuzufügen und haben die Forth-94-Varianten abgekündigt; sie werden im nächsten Standard entfernt. Stattdessen gibt es jetzt:

5 [defined]

Mit `[defined]` *word* bzw. `[undefined]` *word* kann man abfragen, ob ein Wort vorhanden ist. Wenn man nach einem Standard-Wort fragt, und es wird gefunden, gibt es zwar (ausser für `core`-Wörter) keine Garantie, dass es sich so verhält wie im Standard beschrieben, aber im Normalfall reicht das aus.

6 Jenseits von ASCII – XChars

Für Zeichensätze mit mehr als 256 Zeichen gibt es weitgehend ASCII-kompatible Kodierungen, die aus 8-bit-Elementen aufgebaut sind, wobei ein Zeichen dann aus einem oder mehreren 8-bit-Elementen bestehen kann. Am bekanntesten ist die UTF-8-Kodierung von Unicode, dem universellen Zeichensatz, aber auch z.B. die Kodierungen Big5 und GB2312 für chinesische Zeichen. Im folgenden konzentriere ich mich aber auf UTF-8.

Das schöne an den ASCII-kompatiblen Kodierungen ist, dass aufgrund der Kompatibilität der meiste Code ohne Änderungen damit funktioniert. Meistens wird ohnehin mit Strings hantiert, und wenn der String einfach nur eingelesen und dann wieder ausgegeben wird, oder bestenfalls zusammengehängt, ist es egal, wie die Zeichen in den Strings kodiert sind.

So funktionieren auch Forth-Systeme wie Gforth 0.6.2 (aus 2003), die ohne Rücksicht auf UTF-8 unter der Annahme von 8-bit-codierten Zeichen geschrieben wurden, recht gut mit UTF-8; nur der Kommandozeilen-Editor offenbart ein paar Schwächen, und die Anzeige von Fehlermeldungen ist auch suboptimal.

In vielen Fällen verwendet man daher bevorzugt Wörter, die Strings verarbeiten. Wenn man z.B. ein Unicode-Zeichen zweimal ausgeben will, kann man das mit

```
s" Δ" 2dup type type
```

machen, auch wenn man mit dem Xchar-Wordset auch

```
char Δ dup xemit xemit
```

schreiben könnte. Oder wenn man in einem String nach einem UTF-8-codierten Zeichen sucht, geht das z.B. so:

```
buf len s" Δ" search
```

Wobei `search` schon seit Forth-94 standardisiert ist, das nach einem einzelnen Byte suchende `scan` dagegen nicht.

Da alle Strings in einem Forth-System mit UTF-8 kodiert werden können, kann man auch in Wortnamen Unicode-Zeichen verwenden, z.B.:

```
variable Δ
1 Δ !
1 Δ +!
Δ @ .
```

Für die wenigen Fälle, in denen man auf einzelne Zeichen zugreifen muss, stellt Forth-2012 das Xchar-wordset zur Verfügung. Diese Wörter können für verschiedene Kodierungen verwendet werden, sowohl für die klassische 8-bit-Kodierung, also auch für UTF-8 oder die oben erwähnten älteren Kodierungen für Chinesische Zeichen, wenn das Forth-System diese Codierungen unterstützt. Derzeit unterstützt Gforth 8-bit-Codierungen (z.B. ASCII oder Latin-1) und UTF-8.

[2] In Forth-94 ist allerdings von `accept` und `evaluate` als Ersatz die Rede.

Was Forth-2012 allerdings nicht garantiert, ist, dass ein bestimmtes System, selbst wenn es das Xchars-Wordset hat, tatsächlich mit UTF-8 umgehen kann. Allerdings zeigen einige Tests mit SwiftForth und VFX Forth, dass diese Systeme UTF-8 verarbeiten können, auch wenn bei meine Tests Schwächen mit dem Kommandozeileneditor und den Fehlermeldungen offenbarten (wobei ich nicht ausschliessen kann, dass diese Systeme mit entsprechender Konfiguration eine noch bessere UTF-8-Unterstützung bieten).

Das größere Problem in der Praxis ist ausserhalb der Forth-Systeme: Wie bekommt man den Terminal-Emulator bzw. die Console, in der das Forth-System läuft, dazu, UTF-8-Ausgaben auch wie gewuenscht anzuzeigen, und wie gibt man solche Zeichen ein? Dafür gibt es zwar Lösungen, aber zumindest ich muss da noch öfters herumfrickeln.

7 Records

Man kann zwar auch in Forth-94 relativ leicht Records/Strukturen selbst definieren, aber da das ein häufig benötigtes Feature ist, haben wir es in Forth-2012 standardisiert. Ein Beispiel ist:

```
begin-structure flist
  field:  flist-next
  ffield: flist-val
end-structure

falign here flist allot constant flist1
0 flist1 flist-next !
1.23e flist1 flist-val f!
```

In diesem Beispiel haben wir eine Struktur `flist` (für einen Knoten einer verketteten Liste von Gleitkommazahlen) mit zwei Feldern: Einem Zellen-Feld `flist-next`, und einem Gleitkommazahlen-Feld `flist-val`.

Danach kommt die Definition eines Exemplars `field1` dieser Struktur. Um die richtige Ausrichtung (auf die Größe von Gleitkommazahlen) zu garantieren, wird hier nicht das folgende übliche Muster verwendet:

```
create flist1 flist allot
```

Schließlich werden die Felder von `field1` initialisiert.

Es gibt weitere Wörter zur Definition von Feldern mit verschiedenen Größen und Ausrichtungen: `cfield: sffield: dffield:`, und auch ein Basiswort `+field`; ausserdem kann man Strukturen auch ohne `begin-structure` und `end-structure` definieren. Man kann die obige Definition von `flist` daher auch ausschliesslich mit `+field` definieren:

```
0
  1 cells +field flist-next
  faligned 1 floats +field flist-val
constant flist
```

8 Zahlen mit expliziter Basis

Statt `base` zu ändern, kann man die Basis einer Zahl jetzt einfach durch einen Prefix angeben:

```
$ff \ hex
#99 \ Dezimal
%11 \ Binaer
'a' \ Zeichen
$-ff. \ hex double
```

Wie man an den Beispielen sieht, kann man neben Zahlen auch Zeichen so eingeben; die Forth-94-Methode mit `char` bzw. `[char]` funktioniert weiterhin.

9 Funktionstasten

Schon Forth-94 stellt `ekey` zur Verfügung, um Sondertasten und andere Nicht-Zeichen-Eingaben verarbeiten zu können. Allerdings kam bei `ekey` etwas nicht weiter spezifiziertes heraus, das man daher in einem portablen Programm nicht verwenden konnte. Forth-2012 fügt daher Wörter für Funktionstasten und Cursortasten hinzu, die man auch noch mit Modifikationstasten (Shift, Ctrl, Alt) kombinieren kann. Beispiel:

```
... ekey ekey>fkey if
 case
  k-up                                    of ... endof
  k-f1                                    of ... endof
  k-left k-shift-mask or k-ctrl-mask or of ... endof
  ...
 endcase
else
 ...
then
```

Das Ergebnis von `ekey` wird zunächst mit `ekey>fkey` als Sondertaste identifiziert (oder nicht, deswegen das `if`), und wenn es einer ist, wird aus dem `ekey`-code ein fkey-code erzeugt (diese Komplikation ist für Systeme mit komplizierteren `ekey`-Implementierungen gedacht). Der kann dann mit Tastencodes wie `k-up` (Cursor-up-Taste) oder `k-f1` verglichen werden. Man kann auch mit Tastenkombinationen wie Shift-Ctrl-Left vergleichen.

Allerdings gibt es keine Garantie, dass jedes System einen bestimmten Tastencode auch mit `ekey ekey>fkey` erzeugen kann, daher sollte man das Programm so schreiben, dass jede Funktion auch ohne die Sondertasten erreichbar ist.

10 Lokale Variablen

In Forth-94 gibt es eine Syntax für lokale Variablen (locals), die aber von vielen Programmierern nicht akzeptiert wurde, weil die Reihenfolge der locals in dieser Syntax im Vergleich zu Stack-Kommentaren verkehrt herum ist.

Stattdessen verwendeten viele die Syntax `{ a b c }`, wobei `c` mit dem top-of-stack initialisiert wird. Allerdings wird in SwiftForth `{...}` für Kommentare verwendet, sodass letztendlich `{: a b c :}` als Syntax standardisiert wurde. Ein kleines Beispiel, bei dem die Reihenfolge der Elemente eine Rolle spielt:

```
: swap ( a b -- b a ) {: a b :} b a ;
```

Da locals oft am Anfang einer Colon-Definition definiert werden, und dann oft alle Stack-Elemente in locals verfrachtet werden, wie im `swap`-Beispiel oben, wurde die Locals-Definition so erweitert, dass sie den Stack-Effekt-Kommentar ersetzen kann:

```
: swap {: a b -- b a :} b a ;
```

Dabei wird der Bereich von `--` bis (exklusive) `:}` ignoriert und ist nur Kommentar.

Einige Systeme erlauben nur eine Locals-Definition pro Colon-Definition. Daher will man auf diesen Systemen u.U. locals definieren, bevor man weiß, mit welchem Wert man sie initialisiert. Daher sieht der Standard auch die Möglichkeit vor, uninitialisierte locals zu definieren:

```
: foo {: a b | c d -- e f :} ... ;
```

Die Namen zwischen `|` und `--` bzw. `:}`, hier also `c d` sind uninitialisierte locals, die am Anfang einen beliebigen Wert haben können.

Das Wort `locals|` für die alte Locals-Syntax veraltet mit Forth-2012 und wird voraussichtlich im nächsten Standard nicht mehr vorhanden sein.

11 Laden von Dateien

Mit `require` *`file`* bzw. `required ( c-addr u -- )` kann man jetzt eine Forth-Quellcode-Datei einmalig laden. Das ist sinnvoll, wenn z.B. die Datei `lib.4th` unabhängig voneinander von den Bibliotheken `a.4th` und `b.4th` geladen werden, und ein Programm dann sowohl `a.4th` als auch `b.4th` lädt. Dann wird `lib.4th` nur einmal geladen.

Bei der Anwendung von `require` sollte das Interpretieren der geladene Datei allerdings einen neutralen Stack-Effekt haben, z.B. `( -- )` oder `( x y -- x y )`, da sonst der Stack-Effekt beim Nicht-Laden von dem beim Laden abweicht.

Weiters wurde das weit verbreitete Wort `include` endlich standardisiert.

12 Defer

Mit

```
defer foo ( n -- )
```

kann man einen Platzhalter `foo` definieren, und ihn dann mit einem beliebigen Wort füllen, z.B.:

```
' . is foo
```

Wenn man `foo` aufruft, wird bis auf weiteres `.` ausgeführt. Mit weiteren Aufrufen von `is` kann man das Verhalten von `foo` auch wieder ändern. Sinnvollerweise überlegt man sich für jeden Platzhalter einen Stack-Effekt und dokumentiert ihn, und füllt den Platzhalter nur mit Wörtern, die diesen Stack-Effekt haben.

Eine Anwendung davon ist indirekte Rekursion:

```
defer x ( n1 -- n2 )

: y ( n1 n2 -- n3 )
  ... x ... ;

: real-x ( n1 -- n2 )
  ... y ... ;

' real-x is x
```

Eine andere Anwendung sind Hooks, die man umdefinieren kann. So ist z.B. `type` in Gforth und VFX Forth ein deferred word, sodass man z.B. die Ausgabe umleiten oder abdrehen kann.

Man kann das eingefüllte xt mit `action-of` wieder herausbekommen, z.B. um `type` zeitweise umzuleiten, und danach den ursprünglichen Zustand wiederherzustellen:

```
action-of type ( old-xt )
' 2drop is type
... \ code mit abgeschalteter Ausgabe
( old-xt ) is type \ Wiederherstellung
```

Weiters gibt es `defer! ( xt1 xt2 -- )`, eine nicht-parsende Variante von `is`, und `defer@`, eine nicht-parsende Variante von `action-of`.

13 Reflexion über Wordlists

In Forth konnte man von Anfang an in das System Einblick nehmen und auf diverse System-Datenstrukturen zugreifen. Diese Fähigkeiten wurden aber nie in Form von Wörtern standardisiert, sondern bestenfalls die internen Datenstrukturen spezifiziert (z.B. threaded code in Forth-83).

Forth-94 hatte explizit das Ziel, die Implementierung nicht zu spezifizieren, wodurch die Möglichkeiten, die man davor z.B. durch das Wissen

über threaded code hatte, für Standard-Programme wegfielen; nur für wenige von diesen Möglichkeiten wurden Wörter eingeführt, z.B. `compile,` als Ersatz für das compilieren mit `,` in Forth-83.

In der Zwischenzeit haben die Entwickler anderer Programmiersprachen die Vorteile von Relexion erkannt[3], und stellen dafür standardisierte Schnittstellen zur Verfügung, und haben damit Standard-Forth in diesem Bereich überholt.

In Forth-2012 bekamen wir wenigstens ein bisschen Reflexion standardisiert: Wir können jetzt die einzelnen Wörter einer Wordlists herausbekommen, und von einem Wort wichtige Eigenschaften.

Dazu haben wir einen neuen zellengroßen abstrakten Datentypen, der ein benanntes Wort identifiziert, das *name token (nt)*. In einem konkreten System kann z.B. die NFA oder die CFA als nt dienen.

Mit `traverse-wordlist` kann man über alle Wörter einer Wordlist iterieren, und bekommt dabei die nts der einzelnen Wörter heraus. Im folgenden Beispiel werden die Wörter nur gezählt und die nts nicht weiter verwendet:

```
: count-helper ( n1 nt -- n2 f ) drop 1+ true ;
0 ' count-helper forth-wordlist traverse-wordlist .
```

Man muss `traverse-wordlist` das xt eines Wortes übergeben, das für jedes Wort in der Wordlist einmal aufgerufen wird, hier `count-helper`. Davor wird noch mit `0` der anfängliche Zählerstand auf den Stack gelegt. `Count-helper` wirft das nt weg, erhöht den Zählerstand `n1` mit `1+`, und legt noch `true` auf den Stack (bei `false` würde `traverse-wordlist` nicht weiteriterieren). Am Ende bleibt der endgültige Zähler auf dem Stack und wird mit `.` ausgegeben.

Mit dem nt kann man den Namen des Wortes (mit `name>string ( nt -- c-addr u )`), die interpretation semantics `name>interpret ( nt -- xt )` und die compilation semantics `name>compile ( nt -- x xt )` herausfinden (wobei `name>compile execute` die compilation semantics des Wortes ausführt). Ein Beispiel, wie diese Wörter mit `traverse-wordlist` zusammenspielen:

```
: words-helper ( nt -- f ) name>string cr type true ;
' words-helper get-current traverse-wordlist
```

Dieses kleine Programm ist ähnlich `words`, gibt aber die `current`-Wordliste aus.

14 Parse-name

`Parse-name ( -- c-addr u )` wird üblicherweise statt dem idiom `bl word count` verwendet. Die von `Parse-name` zurückgegebene String-Beschreibung beschreibt den String im Input-Buffer, die Beschreibung kann also bis zum nächsten `refill` verwendet werden:

```
parse-name foo parse-name bar type type
```

gibt `barfoo` aus.

15 IORs werfen

Viele Wörter (z.B. `open-file`) geben als Fehlerindikator einen IOR zurück. Es ist üblich, diese `iors` als `throw`-code zu benutzen, aber Forth-94 garantiert keinen besonderen Zusammenhang zwischen IORs und `throw`, abgesehen davon, dass ior=0 keinen Fehler bedeutet und `0 throw` nichts bewirkt. In Forth-2012 werden die IORs jetzt explizit als gültige `throw`-codes genannt, und Forth-Systems sollten jetzt sinnvolle Fehlermeldungen ausgeben, wenn sie solche Fehler-`throw`s fangen; z.B. gibt Gforth bei `open-file throw` die Betriebssystemmeldung für den Fehler aus, z.B. “No such file or directory´´.

16 Steuerzeichen in Strings

Mit `s"` kann man keine Strings mit Steuerzeichen oder ¨ definieren. Deshalb gibt es jetzt das Wort `s\"`, das das erlaubt. Steuerzeichen werden ähnlich wie in C spezifiziert:

```
s\" abc \"def\nghi\tjkl"
```

17 Synonyme

Synonym erlaubt es, einen neuen Namen (bzw. auch den gleichen Namen in einer anderen Wordlist) für ein existierendes Wort zu definieren, z.B.:

```
synonym endif then
```

Warum nicht `alias`? Weil das besonders fuer immediate-Wörter auf verschiedenen Systemen verschiedenes macht.

18 2value fvalue

Es gibt jetzt eine double- und eine FP-Variante von `value`; in allen Fällen wird mit `to` der Wert verändert:

```
5. 2value d 6. to d
5e fvalue f 6e to f
```

[3] https://de.wikipedia.org/wiki/Reflexion_\%28Programmierung\%29

19 Kleinkram

Mit `buffer:` kann man einen uninitialisierten Puffer definieren:

```
500 cells buffer: b
b 500 cells erase
```

Mit `n>r` und `nr>` kann man n Zellen auf den Return-Stack legen bzw. von dort holen; diese Wörter können im Zusammenhang mit Wörtern wie `get-order` sinnvoll sein, die n Zellen und den Zähler n auf den Datenstack legen:

```
get-order n>r
nr> set-order
```

20 Textersetzung

Es gibt jetzt eine Makro-Textersetzung mit `replaces` und `substitute`, gedacht besonders für Internationalisierung:

```
: datum1 s" 2015-04-11" ; datum1 s" date" replaces
: zeit1 s" 21:00" ;       zeit1 s" time" replaces
s" Um %time% am %date% ..." pad 100 substitute throw type
\ ergibt "Um 21:00 am 2015-04-11 ..."
s" On %date% at %time% ..." pad 100 substitute throw type
\ ergibt "On 2015-04-11 at 21:00 ..."
```

Damit man bei einem übergebenen String, der zufällig eine Makro-Sequenz enthält, die man nicht ersetzt haben möchte, auch keine Ersetzung bekommt, kann man den Text zuerst durch `unescape` entschärfen lassen:

```
500 buffer: buf
s" zufälliges %date% im Text" buf unescape ( c-addr u )
pad 100 substitute throw type
\ ergibt "zufälliges %date% im Text"
```

21 Gleitkomma-Arithmetik

In Forth-94 konnten Gleitkommazahlen (floating point numbers, FP) auf dem Datenstack oder auf einem eigenen Gleitkomma-Stack abgelegt werden, und Standard-Programme mussten so geschrieben werden, dass sie für beides funktionieren, was so umständlich ist, dass es kaum jemand gemacht hat, und die allermeisten Programmierer für einen separaten FP-Stack geschrieben haben. In Forth-2012 ist jetzt der separate Gleitkommastack standardisiert.

Es gibt jetzt `s>f f>s` zum Konvertieren zwischen einfachgenauen ganzen Zahlen und Gleitkommazahlen. Mit `ftrunc` kann man jetzt zur nächsten ganzen Zahl in Richtung 0 runden (das Ergebnis ist aber immer noch eine Gleitkomma-Zahl). Weiters wurden die Wörter `fasinh fatan2` genauer spezifiziert.

22 Ausblick

Auch wenn wir mit Forth-2012 einen Meilenstein geschafft haben, geht die Arbeit an Forth-200x weiter.

Derzeit sind `locals|` und `[compile]` als veraltend markiert und werden voraussichtlich mit der nächsten Version aus dem Standard verschwinden. Stattdessen sollte man `{:` bzw. `postpone` benutzen.

Bei der letzten Standardisierungssitzung 2015 haben wir beschlossen, dass Forth-Systeme die 2er-Komplementarithmetik negativer Zahlen und für den Überlauf die übliche Modulo-Arithmetik, sodass jetzt z.B. übliche Hash-Funktionen effizient in Standard-Forth ausgedrückt werden können. In eine ähnliche Richtung geht der Vorschlag, `1 chars=1` zu standardisieren, aber da konnte sich das Kommitte noch nicht zu einem Konsens dafür durchringen.

Es liegen noch weitere Vorschläge vor, die allerdings kontroverser sind und wohl einige Jahre benötigen werden, bis es einen Konsens für sie gibt: *Quotations* erlauben es, Hilfsdefinitionen wie sie für `catch` oder `traverse-wordlist` benötigt werden, direkt und namenlos innerhalb einer Definition zu schreiben. Beispiel:

```
: count-words ( wid -- n )
  0 [: drop 1+ true ;] rot traverse-wordlist ;
```

Recognizer erweitern den Text-Interpreter um die Erkennung beliebiger Wörter. Beispiele für Recognizer sind die Zahlenerkennung und -Konversion in aktuellen Forth-Systemen; diese sind aber derzeit nicht portabel erweiterbar, sondern nur über systemspezifische Hooks. Standardisierte Recognizer würden das ändern.

Weitere geplante Themen sind Wörter für 16-bit-Speicherzugriffe u.ä., multi-tasking und multithreading, und das C-Interface.

23 Macht mit!

Zuviele Köche verderben den Brei und oft auch die Software, aber bei Standards ist das anders: Der Standard soll ja das widerspiegeln, was in der Community üblich ist, und um das herauszufinden, brauchen wir Input von Euch, der Forth-Community. Auch die Probleme, die Ihr habt, und die der aktuelle Standard keine Lösungen bietet, aber Eurer Meinung nach bieten sollte, solltet Ihr an uns herantragen (auch wenn dann in vielen Fällen eine Erklärung zurückkommen wird, wie das jetzt schon geht, oder warum wir das jetzt nicht angehen).

Wie könnt Ihr beitragen? Ihr könnt in der Newsgroup `comp.lang.forth` oder in der Mailingliste[4]

[4] `https://groups.yahoo.com/neo/groups/forth200x/info`

über Vorschläge (RfDs) mitdiskutieren und Euch bei der Umfrage über die fertigen Vorschläge (CfVs) beteiligen.

Wenn wir einen Release Candidate des Standard-Dokuments haben (bisher einmal in 11 Jahren), könnt Ihr Euch an der öffentlichen Begutachtung beteiligen. Dabei sind das letzte mal Fehler gefunden und beseitigt worden; für inhaltliche Änderungen ist es aber sinnvoller, sich schon in der RfD/CfV-Phase zu beteiligen.

Wenn Ihr Euch stärker beteiligen wollt, könnt Ihr selbst Vorschläge (RfDs) machen. Seid darauf gefasst, dass dabei immer von irgendjemandem Gegenwind kommt, egal wie üblich das Vorgeschlagene ist, und lasst Euch davon nicht entmutigen; wenn der Vorschlag umgekehrt gar keine Unterstützung erfährt, dürft Ihr Euch schon entmutigen lassen. In jedem Fall ist mit dem Formulieren des Vorschlags und seiner Revisionen, und der Behandlung der Kommentare ein gewisser Zeitaufwand verbunden.

Wenn Ihr Euch noch stärker beteiligen wollt, kommt zum Standardisierungstreffen (derzeit zwei Tage vor der EuroForth). Ihr erlebt dann intensive Diskussionen über Vorschläge, Verfahrensfragen, Kleinigkeiten, und künftige Entwicklungen, und könnt auch mitreden. Bei Eurem zweiten mal hintereinander dürft Ihr auch an Abstimmungen teilnehmen, mit Zustimmung des Kommittees auch beim ersten mal.