Forth 200*x* Standards Committee

Sidney Sussex College, Cambridge University, England

14–15 September 2006

Agenda

1. Apologies
2. Absences
3. Minutes of last meeting
4. Matters arising:
   (a) Publication of Minutes
   (b) Review of RfD/CfV process
   (c) Apportionment of a Secretary
   (d) Development Reports:
      i. Federico de Ceballos
      ii. M. Anton Ertl
      iii. Peter Knaggs
      iv. Stephen Pelc
      v. Bill Stoddart
   (e) Review of the Forth2006-2 basis document.
5. Frozen proposals for consideration
   (a) EKEY return values for some keyboard events (3/e)
   (b) Separate FP Stack (2/e)
   (c) One-time file loading (2/e)
6. Unfinished proposals for discussion
   (d) Escaped strings (`S\"` and `.\"`) (2/e)
   (e) Front Matter (2/e)
   (f) Floating Values
   (g) Enhanced local variable syntax (2/e)
   (h) Represent
   (i) Structures (3/e)
   (j) Synonym (2/e)
   (k) `THROW` codes and IORs
   (l) XCHAR wordset (for UTF-8 and alike)
7. Copyright
8. Date of Next Meeting
9. Any Other Business

Forth 200x Standards Committee

Hotel Santemar in Santander, Spain

21–23 October 2005

Draft Minutes

Participants

Willem Botha	CCS, South Africa	
Federico de Ceballos	University of Cantabria, Spain	
M. Anton Ertl	Technische Universität Wien, Austria	(Chair)
N. J. Nelson	Micross, England	(Observer)
Dr. Peter Knaggs	University of Bournemouth, England	
Stephen Pelc	MPE, England	
Dr. Bill Stoddart	University of Teesside, England	(Observer)

Action on proposals

Proposal	Comments	Status	Action
X:deferred	`DEFER`, `DEFER@`, `DEFER!`, `IS`, `ACTION-OF`	Passed 5/0/0	
X:defined	`[DEFINED]` and `[UNDEFINED]`	Passed 5/0/0	
X:parse-name	`PARSE-NAME`	Passed 5/0/0	
X:extension-query	Modification to `ENVIRONMENT?`, revised to remove auto-loading of requested extensions	Passed 5/0/0	

Other Business

1. Integration into the standards document

   Should new words be put into the established wordsets, or into new ones?

   The eventual goal is to usually integrate the new words into existing wordsets with related functionality; in some cases it may be more appropriate to create a new word set. However, as an intermediate step the new proposals will at first be kept separate, to make it easier for readers of the document to see what is changing.

2. How are the extension-query names reflected in the standard (if at all)?

   The glossary header for new words includes the extension-query string for the extension that proposed it. In addition, there will be a chapter or normative appendix that lists all the extensions, their extension-query strings and the components (word definitions etc.) that it consists of.

3. Should the tests of a proposal or the reference implementation become normative?

   No. This could lead to conflicting normative sections; also, making the reference implementation normative would lead to over specification.

4. Review of the RfD/CfV process

   How well is the RfD/CfV process working at generating high- quality proposals for standardisation and getting information about their popularity? What could be improved? Or should we do something completely different?

   Many of the participants were not very familiar with how well the process worked in practice, and had no suggestions for improvements.

   The (normative part of the) proposals required adoption before integrating them into the document, but there was a widespread feeling among the participants that proposals in the form of unambiguous instructions to the document editor (which is the form that would be voted on by the standards committee) would be harder to understand for the CfV audience.

The resulting idea was to have two forms of the proposal, with the same content: First the not-so-formal form used in the CfV, and later a form for integrating it into the document.

5. Official standards body

   Should we run the standard through a standards body like ANSI, ISO, IEEE, etc.? If so, which one?

   Some participants consider the blessing of the future standards document by an official standards body very important, and we agreed to work towards this goal by writing the document in the appropriate style, and by keeping documentation about all our steps. However, the general idea was to first develop the document without involving a standards body, and deal with them at the end.

   Various candidate standards bodies were discussed; none was decided on, but it might be that going through ANSI again might be the easiest route.

6. Chairman and Editor M. Anton Ertl was appointed as the chair of the committee unopposed.

   Dr. Peter Knaggs was appointed editor unopposed.

7. Which standard documents should we start from?

   Peter Knaggs has a version of dpANS99a in LaTeX form, convertible into a fully hyperlinked PDF file. However, it is yet unclear how this document differs from the ANS/ISO Forth documents and dpANS6.

8. Integration of CfV into the Standard

   How do we get from the CfV proposal to the form for integration into the document? One opinion was that the original proponent should do it.

9. Proposal Champions

   There was the opinion that a proposal (de-facto) needs a champion in the committee to get approval by the standards committee. So, if the proponent finds a champion in the committee, they could produce the for-the-document version of the proposal together.

10. Improvements for future standard meetings

    - The participants should familiarise themselves with the proposals beforehand.
    - Paper printouts of the proposals should be available at the meeting.
    - Proposals should be available in the form needed for integration into the standards document in addition to the CfV form.
    - There should be a champion for the proposal in the meeting.

Action Items

1. Federico de Ceballos was asked to look into providing a proposal to cover the following topics:

   - number prefixes
   - `0` for `NIL`
   - `!` and `@` for 16-bit and 32-bit signed and unsigned integers, bytes, octets

2. M. Anton Ertl was asked to look into the implications of the following and develop proposals for them:

   - separate FP stack
   - required

- directory stuff in general
- directory handling for included and required
- key names for`EKEY` results

3. Stephen Pelc was asked to look into the implication of the following topics, and develop proposals for:
   - { `(locals)`, fp locals, buffer locals
   - `S\" .\"`
   - iors can be `THROW`n
   - `SYNONYM`
   - structures
4. Bill Stoddart was asked to investigate the implications of allowing the parser to ignore white space (TAB, CR, LF, and FF) in source code, and develop a proposal.
5. Peter Knaggs to contact John Hayes to obtain permission to include the validation suite into the document and to produce a basis document based on the ANS Forth standard with the four accepted proposals and validation suite included.

Next Meeting

It was agreed that the next meeting would take place immediately before the next euroForth conference to be held in Cambridge, England.

A EKEY return values for some keyboard events

A.1 Author

Anton Ertl

A.2 Poll standings

A.2.1 Systems

1. Conforms to ANS Forth.
   (a) iForth
   (b) Gforth
   (c) VFX Forth for Windows/DOS/Linux
   (d) MPE Forth cross compilers
2. Already implements the proposal in full since release:

   None
3. Implements the proposal in full in a development version:

   None
4. Will implement the proposal in full in release.
   (a) Gforth 0.7
5. Will implement the proposal in full in some future release.
   (a) iForth
   (b) VFX Forth for Windows/DOS/Linux
6. There are no plans to implement the proposal in full in [ ].
   (a) 4th V3.5a (no EKEY support)
7. Will never implement the proposal in full:
   (a) MPE Forth cross compilers

A.2.2 Programmers

1. I have used (parts of) this proposal in my programs:
   (a) Robert Epprecht
2. I would use (parts of) this proposal in my programs if the systems I am interested in implemented it:
   (a) David N. Williams
   (b) Graham Smith
3. I would use (parts of) this proposal in my programs if this proposal was in the Forth standard:
   (a) Charles Melice

(b) David N. Williams

(c) Graham Smith

4. I would not use (parts of) this proposal in my programs.

   (a) Hans Bezemer (does not use EKEY)

A.3 Change History

2006-06-23
Added `EKEY>FKEY` after private discussions with Neal Bridges. Added some remarks and comments, and adapted title to satisfy commenters.

2006-05-28
Added `K-INSERT`, `K-DELETE`, `K-F*` and `K-*-MASK`. Added Sections "Programming Advice" and "Shift Keys". Reported comments by various people.

A.4 Problem

How do I write a portable Forth program that reacts to key presses of cursor keys (and possibly other keys not represented by ASCII), e.g., an editor?

A.5 Proposal

`EKEY>FKEY` (u_1 -- u_2 f) "e-key to f-key" facility ext

If the keyboard event u1 corresponds to a keypress in the implementation-defined special key set, return that key's id u_2 and $true$. Otherwise return u_1 and $false$.

The following words produce the same values that `EKEY EKEY>FKEY` may produce when the user presses the corresponding key. Note that, even if this extension is present, the keyboard may lack some of the keys, or the system the capability to report them, so programs should be written such that they also work (although less conveniently or with less functionality) if these key numbers cannot be produced.

Use:

```
... 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
```

Note: The `K-...-MASK` words produce a mask, that you can `OR` with the key values produced by one of the `K-...` words to produce a value that `EKEY` may produce when the user presses the corresponding key combination.

`K-LEFT` (-- u) facility ext

The "cursor left" key

`K-RIGHT` (-- *u*) facility ext

The “cursor right” key

`K-UP` (-- *u*) facility ext

The “cursor up” key

`K-DOWN` (-- *u*) facility ext

The “cursor down” key

`K-HOME` (-- *u*) facility ext

The “home” or “Pos1” key

`K-END` (-- *u*) facility ext

The “End” key

`K-PRIOR` (-- *u*) facility ext

The “PgUp” or “Prior” key

`K-NEXT` (-- *u*) facility ext

The “PgDn” or “Next” key

`K-INSERT` (-- *u*) facility ext

The “Insert” key

`K-DELETE` (-- *u*) facility ext

The “Delete” key

`K-F1` (-- *u*) facility ext

The “F1” key

`K-F2` (-- *u*) facility ext

The “F2” key

`K-F3` (-- *u*) facility ext

The “F3” key

`K-F4` (-- *u*) facility ext

The “F4” key

`K-F5` (-- *u*) facility ext

The “F5” key

`K-F6` (-- *u*) facility ext

The “F6” key

`K-F7` (-- *u*) facility ext

The “F7” key

`K-F8` (-- *u*) facility ext

The “F8” key

`K-F9` (-- *u*) facility ext

The “F9” key

`K-F10` (-- *u*) facility ext

The “F10” key

`K-F11` (-- *u*) facility ext

The “F11” key

`K-F12` (-- *u*) facility ext

The “F12” key

`K-SHIFT-MASK` (-- *u*) facility ext

Mask for the shift key

`K-CTRL-MASK` (-- *u*) facility ext

Mask for the ctrl key

`K-ALT-MASK` (-- *u*) facility ext

Mask for the alt key

A.6 Remarks

A.6.1 `EKEY>FKEY`

Introducing that word leaves more freedom to implementations about the representation of `EKEY` results. In particular, based on private discussions, it would be easier for Neal Bridges to implement the proposal in Quartus Forth. Systems that do not need this additional freedom could use the same representation for the `EKEY` and `EKEY>FKEY` results; a simple implementation of `EKEY>FKEY` on such a system would be:

```
:  EKEY>FKEY ( u1 -- u2 flag )
  dup ekey>char nip 0= ;
```

A.6.2 `EKEY` return values

There was a discussion about what the standard says about `EKEY` return values. In particular, I interpret the standard's non-specification of a lifetime for the return value as meaning that it has unlimited lifetime and thus an implementation where the return value is an address of an overwritable buffer is non-standard; whereas Elizabeth Rather believes that the implementation-defined nature of the result would make such an `EKEY` implementation standard. However, no such implementation of `EKEY` is known, and a clarification of that aspect is desirable. However, with the introduction of `EKEY>FKEY` the proposal could even be implemented on a system with such an `EKEY` implementation, so the place for the clarification is probably not in this proposal (unless we take `EKEY>FKEY` out again).

A.6.3 Programming advice

Note that, even if a Forth system supports these words, many environments do not have or do not report all these keys and key combinations, so it is a good idea to write your program such that it is still useful even if these keys and key combinations cannot be pressed or are not recognized.

A.6.4 Shift keys

Note that, as defined, the shift key masks defined above are only useful for recognizing shifted cursor and function keys, because these are the only keys that `EKEY` return values are defined for. E.g., we cannot program Forth to recognize ALT-T, because no `EKEY` return value for T has been defined.

A.6.5 Other Keys

Gforth and PFE have words `K1`...`K10` for the function/keypad keys; they also contain `S-K1`...`S-K10` for shifted function/keypad keys, but they don't work as widely. Moreover, Gforth (but not PFE) also has `K-INSERT`, `K-DELETE`, `K11`, `K12`, `S-K11` and `S-K12`. Should any of these words be added to this proposal? After the first RfD, several people were in favour of adding such words and nobody spoke out against, so I added the words after K-NEXT. If you don't like such words in general, or would like them, but differently, please speak up now.

A.7 Experience

These words have been implemented for several years in PFE and Gforth. Several editors have been published in ANS Forth that would have profited from these words. Also, the original version of the MiniSpreadsheet[1] hardcoded the values for the keys for one platform; I then fixed it to use the cursor-key words proposed above.

A.8 Implementation and Tests

The implementation is closely tied to the implementation of `EKEY`, and therefore unportable, so I don't provide a reference implementation. However, you can look at the Gforth implementation[2] of `EKEY` and many of these words (based on ANSI terminal escape sequences).

Tests[3].

[1] http://wiki.forthfreak.net/index.cgi?MiniSpreadsheet

[2] http://b2.complang.tuwien.ac.at/cgi-bin/cvsweb/~checkout~/gforth/ekey.fs?rev=HEAD;content-type=text%2Fplain;cvsroot=gforth

[3] http://www.complang.tuwien.ac.at/forth/ansforth/tests/ekeys.fs

A.9 Comments

- Alex McDonald writes:

  Win32Forth provides;

  k_home k_up k_pgup k_left k_right k_end k_down k_pgdn k_insert k_delete k_scroll

  plus k_1 through k_12 for function keys.

  +k_control +k_alt +k_shift provide a mask when these keys are depressed with any of the above.

- Marcel Hendrix writes that iForth implements this functionality, but with different key names.

- Mitch Bradley reports that the current proposal cannot be implemented in a useful way on the current Quartus Forth even though the platform has some arrow buttons. Quartus Forth implements an `EKEY` that just returns an event type (and requires the use of another word to get more information).

- Charles Melice and Alex McDonald would prefer `K-F1` etc. for function keys, and masks for shifted keys etc.

- Charles Melice would also like to see some events defined for mouse events. There was some discussion on how that should be done; however, it will not be done in this proposal.

- Sam Falvo also has some interesting suggestions in this area that might be interesting for a future RfD.

- Charles Melice also proposed having a word `EKEY>FKEY` and standardizing the numeric `FKEY` values to make it easier to save, exchange, and replay these keys between systems.

- Robert Epprecht uses these words and would like to have them available in different Forth implementations. Also, he would like to see the words for the function keys.

B Separate FP Stack

B.1 Author

Anton Ertl

B.2 Poll standings

B.2.1 Systems

1. Conforms to ANS Forth.
   (a) Mops/PowerMops (Mike Hore)
   (b) ForthCAD (Charles Melice)
   (c) iForth (Marcel Hendrix)
   (d) bigForth (Bernd Paysan)
   (e) Gforth (Anton Ertl)
   (f) VFX Forth for Windows/DOS/Linux (Stephen Pelc)
   (g) MPE Forth cross compilers
   (h) MacForth/Power MacForth/Carbon MacForth (Ward McFarland)
2. Already implements the proposal in full since release:
   (a) PowerMops 3
   (b) iForth 1
   (c) bigForth 1.0
   (d) Gforth 0.1
   (e) VFX Forth for Windows/DOS/Linux 3.0
   (f) MacForth/Power MacForth/Carbon MacForth since at least 10 years
3. Implements the proposal in full in a development version:

   None
4. Will implement the proposal in full in release.
   (a) MPE Forth cross compilers 7.0
5. Will implement the proposal in full in some future release.
   (a) ForthCAD
6. There are no plans to implement the proposal in full in.
   (a) 4th V3.5a (no FLOAT wordset at all)
7. Will never implement the proposal in full.

   None

B.2.2 Programmers

1. I have used (parts of) this proposal in my programs:
   (a) David N. Williams
   (b) Charles Melice
   (c) Julian V. Noble
   (d) Bernd Paysan
   (e) Anton Ertl
   (f) Stephen Pelc
2. I would use (parts of) this proposal in my programs if the systems I am interested in implemented it:
   (a) David N. Williams
   (b) Charles Melice
   (c) Stephen Pelc
   (d) Tessa Celine Bonting
3. I would use (parts of) this proposal in my programs if this proposal was in the Forth standard:
   (a) David N. Williams
   (b) Stephen Pelc
4. I would not use (parts of) this proposal in my programs.
   (a) Hans Bezemer (does not use FP)

B.2.3 Informal results

- Charles Melice has developed a program that allows running his unified-stack 1-cell-per-fp legacy code on a separate-stack system.
- Julian V. Noble would never use a system with a unified stack.
- BigForth: The 68k software FP implementation of bigFORTH containes an optional mode where the stack is unified. The x86 version uses the native stack of the x87 FPU, and didn't offer this unified stack option.

B.3 Change History

2006-07-04
No normative changes. Updated Section Experience with ForthCAD material. Added Tests.

2006-06-27
No normative changes. Updated Section Experience with material from feedback on the 1st RfD. Added Section Remarks.

B.4 Problem

Writing floating-point code such that it can run on a unified stack is such a pain that most programmers don't do it.

B.5 Proposal

The floating-point stack is separate from the data stack.

Typical Use

```
\ from <126r7o1srr8aof0@news.supernews.com>
: square ( f - f') fdup f* ;
: pow ( n) ( f - f')
  ?dup 0= if fdrop 1e0 exit then
  dup 1 = if drop exit then
  2 /mod fdup square recurse fswap recurse f* ;
```

So what is the actual proposal? (Ed)

B.6 Remarks

Unified stacks with a specified size for FP values

While many agreed with the "pain" assessment above, several comments pointed out that the unified stack is not as big a pain if the programmer programs for a specific on-stack size of FP values (e.g., double-cells), if that size is not larger than double-cells.

However, such a dependency on a specific size of FP values reduces the portability even to other systems with a unified stack (and maybe even to other platforms for the same system) as well as eliminating the portability to systems with a separate FP stack.

B.6.1 Temporary storage

C. G. Montgomery proposed adding words for transferring values between the FP stack and the data stack for temporary storage. The return stack seems to be a better place for temporary storage, though. Marcel Hendrix would prefer to see FP locals instead. Since there is a promise for an upcoming proposal for FP locals, I will not propose any such words in this proposal.

B.7 Experience

Most ANS Forth systems that I know implement a separate floating-point stack. MPE's embedded systems use a unified stack (their desktop system use separated stacks). Kforth implements a subset of ANS Forth subset and has a unified FP stack. PFE implements ANS Forth and has an optional module for a unified stack (it also supports a separate FP stack). ForthCAD, COLDFORTH and uCFORTH use a unified stack (ANS Forth status unknown). ForthCAD originally used this to make the implementation of the C call interface simpler on the IA-32 architecture, and sticks with it because there is too much code that relies on this.

All ANS Forth programs dealing with FP numbers that I know except code written specifically for kforth just assume a separate floating-point stack, including the Forth Scientific Library.

B.8 Implementation and Tests

- Incomplete Implementation[4] the main missing piece is the (system-specific) text-interpreter floating-point input.

[4] http://www.forth200x.org/reference-implementations/fp-stack.fs

- Tests[5]

[5] http://www.complang.tuwien.ac.at/forth/ansforth/tests/fp-stack.fs

C One-time file loading

C.1 Author

Anton Ertl

C.2 Poll standings

C.2.1 Systems

1. Conforms to ANS Forth.
   (a) Gforth
   (b) VFX Forth
   (c) Forth6 cross compilers (Stephen Pelc)
2. Already implements the proposal in full since release:
   (a) Gforth, all releases
3. Implements the proposal in full in a development version:

   None.
4. Will implement the proposal in full in release.
   (a) VFX Forth 4
5. Will implement the proposal in full in some future release.
   (a) Forth6 cross compilers
6. There are no plans to implement the proposal in full.
   (a) 4th V3.5a (Hans Bezemer)
   (b) MacForth (Ward McFarland)
7. Will never implement the proposal in full:

   None.

C.2.2 Programmers

1. I have used (parts of) this proposal in my programs:
   (a) Hans Bezemer
   (b) Anton Ertl
   (c) Robert Epprecht
   (d) David N. Williams
2. I would use (parts of) this proposal in my programs if the systems I am interested in implemented it:
   (a) Stephen Pelc
3. I would use (parts of) this proposal in my programs if this proposal was in the Forth standard:

(a) Stephen Pelc

(b) Charles Mélice

4. I would not use (parts of) this proposal in my programs.

   None.

C.3 Change History

2006-02-12
Added discussion on whether `REQUIRED` should re-include files that were `INCLUDED`.

2006-01-21

- Normative: Added memory allocation restriction to `INCLUDED` (and a related section in the informative parts).
- Normative: After forgetting the contents of a file with `FORGET` or a marker, `REQUIRED` will include the file again (plus an informative section on that; the reference implementation does not implement that yet).
- Normative: Put the new words in the FILE EXT word set.
- Normative: replaced "perform word" with "perform the function of word".
- Recorded answers to the question about the name of `REQUIRE` / `NEEDS` / `REQUIRES`, and corrected information about `REQUIRES` in MPE systems (it does not perform what is proposed).
- Added a section on `INCLUDE` interacting with `BASE`.
- Added information about features to increase the efficiency of the C approach.

2001-08-14
Changed stack effect from (i*x c-addr u -- j*x) to (i*x c-addr u -- i*x), following a suggestion from Guido Draheim, and related changes in the text.

C.4 Problem

A library is needed by several parts of the source code (e.g., in other libraries), but it should be loaded only once.

C.5 Proposal

Add the following restriction to the definition of `INCLUDED`:

> `INCLUDED` may allocate memory in data space before it starts interpreting the file.

In the definition of `INCLUDED`, change "Text interpretation begins at the file position where the next file read would occur" into "Text interpretation begins at the start of the file".

Define the following words:

`REQUIRED` ($i*x$ $c-addr$ u -- $i*x$) FILE-EXT

> If the file specified by $c-addr$ u has been `INCLUDED` or `REQUIRED` already, but not between the definition and execution of a marker (or equivalent usage of `FORGET`), discard $c-addr$ u; otherwise, perform the function of `INCLUDED`.

An ambiguous condition exists if a file is REQUIRED while it is being REQUIRED or INCLUDED.

An ambiguous condition exists, if a marker is defined outside and executed inside a file or vice versa, and the file is REQUIRED again.

An ambiguous condition exists if the same file is REQUIRED twice using different names (e.g., through symbolic links), or different files with the same name are REQUIRED (by doing some renaming between the invocations of REQUIRED).

An ambiguous condition exists if the stack effect of including the file is not ($i * x$ -- $i * x$).

REQUIRE ($i * x$ "*name*" -- $i * x$) FILE-EXT

Skip leading white space and parse name delimited by a white space character. Push the address and length of the name on the stack and perform the function of REQUIRED.

INCLUDE ($i * x$ "*name*" -- $j * x$) FILE-EXT

Skip leading white space and parse name delimited by a white space character. Push the address and length of the name on the stack and perform the function of INCLUDED.

C.6 Typical Use

```
... s" filename" required ...
require filename
include filename
```

C.7 Remarks

C.7.1 Changes to INCLUDED

The added restriction codifies an existing environmental restriction in widely-used systems (at least Win32Forth and MPE's systems) that has not led to problems in programs. In the context of REQUIRED this restriction allows a simpler implementation. In effect this restriction means that colon definitions or contiguous data regions cannot extend across calls to INCLUDED (one can use INCLUDE-FILE instead).

As an alternative to adding this restriction, we could drop the requirement that REQUIRED does not re-include files that were INCLUDED. People probably do not use REQUIRED and INCLUDED on the same file anyway. The disadvantages of this alternative are: the result is probably confusing in cases where people do use REQUIRED and INCLUDED on the same file; and AFAIK all systems that implement REQUIRED (or similar words) support that requirement.

Should we also add a restriction that allows INCLUDED to allocate data space after the interpretation of the included file ends?

The change in the definition fixes an obvious copy-and-paste error.

C.7.2 REQUIRED syntax

The syntax follows the good example of INCLUDED in being non-parsing. That syntax allows more flexible uses and it allows REQUIREing filenames containing spaces.

C.7.3 INCLUDE

INCLUDE is implemented and used widely, so we might just as well standardize it (and why not in this RfD). OTOH, once REQUIRE is widely implemented, it might see much less use. If enough people argue against its inclusion, I will remove it from this RfD.

C.7.4 INCLUDE and BASE

There are supposedly systems that set BASE to decimal at the start of INCLUDE. One could accommodate such systems by adding a restriction:

> An ambiguous condition exists if the value of BASE is not (decimal) ten.

However, IMO it is confusing if INCLUDE silently changes BASE and INCLUDED does not, so I will only add this restriction if somebody can name such a system (that is a necessary, not a sufficient condition).

A better approach IMO is to warn the user if BASE is not decimal when starting INCLUDE, and if the BASE has changed when ending INCLUDE (and probably for INCLUDED and INCLUDE-FILE, too).

C.7.5 REQUIRE

Given what happened with INCLUDED and INCLUDE, we might just as well provide a parsing variant of REQUIRED right from the start. However, the essential word is REQUIRED, the parsing variant is just syntactic sugar.

C.7.6 REQUIRE name

The name is modelled on the relation between INCLUDED and INCLUDE, and also on Emacs Lisp's require. However, this word has been implemented under other names: NEEDS in various systems, REQUIRES in PFE.

What's worse, some systems have used these names for other purposes: MPE's systems have REQUIRES with a different meaning, PFE has a NEEDS with a different meaning, and ciforth has a REQUIRE with a different meaning (but ciforth's author thinks that ciforth is not significant and has promised to rename/drop his word if consensus on using REQUIRE for another purpose is reached).

So, which of the names would you prefer? Please post or mail me your preferences, and I will collect them and then decide on the final name, if any. Also, if you know of other conflicts for these names, please let me know.

Results from the RfDs

- Stephen Pelc (MPE) can live with REQUIRED and REQUIRE. REQUIRES would conflict with MPE's systems.
- Alex McDonald (Win32Forth) can live with REQUIRE currently Win32Forth supports NEEDS and REQUIRES (with ideas to drop REQUIRES).

C.7.7 Why not use load screens?

Some people prefer to have a single file that contains INCLUDEs for all the other files in a program. These people do not need REQUIRED.

However, other people want to build programs out of reusable (and possibly independently developed) libraries, and the load-screen approach causes increased maintenance work in this context: E.g., if a new version of a library needs to load an additional sublibrary, the load screens of all programs using the library would have to be changed. In contrast, with REQUIRED, the library REQUIREs the sublibrary itself, and REQUIRED makes sure that it is not loaded twice.

C.7.8 What about the C approach

Some might say: Why not use the C solution: The C solution to this problem is putting a wrapper like

```
#ifndef FILE_H
#define FILE_H
...
#endif
```

around every source file. This is inefficient (the whole file has to be read again, unless the compiler does some pretty sophisticated stuff), and requires cooperation from the author of the file (which is problematic, because not the author, but the users of the file have the trouble).

SwiftForth has a word \\that can be used to mostly eliminate the inefficiency mentioned above: it ends interpreting the current file; Win32Forth has an equivalent word \s. SwiftForth also has a word OPTIONAL that supports the C-like technique.

C.7.9 Using the dictionary for implementation

One thing that could be done to make the implementations simpler is to specify that INCLUDED and REQUIRED can change the dictionary before or after text-interpreting the file; that would allow the implementation to store the names of the included files in a wordlist. The downside would be that INCLUDED could not be used for including parts of colon definitions or the data part of CREATEd words. However, such uses are probably rare (if they occur at all in existing code), and they can still be performed (although in a more cumbersome way) with INCLUDE-FILE. What do you think?

C.7.10 What about MARKER and FORGET?

The proposed behaviour is probably what the users are expecting, and AFAIK all implementations of such words already support that behaviour. The ambiguous condition allows different implementation approaches, and the usage that is explicitly unspecified in this way probably does not matter to users.

C.8 Experience

Many systems have REQUIRED, REQUIRE, REQUIRES, and/or NEEDS. Many systems contain INCLUDE. The occurrences of these words in the Gforth sources are:

occurences	regexp
88	"^require "
45	"^include "
4	" included$"
2	" required$"

C.9 Implementation and Tests

- Reference implementation[6] (does not deal as proposed with markers and `FORGET`).
- Tests[7].

C.10 Comments

- Albert van der Horst suggested that `REQUIRE` should in some way be able to use `LOAD`. However, I believe that such a word, if needed, should have a different name.
- Hans Bezemer writes that 4th has a parsing `[NEEDS` (apparently used like "`[NEEDS` file name `]`"). 4th also has `INCLUDE`.
- Stephen Pelc:

  > In order to ease use of library code, it may be as well to consider specifying how the following items are handled when the file/source is `INCLUDED`: `BASE` `decimal`? search order
  > ...

- Phil Budne writes that SNOBOL4 has a `-INCLUDE` that behaves like the proposed `REQUIRE`.

from an earlier (Pre-RfD) version of the proposal

- Michael L. Gassanenko:

  > Yes. I do use NEEDS.

- Peter Knaggs:

  > Perl introduced a version of `REQUIRED` some time ago that works exactly in this manner. I agree very much with its usefulness, indeed it would allow a standard "library" model. Having said that, it can be defined using standard ANS.

- Guido Draheim:
  - Both Forth.com's Swiftforth and MPE's ProforthVFX have defined 'requires' as the selfparsing version of 'required', and they use it [It turns out that `REQUIRES` in MPE's systems has a different meaning]. PFE will adopt this in the next version (> 0.30.30)
  - The specification is non-deterministic in its stack effect — the user of 'required' has no way to check in advance if a file will get 'included'. Many systems use a word like 'loaded¿ but it can not as easily specified for many systems just as it is done for 'required' itself, where the form of 'requires' has found wide acceptance. It may be useful to **allow** the use CSP-like techniques to ensure a deterministic stack-effect, and it may be useful to **recommend** that at least a warning message is shown for the case of stack-depth differences.

[6] http://www.complang.tuwien.ac.at/forth/ansforth/reference-implementations/required.fs
[7] http://www.complang.tuwien.ac.at/forth/ansforth/tests/required.fs

D Escaped strings

D.1 Author

Stephen Pelc

D.2 Change History

2006-08-22 Updated solution section.

2006-08-21 First draft.

D.3 Problem

The word `S"` 6.1.2165 is the primary word for generating strings. In more complex applications, it suffers from several deficiencies:

1. the `S"` string can only contain printable characters,
2. the `S"` string cannot contain the '"' character,
3. the `S"` string cannot be used with wide characters as dicussed in the Forth 200x internationalisation and XCHAR proposals.

D.4 Current practice

At least SwiftForth, gForth and VFX Forth support `S\"` with very similar operations. `S\"` behaves like `S"`, but uses the '\' character as an escape character for the entry of characters that cannot be used with `S"`.

This technique is widespread in languages other than Forth.

It has benefit in areas such as

1. construction of multiline strings for display by operating system services,
2. construction of HTTP headers,
3. generation of GSM modem control strings.

The majority of current Forth systems contain code, either in the kernel or in application code, that assumes char=byte=au. To avoid breaking existing code, we have to live with this practice.

D.5 Considerations

We are trying to integrate several issues:

1. no/least code breakage
2. minimal standards changes
3. variable width character sets
4. small system functionality

Item 1 is about the common char=byte=au assumption.

Item 2 includes the use of `COUNT` to step through memory and the impact of char in the file word sets.

Item 3 has to rationalise a fixed width serial/comms channel with 1ldots4 byte characters, e.g. UTF-8

Item 4 should enable 16 bit systems to handle UTF-8 and UTF-32.

The basis of my current approach is to use the terminology of primitive characters and extended characters. A primitive character (called a *pchar* here) is a fixed-width unit handled by `EMIT` and friends. It corresponds to the current ANS definition of a character. An extended character (called an *xchar* here) consists of one or more primitive characters and represents the encoding for a "display unit". A string is represented by caddr/len in terms of primitive characters.

The consequences of this are:

1. No existing code is broken.
2. Most systems have only one keyboard and only one screen/display unit, but may have several additional comms channels. The impact of a keyboard driver having to convert Chinese or Russian characters into a (say) UTF-8 sequence is minimal compared to handling the key stroke sequences. Similarly on display.
3. Comms channels and files work as expected.
4. 16-bit embedded systems can handle all character widths as they are described as strings.
5. No conflict arises with the XCHARs proposal.

Multiple encodings can be handled if they share a common primitive character size — nearly all of these are described in terms of octets: TCP/IP, UTF-8, UTF-16, UTF-32, ...

The XCHARs proposal can be used to handle extended characters on the stack. XEMIT and friends allow us to handle some additional odd-ball requirements such as 9-bit control characters, e.g. for the MDB bus used by vending machines.

D.6 Solution

To ease discussion we refer to character handled by `C@`, `C!` and friends as "primitive characters" or *pchars*. Characters that may be wider than a *pchar* are called "extended characters" or *xchars*. These are compatible with the XCHARs proposal. This proposal does note requires systems to handle *xchars*, but does not disenfranchise those that do.

`S\"` is used like `S"` but treats the '\' character specially. One or more characters after the '\' indicate what is substituded. The following list is what is currently available in the Forth systems surveyed.

\a	BEL	alert, ASCII 7
\b	BS	backspace, ASCII 8
\e	ESC	not in C99, ASCII 27
\f	FF	form feed, ASCII 12
\l	LF	ASCII 10
\m	CR/LF pair	ASCII 13, 10 — for HTML etc.
\n	newline	CRLF for Windows/DOS, LF for Unices
\q	double-quote	ASCII 34
\r	CR	ASCII 13
\t	HT	tab, ASCII 9
\v	VT	ASCII 11
\z	NUL	ASCII 0
\"	"	
\[0-7]+	Octal numerical character value, finishes at the first non-octal character	
\x[0-9a-f]+	Hex numerical character value, finishes at the first non-hex character	
\\	backslash itself	
\	before any other character represents that character	

The following three of these cause parsing and readability problems. As far as I know, requiring characters to come in 8 bit units will not upset any systems. Systems with characters less than 7 bits are non-compliant, and I know of no 7 bit CPUs. All current systems use character units of 8 bits or more.

\[0-7]+	Octal numerical character value, finishes at the first non-octal character
\x[0-9a-f]+	Hex numerical character value, finishes at the first non-hex character

Why do we need two representations, both of variable length? This proposal selects the hexadecimal representation, requiring two hex digits. A consequence of this is that *xchars* must be represented as a sequence of *pchars*. Although initially seen as a problem by some people, it avoids at least the following problems:

1. Endian issues when transmitting an *xchar*, e.g. big-endian host to little-endian comms channel
2. Issue when an *xchar* is larger than a cell, e.g. UTF-32 on a 16 bit system.
3. Does not have problems in distinguishing the end of the number from a following character such as '0' or 'A'.

At least one system (Gforth) already supports UTF-8 as it's native character set, and one system (JaxForth) used UTF-16. These systems are not affected.

\	before any other character represents that character

This is an unnecessary general case, and so is not mandated. By making it an ambiguous condition, we do not disenfranchise existing implementation, and leave the way open for future extensions.

D.7 Proposal

6.2.xxxx `S\"` s-slash-quote CORE EXT

Interpretation: Interpretation semantics for this word are undefined.

Compilation: ("*ccc*⟨quote⟩" --)

Parse *ccc* delimited by " (double-quote), using the translation rules below. Append the run-time semantics given below to the current definition.

Translation rules: Characters are processed one at a time and appended to the compiled string. If the character is a '\' character it is processed by parsing and substituting one or more characters as follows:

\a	BEL	alert, ASCII 7
\b	BS	backspace, ASCII 8
\e	ESC	not in C99, ASCII 27
\f	FF	form feed, ASCII 12
\l	LF	ASCII 10
\m	CR/LF pair	ASCII 13, 10
\n	newline	implementation dependent newline e.g., CR/LF, LF or LF/CR
\q	double-quote	ASCII 34
\r	CR	ASCII 13
\t	HT	tab, ASCII 9
\v	VT	ASCII 11
\z	NUL	ASCII 0
\"	"	
\x*AB*	*A* and *B* are Hexadecimal numerical characters. The resulting character is the conversion of these two characters.	
\\	backslash itself	
\	before any other character constitutes an ambiguous condition.	

Run-time: (-- *c – addr u*)
Return *c – addr* and *u* describing a string consisting of the translation of the characters *ccc*. A program shall not alter the returned string.

See: 3.4.1 Parsing, 6.2.0855 `C"`, 11.6.1.2165 `S"`, A.6.1.2165 `S"`.

D.8 Labelling

- `ENVIRONMENT?` impact
  name stack conditions
- Ambiguous conditions occur:
  - If a hex value is more than two characters
  - If \x is not followed by by two hexadecimal characters
  - If \is followed by a character other than those listed in `S\"` (a, b, e, f, l, m, n, q, r, t, v, z, ", x or \).

D.9 Reference Implementation

(as yet untested)

Taken from the VFX Forth source tree and modified to remove most implementation dependencies. Assumes the use of the # and $ numeric prefices to indicate decimal and hexadecimal respectively.

```
decimal

: PLACE         \ c-addr1 u c-addr2 --
\ *G Copy the string described by c-addr1 u to a counted string at
\ ** the memory address described by c-addr2.
  2dup 2>r \ write count last
  1 chars + swap move
  2r> c! \ to avoid in-place problems
```

```
;

: $,\ caddr len --
\ *G Lay the string into the dictionary at *\fo{HERE}, reserve
\ ** space for it and *\fo{ALIGN} the dictionary.
  dup >r
  here place
  r> 1 chars + allot
  align
;

: addchar       \ char string --
\ *G Add the character to the end of the counted string.
  tuck count + c!
  1 swap c+!
;

: append \ c-addr u $dest --
\ *G Add the string described by C-ADDR U to the counted string at
\ ** $DEST. The strings must not overlap.
  >r
  tuck  r@ count +  swap cmove          \ add source to end
  r> c+!                                \ add length to count
;

: extract2H \ caddr len -- caddr' len' u
\ *G Extract a two-digit hex number in the given base from the
\ ** start of the* string, returning the remaining string
\ ** and the converted number.
  base @ >r  hex
  0 0 2over >number 2drop drop
  >r  2 chars /string r>
  r> base !
;

create EscapeTable \ -- addr
\ *G Table of translations for \a..\z.
  7 c,\ \a
  8 c,\ \b
  char c c,\ \c
  char d c,\ \d
  #27 c,\ \e
  #12 c,\ \f
  char g c,\ \g
  char h c,\ \h
  char i c,\ \i
  char j c,\ \j
  char k c,\ \k
  #10 c,\ \l
  char m c,\ \m
  #10 c,\ \n (Unices only)
  char o c,\ \o
  char p c,\ \p
  char " c,     \ \q
```

```
  #13 c,\ \r
  char s c,\ \s
  9 c,\ \t
  char u c,\ \u
  #11 c,\ \v
  char w c,\ \w
  char x c,\ \x
  char y c,\ \y
  0 c,\ \z

create CRLF$ \ -- addr ; CR/LF as counted string
 2 c,  #13 c,  #10 c,

internal
: addEscape \ caddr len dest -- caddr' len'
\ *G Add an escape sequence to the counted string at dest,
\ ** returning the remaining string.
  over 0= \ zero length check
  if  drop  exit  endif
  >r \ -- caddr len ; R: -- dest
  over c@ [char] x = if \ hex number?
    1 chars /string extract2H r> addchar  exit
  endif
  over c@ [char] m = if \ CR/LF pair?
    1 chars /string  #13 r@ addchar  #10 r> addchar  exit
  endif
  over c@ [char] n = if \ CR/LF pair?
    1 chars /string  crlf$ count r> append  exit
  endif
  over c@ [char] a [char] z 1+ within if
    over c@ [char] a - EscapeTable + c@  r> addchar
  else
    over c@ r> addchar
  endif
  1 chars /string
;
external

: parse\" \ caddr len dest -- caddr' len'
\ *G Parses a string up to an unescaped '"', translating '\'
\ ** escapes to characters much as C does. The
\ ** translated string is a counted string at *\i{dest}
\ ** The supported escapes (case sensitive) are:
\ *D \a      BEL (alert)
\ *D \b      BS (backspace)
\ *D \e      ESC (not in C99)
\ *D \f      FF (form feed)
\ *D \l      LF (ASCII 10)
\ *D \m      CR/LF pair - for HTML etc.
\ *D \n      newline - CRLF for Windows/DOS, LF for Unices
\ *D \q      double-quote
\ *D \r      CR (ASCII 13)
\ *D \t      HT (tab)
\ *D \v      VT
```

```
\ *D \z      NUL (ASCII 0)
\ *D \"      "
\ *D \xAB    Two char Hex numerical character value
\ *D \\      backslash itself
\ *D \       before any other character represents that character
  dup >r  0 swap c! \ zero destination
  begin \ -- caddr len ; R: -- dest
    dup
   while
    over c@ [char] " <>\ check for terminator
   while
    over c@ [char] \ = if \ deal with escapes
      1 /string r@ addEscape
    else \ normal character
      over c@ r@ addchar  1 /string
    endif
  repeat then
  dup  \ step over terminating "
  if  1 /string  endif
  r> drop
;

: readEscaped \ "string" -- caddr
\ *G Parses an escaped string from the input stream according to
\ ** the rules of *\fo{parse\"} above, returning the address
\ ** of the translated counted string in *\fo{PAD}.
  source >in @ /string tuck \ -- len caddr len
  pad parse\" nip
   - >in +!
  pad
;

: S\" \ "string" -- caddr u
\ *G As *\fo{S"}, but translates escaped characters using
\ ** *\fo{parse\"} above.
  readEscaped count  state @ if
    compile (s") $,
  then
; IMMEDIATE
```

D.10 Test Cases

TBD.

E Front Matter

E.1 Author

Peter Knaggs

E.2 Change History

2006-09-08 Modified to reflect that the new document is based on the final *draft* of the 1999 ANSI review.
Added sections 7 and 8.
Modified the proposed Forward.

2006-08-30 First draft.

E.3 Problem

The basis document, as published, is too heavily based on the original ANS document. The first two parts of the proposal update appendix B to include reference to the ANS and ISO standards. Much of appendix B need to be updated, but I will leave that to the interested parties.

The front matter refers too directly to the ANS procedure. Parts 4 and 5 replace the front matter with material more appropriate to the aim of the document. Part 5 is a description of the RfD/CfV procedure for including proposals into the basis document / revised standard.

Finally part 6 is a copy of the minutes from the first standards meeting. Whether this should be included in the basis document, let alone in the front matter is a topic of debate. Can we simply leave this on the web site, or do we need a printed record. Is there any ware else we can publish the minutes? JFAR or the ACM are both possibilities.

E.4 Proposal

1. Remove the last line of the second paragraph in Appendix B.

   "Several members of the Forth Standards Team have also been members of the X3J14 Technical Committee."

2. Add the following to the end of the "Industry standards" section of Appendix B.

   > The following standards where developed under the control of the ANSI. The committee drawing up the ANSI standard included several members of the Forth Standards Team.
   > ANSI X3.215-1994 *Information Systems — Programming Language FORTH*
   > ISO/IEC 15145:1997 *Information technology. Programming languages. FORTH*

3. Remove the "X3 Membership" and "X3J14 Membership" sections, as ISO did, when they adopted the ANSI document.

4. Replace the "Forward" with the following:

   > Forth is a language for direct communication between human beings and machines. Using natural-language diction and machine-oriented syntax, Forth provides an economical, productive environment for interactive compilation and execution of programs. Forth also provides low-level access to computer-controlled hardware, and the ability to extend the language itself. This extensibility allows the language

to be quickly expanded and adapted to special needs and different hardware systems.

Forth was invented by Mr. Charles Moore to increase programmer productivity without sacrificing machine efficiency. Forth is a layered environment containing the elements of a computer language as well as those of an operating system and a machine monitor. This extensible, layered environment provides for highly interactive program development and testing.

In the interests of transportability of application software written in Forth, standardization efforts began in the mid-1970s by an international group of users and implementors who adopted the name "Forth Standards Team". This effort resulted in the Forth-77 Standard. As the language continued to evolve, an interim Forth-78 Standard was published by the Forth Standards Team. Following Forth Standards Team meetings in 1979, the Forth-79 Standard was published in 1980. Major changes were made by the Forth Standards Team in the Forth-83 Standard, which was published in 1983.

In 1987, the ANSI (American National Standards Institute) and the IEEE (Institute of Electrical and Electronics Engineers) convened the X3J14 committee on Forth Programming Systems. The committee included many members of the Forth Standards Team. After some twenty-three meetings (88 days) the committee published ANS X3.215-1994 *Information Systems — Programming Languages FORTH* in 1994. This document was adopted as an international standard, by the ISO, in 1997 and published as ISO/IEC 15145:1997 *Information technology, Programming languages, FORTH.*

ANSI launched a review of the standard in 1999. A number of alterations where proposed, however the committee eventually agreed to leave 1994 document unchanged.

The current project to update the ANS Forth standard was launched at the 2004 EuroForth conference. Proposals to be posted and discussed via the `comp.lang.forth` usenet news group and an email list (`forth200x@yahoogroups.com`. With an open meetings to discuss proposals, held immediately before the annual EuroForth conference.

This document is based on the final draft of the of the standard published by Technical Committee on Forth Programming Systems as part of the 1999 review. It has been modified in accordance with the directions of the Forth 200x Standards Committee which first met on October 21–23, 2005.

5. Add a new "Proposals Process" section after the "Forward":

   In developing a standard it is necessary for the standards committee to know what the system implementors and the programmers are already doing in that area, and what they would be willing to do, or wish for.

   To that end we have introduced a system of consultation with the Forth community:

   (a) A proponent of an extension or change to the standard write a proposal

   (b) This proposal is published on `comp.lang.forth` usenet news group and on the `forth200x` email list as an RfD (Request for Discussion) where the Forth community can comment on the proposal.

   (c) The proponent can modify the proposal, taking the comments into consideration. Where they dismiss comments, the reasons for this dismissal must be given. The revised proposal is republished as a new RfD.

   (d) Once a proposal has settled down, it is republished as a CfV (Call for Votes). System implementors state, whether their system implements the proposal,

or what the chances are that it ever will. Similarly, programmers can state whether they have used something similar to the proposed extension and whether they would used the proposed extension once it is standardized.

(e) After a deadline, a preliminary poll result will be published, but the poll will remain open for adding to the Web page (especially system implementors are invited to do that).

Note that should a contributor consider their comments to have been dismissed without due consideration, they are encouraged to submit a counter proposal.

Proposals which have passed the poll will be integrated into the basis document in preparation for the approaching Standards Committee meeting. No proposal with a CfV deadline of less than three months prior to the next standards meeting will be integrated into the basis document.

A proposal should contain the following

- Normative parts
  This is the formal part of the proposal.
  Proposal

  The actual proposal should be as well-specified as possible.
  Some issues could be left undecided, left for the discussion of the proposal. These issues should be mentioned in the Remarks section in addition to the proposal.
  If you want to leave something open to the system implementor, make that explicit, indicating it as implementation specific, or possibly by making it an ambiguous condition.
  For word definitions, take you example from the basis document. The definition should include validation and rationale sections, as appropriate.
- Informative parts
  The rest of the proposal is informative, giving a rationale for the proposal, so that system implementors and programmers will see what it is good for and why they should adopt it (and vote for it). This includes:

  Problem
  This states what problem the proposal addresses. It usually comes before the (normative) proposal.

  Typical use
  Shows a typical use of the word/feature you propose; this should make the formal wording easier to understand.

  Remarks
  This gives the rationale for specific decisions you have taken in the proposal, or discusses specific issues that have not been decided yet.

  Reference implementation
  This makes it easier for system implementors to adopt your proposal. Where possible a reference implementation should be provided in ANS Forth. Where this is not possible, system specific knowledge is required or non standard words are used, this should be documented.

  Test cases
  This should test the feature/words you propose, in particular, it should test boundary conditions. Where possible test cases should be written to the ANS Forth validation suite, as developed by John Hayes, and documented in appendix F of this document.

  Experience
  Indicate where the proposal has already been implemented and/or used.

Comments

Initially this is blank. As comments are made on the proposal, they should be incorporated into the proposal. Comment which can not be incorporated should be included in this section. A response to the comment may be included after the comment itself.

Instructions for voting

Once the proposal enters the Call for Votes stage, the vote-taker will add the voting instructions to the proposal. The vote-taker will normally be a member of the standards committee not directly involved in the drafting of the proposal.

6. Add a new "Minutes of Meetings" section after the new "Proposals Process" section.

   Note that as this is front-matter it will appear before the table of contents and thus the page numbering of the main document will not be effected.

Minutes of the Forth200*x* Standards Committee #1

October 21–23, 2005, Santander, Spain

Participants

Willem Botha	CCS, South Africa	
Federico de Ceballos	University of Cantabria, Spain	
M. Anton Ertl	Technische Universität Wien, Austria	(Chair)
N. J. Nelson	Micross, England	(Observer)
Dr. Peter Knaggs	University of Bournemouth, England	
Stephen Pelc	MPE, England	
Dr. Bill Stoddart	University of Teesside, England	(Observer)

Action on proposals

Proposal	Comments	Status	Action
X:deferred	`DEFER`, `DEFER@`, `DEFER!`, `IS`, `ACTION-OF`	Passed 5/0/0	
X:defined	`[DEFINED]` and `[UNDEFINED]`	Passed 5/0/0	
X:parse-name	`PARSE-NAME`	Passed 5/0/0	
X:extension-query	Modification to `ENVIRONMENT?`, revised to remove auto-loading of requested extensions	Passed 5/0/0	

Other Business

(a) Integration into the standards document
Should new words be put into the established wordsets, or into new ones?
The eventual goal is to usually integrate the new words into existing wordsets with related functionality; in some cases it may be more appropriate to create a new word set. However, as an intermediate step the new proposals will at first be kept separate, to make it easier for readers of the document to see what is changing.

(b) How are the extension-query names reflected in the standard (if at all)?
The glossary header for new words includes the extension-query string for the extension that proposed it. In addition, there will be a chapter or normative appendix that lists all the extensions, their extension-query strings and the components (word definitions etc.) that it consists of.

(c) Should the tests of a proposal or the reference implementation become normative?
No. This could lead to conflicting normative sections; also, making the reference implementation normative would lead to over specification.

(d) Review of the RfD/CfV process
How well is the RfD/CfV process working at generating high- quality proposals for standardisation and getting information about their popularity? What could be improved? Or should we do something completely different?

Many of the participants were not very familiar with how well the process worked in practice, and had no suggestions for improvements.
The (normative part of the) proposals required adoption before integrating them into the document, but there was a widespread feeling among the participants that proposals in the form of unambiguous instructions to the document editor (which is the form that would be voted on by the standards committee) would be harder to understand for the CfV audience.
The resulting idea was to have two forms of the proposal, with the same content: First the not-so-formal form used in the CfV, and later a form for integrating it into the document.

(e) Official standards body
Should we run the standard through a standards body like ANSI, ISO, IEEE, etc.? If so, which one?
Some participants consider the blessing of the future standards document by an official standards body very important, and we agreed to work towards this goal by writing the document in the appropriate style, and by keeping documentation about all our steps. However, the general idea was to first develop the document without involving a standards body, and deal with them at the end.
Various candidate standards bodies were discussed; none was decided on, but it might be that going through ANSI again might be the easiest route.

(f) Chairman and Editor M. Anton Ertl was appointed as the chair of the committee unopposed.
Dr. Peter Knaggs was appointed editor unopposed.

(g) Which standard documents should we start from?
Peter Knaggs has a version of dpANS99a in LaTeX form, convertible into a fully hyperlinked PDF file. However, it is yet unclear how this document differs from the ANS/ISO Forth documents and dpANS6.

(h) Integration of CfV into the Standard
How do we get from the CfV proposal to the form for integration into the document? One opinion was that the original proponent should do it.

(i) Proposal Champions
There was the opinion that a proposal (de-facto) needs a champion in the committee to get approval by the standards committee. So, if the proponent finds a champion in the committee, they could produce the for-the-document version of the proposal together.

(j) Improvements for future standard meetings
- The participants should familiarise themselves with the proposals beforehand.
- Paper printouts of the proposals should be available at the meeting.
- Proposals should be available in the form needed for integration into the standards document in addition to the CfV form.
- There should be a champion for the proposal in the meeting.

Action Items

(a) Federico de Ceballos was asked to look into providing a proposal to cover the following topics:
- number prefixes
- `0` for `NIL`
- `!` and `@` for 16-bit and 32-bit signed and unsigned integers, bytes, octets

(b) M. Anton Ertl was asked to look into the implications of the following and develop proposals for them:

- separate FP stack
- required
- directory stuff in general
- directory handling for included and required
- key names for`EKEY` results

(c) Stephen Pelc was asked to look into the implication of the following topics, and develop proposals for:

- { `(locals)`, fp locals, buffer locals
- `S\" .\"`
- iors can be `THROW`n
- `SYNONYM`
- structures

(d) Bill Stoddart was asked to investigate the implications of allowing the parser to ignore white space (TAB, CR, LF, and FF) in source code, and develop a proposal.

(e) Peter Knaggs to contact John Hayes to obtain permission to include the validation suite into the document and to produce a basis document based on the ANS Forth standard with the four accepted proposals and validation suite included.

Next Meeting

It was agreed that the next meeting would take place immediately before the next euroForth conference to be held in Cambridge, England.

7. Replace the title on page 1:

   American National Standard for Information Systems
   Programming Language
   Forth

   with

   Forth200*x* Standards Committee
   Forth2006.2 Draft Standard

8. Replace the title on the title page:

   Forth 200*x*

   with

   Forth 200*x* Draft

E.5 Remarks

Do we need a printed copy of the minutes from the standards meetings? My personal view is that we do, so we can have independent documentation of the procedures we have undertaken in the development of the standard.

As the most significant word in the previous paragraph is "independent", the question come as to where the minutes should be published. I could published them in JFAR, but as the editor of both Forth200x and JFAR, I would hardly call that "independent". Any other suggestions.

I have reformatted the minutes published on forth200x.org, in the same style as those of the X3J14 minutes published in JFAR.

F Floating Values

F.1 Author

Marcel Hendrix

F.2 Change History

2006-08-25 First draft.

F.3 Rationale

F.3.1 Problem

The word `VALUE` was considered useful enough to be included in Forth94. A search through 4827 source files shows 532 occurrences of `VARIABLE` versus 4241 uses of `VALUE`. It would be obviously useful to have a variant of `VALUE` that works for floating-point values.

The main idea behind `FVALUE` is that fetching a variable is more frequent than changing it. Therefore both readability of source code and efficiency of execution can be achieved by making '`F@`' the default action of `FVALUE`.

F.3.2 Current practice

The proposed form of `FVALUE` has been in use in tForth, iForth, and their predecessors since 1985. `FVALUE` (with `TO`) is also in use for Mops and MacForth. The systems mentioned define more `TO`-like words that work on values, e.g. `+TO -TO CLEAR`, with obvious meanings.

This proposal does not propose to standardize on these extensions.

F.3.3 Solution

Although many people have objected to parsing words, parsing permits the host system the most flexibility in implementation and is thus the preferred solution (see also `TO`).

The syntax is:

```
123e0 FVALUE fdata
```

to define `fdata` as a floating-point value initialized to 123e0.

To access the value of `fdata`:

```
fdata 2e0 F+ F. <cr> 125.000000  ok
```

To change `fdata`:

```
fdata 2e0 F+ TO fdata  fdata F. <cr> 125.000000  ok
```

F.4 Proposal

12.6.1.xxxx `FVALUE` "f-value" FLOATING

(F: *x* --) ("⟨spaces⟩*name*" --)

Skip leading space delimiters. Parse *name* delimited by a space. Create a definition for *name* with the execution semantics defined below, with an initial value equal to *x*. *name* is referred to as an "f-value".

***name* Execution:** (F: -- *x*)

Place *x* on the FP stack. The value of *x* is that given when *name* was created, until the phrase *x* `TO` *name* is executed, causing a new value of *x* to be associated with *name*.

See: 3.4.1 *Parsing*.

6.2.2295 `TO` CORE EXT

Interpretation: (*x* "⟨spaces⟩*name*" --) or (F: *x* --) ("⟨spaces⟩*name*" --)

Skip leading spaces and parse name delimited by a space. Store *x* in *name*. An ambiguous condition exists if *name* was not defined by `VALUE` or `FVALUE`.

Compilation: ("⟨spaces⟩*name*" --)

Skip leading spaces and parse name delimited by a space. Append the run-time semantics given below to the current definition. An ambiguous condition exists if *name* was not defined by `VALUE` or `FVALUE`.

Run-time: (*x* --) or (F: *x* --)

Store *x* in *name*.

Note: An ambiguous condition exists if either `POSTPONE` or `[COMPILE]` is applied to `TO`.

See: 6.2.2405 `VALUE`, **12.6.1.xxxx** `FVALUE`, **13.6.1.2295** `TO`.

13.6.1.2295 `TO` LOCAL

Extend the semantics of **6.2.2295** `TO` to be:

Interpretation: (*x* "⟨spaces⟩*name*" --) or (F: *x* --) ("⟨spaces⟩*name*" --)

Skip leading spaces and parse *name* delimited by a space. Store *x* in *name*. An ambiguous condition exists if *name* was not defined by `VALUE` or `FVALUE`.

Compilation: ("⟨spaces⟩*name*" --)

Skip leading spaces and parse *name* delimited by a space. Append the run-time semantics given below to the current definition. An ambiguous condition exists if *name* was not defined by either `VALUE`, `FVALUE` or `(LOCAL)`.

Run-time: (*x* --) or (F: *x* --)

Store *x* in *name*.

Note: An ambiguous condition exists if either `POSTPONE` or `[COMPILE]` is applied to `TO`.

See: 3.4.1 *Parsing*, **6.2.2295** `TO`, **6.2.2405** `VALUE`, **12.6.1.xxxx** `FVALUE`, **13.6.1.0086** `(LOCAL)`.

F.5 Labeling

TBD

F.6 Reference Implementation

The implementation of FVALUE requires carnal knowledge of the host implementation, which is the main reason why it should be standardized.

The implementation below disregards the issue that TO should also work for integer VALUEs and locals.

```
: FVALUE  CREATE F, ( F: r -- )  DOES> F@ ; ( F: -- r )

: TO    ( F: r -- ) ( "<spaces>name" -- )
' >BODY
STATE @ IF  POSTPONE LITERAL  POSTPONE F!
      ELSE  F!
      THEN ; IMMEDIATE
```

F.7 Test Cases

```
  123e FVALUE fdata
    2   VALUE idata

: foo ( -- )
    idata LOCALS| ldata |
    ldata .  idata 2* TO ldata  ldata .
    idata 1 D>F TO fdata  fdata F.
    fdata 33e F+ F>D D>S TO idata  idata .
    ldata idata + fdata F>D D>S + . ;

foo <cr> 2 4 4.294967e9 35 41  ok
```

G Enhanced local variable syntax

G.1 Author

Stephen Pelc

G.2 Change History

2006-08-22 Added explanatory text.
Corrected reference implementation.
Updated ambiguous conditions.

G.3 Problem

1. The current `LOCALS| ... |` notation explicitly forces all locals to be initialised from the data stack.
2. The current `LOCALS| ... |` notation defines locals in reverse order to the normal stack notation.
3. When programming large applications, especially those interfacing with a host operating system, there is a frequent need for temporary buffers.
4. Current implementations show that creation and destruction of local buffers are much faster than using `ALLOCATE` (14.6.1.0707) and `FREE` (14.6.1.1605).

G.4 Solution

G.4.1 Base version

The following syntax for local arguments and local variables is proposed. The sequence:

`{` ni_1 ni_2 ... `|` lv_1 lv_2 ... `--` o_1 o_2 `}`

defines local arguments, local variables, and outputs. The local arguments are automatically initialised from the data stack on entry, the rightmost being taken from the top of the data stack. Local arguments and local variables can be referenced by name within the word during compilation. The output names are dummies to allow a complete stack comment to be generated.

The items between `{` and `|` are local arguments.
The items between `|` and `--` are local variables or buffers.
The items between `--` and `}` are outputs for formal comments only.

The outputs are provided in the notation so that complete stack comments can be produced. However, all text between `--` and `}` is ignored. The facility is there to permit the notation to form a complete stack comment. This eases documentation and current users of the notation like this facility.

Local arguments and variables return their values when referenced, and must be preceded by `TO` to perform a store.

Local buffers may be defined in the form:

arr[⟨expr⟩]

Any name ending in the '[' character will be treated as a buffer, the expression up to the terminating ']' will be interpreted to provide the size of the buffer. Local buffers only return their base address, all operators such as TO generate an ambiguous condition.

In the example below, a and b are local arguments, a+b and a*b are local variables, and arr[is a 10 byte local buffer.

```
: foo  { a b | a+b a*b arr[ 10 ] -- }
  a b + to a+b
  a b * to a*b
  cr a+b .  a*b .
  arr[ 10 erase
  s" Hello" arr[ swap cmove
;
```

G.4.2 Local types

Some current Forth systems use indicators to define local variables of sizes other than a cell. It is proposed that any name ending in a ':' (colon) be reserved for this use.

```
: foo  { a b | F: f1 F: f2 -- c }
  ...
;
```

G.5 Discussion

The '|' (ASCII 0x7C) character is widely used as the separator between local arguments and local variables. Other characters accepted in current Forth implementations are '\' (ASCII 0x5C) and '||' (ASCII 0xA6). Since the ANS standard is defined in terms of 7 bit ASCII, and with regard to internationalistion, we propose only to consider the '|' and '\' characters further. Only recognition of the '|' separator is mandatory.

The use of local types is contentious as they only become useful if TO is available for these. In practice, many current systems permit TO to be used with floats (children of FVALUE) and other data types. Such systems often provide additional operators such as +TO (add from stack to item) for children of VALUE and FVALUE. Standardisation of operators with (for example) floats needs to be done before the local types extension can be incorporated into Forth200x. Apart from forcing allocation of buffer space, no additional functionality is provided by local types that cannot be obtained using local buffers. More preparatory standardisation needs to be done before local types.

Apart from { (brace) itself, the proposal introduces one new word BUILDLV. The definition of this word is designed for future enhancements, e.g. more local data types, without having to introduce more new words.

G.6 Proposal

13.3 and **13.4**

Add "and BUILDLV" where (LOCAL) is referenced.

13.6.2.xxxx { "brace" LOCAL EXT

Interpretation: Interpretation semantics for this word are undefined.

Compilation: (“⟨spaces⟩arg_1” ... “⟨spaces⟩arg_n” `|` “⟨spaces⟩lv_1” ... “⟨spaces⟩lv_n” --)

Create up to eight local arguments by repeatedly skipping leading spaces, parsing *arg*, and executing **13.6.2.yyyy** `BUILDLV`. The list of local arguments to be defined is terminated by “`|`”, “`--`” or “`}`”. Append the run-time semantics for local arguments given below to the current definition. If a space delimited ‘`|`’ is encountered, create up to eight local variables or buffers by repeatedly skipping leading spaces, parsing *lv*, and executing **13.6.2.yyyy** `BUILDLV`. The list of local variables and buffers to be defined is terminated by “`--`” or “`}`”. Append the run-time semantics for local variables and local buffers given below to the current definition. If “`--`” has been encountered, further text between “`--`” and `}` is ignored.

Local buffers have names that end in the ‘`[`’ character. They define their size by parsing the text string up to the next ‘`]`’ character, and passing that string to **7.6.1.1360** `EVALUATE` to obtain the size of the storage in address units.

Local *argument* run-time: (x_1 ... x_n --)

Local *variable* run-time: (--)

Local *buffer* run-time: (--)

Initialize up to eight local arguments as described in **13.6.2.yyyy** `BUILDLV`. Local argument arg_1 is initialized with x_1, arg_2 with x_2 up to arg_n from x_n, which is on the top of the data stack. When invoked, each local argument will return its value. The value of a local argument may be changed using **13.6.1.2295** `TO`.

Initialize up to eight local variables or local buffers as described in **13.6.2.yyyy** `BUILDLV`. The initial contents of local variables and local buffers are undefined. When invoked, each local variable returns its value. The value of a local variable may be changed using **13.6.1.2295** `TO`. The size of a local variable is a cell. When invoked, each local buffer will return its address. The user may make no assumption about the order and contiguity of local variables and buffers in memory.

Ambiguous conditions:

1. The `{ ... }` text extends over more than one line.
2. The expression for local buffer size does not return a single cell.
3. `{ ... }` is declared more than once in a word.
4. Parsing units ‘`|`’, ‘`]`’, ‘`--`’ and ‘`}`’ are not white-space delimited.

13.6.2.yyyy `BUILDLV` “build-l-v” LOCAL EXT

Interpretation: Interpretation semantics for this word are undefined.

Execution: ($c-addr$ u $+n$ *mode* --)

When executed during compilation, `BUILDLV` passes a message to the system identifying a new local argument whose definition name is given by the string of characters identified by $c-addr$ u. The size of the data item is given by $+n$ address units, and the *mode* identifies the construction required as follows:

0 finish construction of initialisation and data storage allocation code. $C-addr$ and u are ignored. $+n$ is 0 (other values are reserved for future use).

1 identify a local argument, $+n$ = cell

2 identify a local variable, $+n$ = cell

3 identify a local buffer, $+n$ = storage required.

4+ reserved for future use

-ve implementation specific values

The result of executing BUILDLV during compilation of a definition is to create a set of named local arguments, variables and/or buffers, each of which is a definition name, that only have execution semantics within the scope of that definition's source.

local *argument* execution: (-- *x*)

Push the local argument's value, *x*, onto the stack. The local argument's value is initialized as described in **13.6.2.xxxx** { and may be changed by preceding the local argument's name with TO.

local *variable* execution: (-- *x*)

Push the local variables' value, *x*, onto the stack. The local variable is not initialised. The local variable's value may be changed by preceding the local variable's name with TO.

local *buffer* execution: (-- *a* − *addr*)

Push the local buffer's address, *a* − *addr*, onto the stack. The address is aligned as in **3.3.3.1**. The contents of the buffer are not initialised.

Note: This word does not have special compilation semantics in the usual sense because it provides access to a system capability for use by other user-defined words that do have them. However, the locals facility as a whole and the sequence of messages passed defines specific usage rules with semantic implications that are described in detail in section **13.3.3** *Processing locals*.

Note: This word is not intended for direct use in a definition to declare that definition's locals. It is instead used by system or user compiling words. These compiling words in turn define their own syntax, and may be used directly in definitions to declare locals. In this context, the syntax for BUILDLV is defined in terms of a sequence of compile-time messages and is described in detail in section **13.3.3** *Processing locals*.

Note: The LOCAL EXT word set modifies the syntax and semantics of **6.2.2295** TO as defined in the CORE EXT word set.

See: 3.4 *The Forth text interpreter*

Ambiguous conditions:
A local argument, variable or buffer is executed while in interpretation state.

G.7 Reference implementation

(currently untested)

```
: TOKEN         \ -- caddr u
\ Get the next space delimited token from the input stream.
  BL PARSE
;

: LTERM?        \ caddr u -- flag
\ Return true if the string caddr/u is "--" or "}"
  2DUP S" --" COMPARE 0= >R
  S" }" COMPARE 0=  R> OR
;

: LBSIZE        \ -- +n
\ Parse up to the terminating ']' and EVALUATE the expression
\ not including the terminating ']'.
  [CHAR] ] PARSE EVALUATE
```

```
;

: LB?           \ caddr u -- flag
\ Return true if the last character of the string is '['.
  + 1 CHARS - C@ [CHAR] [ =
;

: LSEP?         \ caddr u -- flag
\ Return true if the string caddr/u is the separator between
\ local arguments and local variables or buffers.
  2DUP S" |" COMPARE 0= >R
  S" \" COMPARE 0=  R> OR
;

: {             ( -- )
  0 >R                                  \ indicate arguments
  BEGIN
    TOKEN 2DUP LTERM? 0=
  WHILE                                 \ -- caddr len
    2DUP LSEP? IF                       \ if '|'
      R> DROP  1 >R                     \ change to vars and buffers
    ELSE
      R@ 0= IF                          \ argument?
        CELL 1
      ELSE                              \ variable or buffer
        LB?
        IF  LBSIZE 3  ELSE  CELL 2  THEN
      THEN
      BUILDLV
    THEN
  REPEAT
  BEGIN
    S" }" COMPARE
   WHILE
    TOKEN
  REPEAT
  0 0 0 0 BUILDLV
  R> DROP
; IMMEDIATE
```

H Represent

H.1 Author

Ed (edsa@alphalink.com.au)

H.2 Change History

2005-10-09 Clarify number of characters at $c - addr$ when $flag_2$ is $false$.
`REPRESENT` code cleaned-up.
Addendum added.

2005-09-02 First release

H.3 Background

Forth-94 has few floating point display functions [none in the Floating-Point wordset, three in the Extension wordset]. To allow users to create their own output functions the Standard provides the float-to-string primitive `REPRESENT`.

Though intended to be portable, `REPRESENT`'s loose definition has given rise to implementations that differ in critical respects. It is also lacks adequate functionality to implement fixed-point notation display.

These deficiencies create significant obstacles for programmers attempting to write portable applications based on `REPRESENT`. While "work-arounds" may be used to minimize the problems, they tend to be complex and cumbersome (ref. 2).

It is perhaps inevitable that `REPRESENT` will need revision in the future. To that end, I wish to propose the following redefinition. It addresses all the key issues while remaining compatible with the Forth-94 specification. As the changes largely reflect common practice, compatibility with existing applications is unlikely to be affected.

H.4 Proposal

Replace the existing definition of `REPRESENT` with the following:

12.6.1.2143 `REPRESENT` FLOATING

($c - addr$ u -- n $flag_1$ $flag_2$) (F: r --) or (r $c - addr$ u -- n $flag_1$ $flag_2$)

At $c - addr$, place the character-string external representation of the significand of the floating-point number r. Return the decimal-base exponent as n, the sign as $flag_1$ and "valid result" as $flag_2$.

If u is greater than zero the character string shall consist of the u most significant digits of the significand represented as a decimal fraction with the implied decimal point to the left of the first digit, and the first digit zero only if all digits are zero. The significand is rounded to u digits.

If u is zero the string shall consist of one digit representing the fractional significand r rounded to a whole number, either one or zero, with an implied decimal point to the left of the digit.

Rounding follows the "round to nearest" rule; n is adjusted, if necessary, to correspond to the rounded magnitude of the significand. If r is zero or evaluates to zero after rounding, then n is 1 and the sign is implementation-defined.

If $flag_2$ is *true* then r was in the implementation-defined range of floating-point numbers. If $flag_1$ is *true* then r is negative.

An ambiguous condition exists if the value of `BASE` is not decimal ten.

When $flag_2$ is $false$, n and $flag_1$ are implementation-defined, as are the contents of $c-addr$. The string at $c - addr$ shall consist of graphic characters left-justified with any unused positions to the right filled with space characters. The number of characters is one if u is zero, or u characters otherwise.

See: 3.2.1.2 *Digit Conversion*, **6.1.0750** `BASE`, **12.3.2** *Floating-point operations.*

H.5 Rationale

1. Floating point zero issues

   (a) Exponent value Forth-94 does not explicitly indicate the exponent value (n) that should be returned in the case of floating-point zero.

   Mathematically any value would do since zero raised to any power remains zero. This has led to portability issues with various implementations returning $n = 0$, 1 or other value.

   From a number display perspective however, the exponent value is important since it determines where the decimal point shall be shown.

   The new specification requires `REPRESENT` to return $n = 1$. This corresponds to the common convention that zero has an implied decimal point to the right of the digit i.e. "0."

   [C library function `ecvt` similarly returns 1 for the exponent value — presumably for the same reasons.]

   (b) Negative zero

   Certain floating point representations such as IEEE-754 allow "negative zero". While Forth-94 is silent regarding "negative zero" there appears to be nothing that would prevent its use. See Implementation

   Note: Forths that choose to implement negative zero should do so consistently. Not only must `REPRESENT` return the appropriate sign, so also should `>FLOAT FROUND F. FS. FE.`

2. Out of range numbers

   Forth-94 states that when $flag_2 = false$, the number is not within the defined range and a user-defined string shall be returned in the buffer.

   (a) String contents

   "the string at $c - addr$ shall consist of graphic characters"

   Nothing is stated regarding the string's length, alignment or padding. Most forths adopt the following practice

   - The length of the returned string is the same as when $flag_2 = true$ i.e. the length is determined by u.
   - Characters are positioned left-justified within the buffer with any unused positions to the right filled with space characters. A users may subsequently trim off the padding spaces with `-TRAILING` e.g.

     ```
     ( r ) PAD u REPRESENT ...  PAD u -TRAILING ( c-addr u2 )
     ```

   (b) Value of n and $flag_1$

   "n and $flag_1$ are implementation defined"

Some forths have `REPRESENT` return a basic string ('NAN', 'INF' etc) and then use $flag_1$ to later indicate the sign. While such practices are permitted under Forth-94, they are not portable.

Portable applicationa can only make use of the returned string. If a sign needs to be passed to an application then it must be included in the string e.g. '+NAN' '-INF'

(c) String length

A difficulty with the present system is that the returned string becomes increasingly truncated as u decreases. At one or two characters most strings are unintelligible.

The situation could be alleviated somewhat by requiring `REPRESENT` to return a minimum string of [say] five characters. This would give a programmer greater scope in handling "not a number" situations since a usable string would always be available e.g.

```
( r ) PAD u REPRESENT 0= IF PAD u 5 MAX ...  THEN
```

Such a requirement could, however, break existing code and therefore has NOT been included in the new specification. It is mentioned here only for purposes of opening up discussion on the topic.

3. Rounding

Applications often require floating point numbers to be displayed rounded to a lesser precision than the internal maximum allows.

In Forth-94 this rounding is performed through `REPRESENT` with parameter u controlling the amount of rounding e.g.

```
( r ) PAD u REPRESENT 2DROP PAD u TYPE SPACE .
```

r	u	string	n (exponent)
0.6489	4	'6489'	0
0.6489	3	'649'	0
0.6489	2	'65'	0
0.6489	1	'6'	0

But what if we need to display 0.6489 rounded to 0 decimal places? This would require fraction .6489 be rounded to the nearest whole number i.e. 1.

Such situations arise when displaying fixed-point notation to a given number of decimal places. Failure to round the entire significand when appropriate leads to incorrect results. Here are some examples.

Display 0.009 to 2 decimal places in a field width of 5 characters:

```
0.009E  2 5 F.R       0.00 ok  ( FPOUT 1.6 and prior )
0.009E  5 2 0 F.RDP   0.00 ok  ( Gforth 0.6.2 )
```

(The result should have been 0.01)

`REPRESENT` currently has no provision for such rounding and it is a serious omission.

Luckily we can add the missing functionality by simply allowing u to take the value zero e.g.

```
( r ) PAD u REPRESENT 2DROP PAD u 1 MAX TYPE SPACE .
```

r	u	string	n (exponent)
0.6489	4	'6489'	0
0.6489	3	'649'	0
0.6489	2	'65'	0
0.6489	1	'6'	0
0.6489	0	'1'	1

As the above table shows we are merely extending `REPRESENT` in a logical direction. This makes it easy to implement and use.

H.6 Usage

Existing applications using REPRESENT may be left alone and they will continue to function as before.

Applications that will benefit from REPRESENT's new rounding facility are those in which parameter u takes the value zero.

Previously such applications needed 1 MAX inserted before REPRESENT to mask the fact that it could not handle $u = 0$ e.g.

```
c-addr u ...  1 MAX REPRESENT ...  c-addr u ...
```

Making these applications work correctly will require the following code re-arrangement in addition to the replacement REPRESENT.

```
c-addr u ...  REPRESENT ...  c-addr u 1 MAX ...
```

By way of demonstration, let's apply these principles to our earlier Gforth example.

Step 1 Load a version of REPRESENT that is compliant with the new specification (the sample one given below will do).

Step 2 Edit the source for F.RDP (located in Gforth 0.6.2 distribution file "stuff.fs"). Change line 211 from "1 max ur min" to "0 max ur min" and then re-load.

Step 3 Trying our previous example:

```
0.009E  5 2 0 F.RDP   0.01 ok
```

It now functions correctly.

H.7 Implementation

Implementing the new specification should not be difficult. A sample REPRESENT which includes all the critical features is given below.

Tips:

- Forth implementations that have "negative zero", and wish to display it, should ensure REPRESENT returns the appropriate sign even when rounding produces a result of zero e.g.

```
-0.4E PAD 0 REPRESENT PAD 1 TYPE  DROP SPACE .   0 -1 ok
 0.4E PAD 0 REPRESENT PAD 1 TYPE  DROP SPACE .   0  0 ok
```

- "Forths written in C" may be able to implement REPRESENT using library functions fcvt and ecvt.

A sample REPRESENT:

```
\ Assumes flag2 is always true and MPREC digits can be held
\ as a double number. "Negative zero" is not implemented.

 7 VALUE MPREC  \ your maximum precision
 2VARIABLE EXP  \ exponent & sign
```

```
: REPRESENT  ( c-addr u -- n flag1 flag2 ) ( F: r -- )
  2DUP [CHAR] 0 FILL
  MPREC MIN  2>R
  FDUP F0<  0 EXP 2!
  FABS  FDUP F0= 0=
  BEGIN  WHILE
    FDUP 1.0E F< 0= IF
      10.0E F/
      1
    ELSE
      FDUP 0.1E F< IF
        10.0E F*
        -1
      ELSE
        0
      THEN
    THEN
    DUP EXP +!
  REPEAT
  1.0E  R@ 0 ?DO 10.0E F* LOOP  F*
  FROUND F>D
  2DUP <# #S #>  DUP R@ - EXP +!
  2R>  ROT MIN 1 MAX CMOVE
  D0=  EXP 2@ SWAP  ROT IF 2DROP 1 0 THEN  \ 0.0E fix-up
  TRUE ;
```

H.8 Summary

A portable and functional REPRESENT reduces the burden on the programmer. It eliminates the need for work-arounds, simplifies application code and hides system-specific detail such as negative zero and rounding method.

As author of the FPOUT package, it is perhaps fitting to finish off with a demonstration of what the code would have been had the REPRESENT proposed here been available. In the following the first version shows original subroutine (F1) with work-arounds to make it portable; while the second version shows (F1) as it would be using the proposed REPRESENT.

1. Original subroutine (F1) with portability "work-arounds"

```
0 VALUE NZ#  ( initialized to true if REPRESENT responds
               to "negative zero"; or false otherwise )

\ float to ascii
: (F1)  ( F: r -- ) ( places -- c-addr u flag )
  TO PL#  PRECISION TO BS#
  FDUP FBUF BS# REPRESENT SWAP ( r exp flag2 sgn )
  \ save sign for negative zero systems
  [ NZ# ] [IF]  TO NZ#  [ELSE]  DROP  [THEN]
  FBUF C@ [CHAR] 0 = IF ( r=0 )
    >R  DROP FDROP  1 NZ#  R> ( exp sgn flag2 )
  ELSE
    AND ( exp & flag2 )  PL# 0< IF
      DROP PRECISION
```

```
  ELSE
    EF# 0> IF  1- (F0) DROP 1+  THEN  PL# +
  THEN
  DUP ( size ) 0= >R  1 MAX  PRECISION MIN  TO BS#
  FBUF R@ IF  PRECISION  ELSE  BS#  THEN  REPRESENT
  DUP  R> AND IF ( flag2 & size=0 )
    >R  FBUF C@  DUP [CHAR] 5 =
    FBUF PRECISION  1 /STRING (T0) NIP 0=  AND
    SWAP [CHAR] 5 <  OR
    IF    2DROP  1 NZ#  [CHAR] 0
    ELSE  SWAP 1+ SWAP  [CHAR] 1
    THEN  FBUF C!  R>
  THEN
THEN
>R  TO SN#  1- TO EX#  FBUF BS#  -TRAILING  R> <# ;
```

2. Replacement subroutine (F1) using the new `REPRESENT`

```
\ float to ascii
: (F1)  ( F: r -- ) ( places -- c-addr u flag )
  TO PL#  FDUP FBUF PRECISION REPRESENT NIP AND
  PL# 0< IF
    DROP PRECISION
  ELSE
    EF# 0> IF  1- (F0) DROP 1+  THEN  PL# +
  THEN  0 MAX  PRECISION MIN  TO BS#
  FBUF BS# REPRESENT >R  TO SN#  1- TO EX#
  FBUF BS#  1 MAX  -TRAILING  R> <# ;
```

H.9 References

1. ANS Forth-94 Standard
2. FPOUT — a floating point output package[8]

H.10 Addendum

1. If only one float-to-string primitive is needed then why do certain C language libraries provide several?

   A good question and one that should be asked of the library designers! As applications such as `FPOUT` demonstrate it is not only easy to reproduce the functionality of `ecvt fcvt gcvt` etc with but a single primitive, it is more efficient to do so.

2. Rather than change `REPRESENT` isn't it better to leave it alone and introduce another function like `fcvt` which has the necessary rounding required for fixed-point notation?

   This is a restatement of the previous question in a different form. Reflection will reveal that in order to implement `fcvt` it would require, at minimum, all the functionality of the proposed `REPRESENT`. In other words, `fcvt` is not a primitive at all but rather an application built upon an underlying primitive.

[8] ftp.taygeta.com/pub/Forth/Applications/ANS/fpout18.f

3. "$c-addr\ u$" is a buffer address and length. The proposal has $u = 0$ write a character beyond the end of the buffer.

   The premise that "$c - addr\ u$" represents a buffer and length into which `REPRESENT` writes is incorrect. ANS defines $c-addr$ as the buffer address where the character string is placed, and u as the number of "most significant digits" that shall be represented by the string. It was always the programmer's responsibility to ensure that adequate space was allocated to the buffer beforehand.

4. If u represents the number of significant digits then surely $u = 0$ means that no characters or a null string should result?

   This assumes that u is a length — which it is not. The usual result of rounding a number to n significant digits is another number which can be subsequently represented as an ASCII string. Before one can assert a null string should result, one would need to demonstrate there exists a number which it represents and that this number is the valid result of rounding to zero significant digits.

5. Most forths currently have `REPRESENT` return a null string when $u = 0$. Even though such behaviour may be technically in doubt, won't changing it affect existing applications?

   No. It should be remembered that it is the application and not `REPRESENT` that determines how many characters will be extracted from the buffer. So, if an application [mistakenly] chooses to extract u characters when $u = 0$ then the result will be the same irrespective of which `REPRESENT` was used. Having said that, I am unaware of any applications which actually use the null string.

6. If the ANS requirements at $u = 0$ are undefined then what makes you so certain the rounding behaviour you're advocating should be the adopted one?

   Put simply, it produces the right outcome in applications on every occasion without contrivance or work-arounds. The proposed rounding turns `REPRESENT` into a true universal primitive — one that can be used to build any float-to-string function.

I Structures

I.1 Author

Stephen Pelc

I.2 Change History

2006-08-30 Removed some restrictions on `FIELD:` and children.
Moved some words to FLOATING EXT.

2006-08-27 Added alignment to discussion.

2006-08-22 Rewrite after criticism

2006-08-21 First draft

I.3 Rationale

I.3.1 Problem

Virtually all serious Forth systems provide a means of defining data structures. The notation is different for nearly all of them.

I.3.2 Current practice

Current practice is too varied to merge or agree on one word set, either of names or of stack effects. In particular there are two camps, one which defines the name of the structure at the start, and one which defines it at the end. Examples are:

```
STRUCT POINT \ -- len
1 CELLS FIELD: P.X
1 CELLS FIELD: P.Y
END-STRUCT

STRUCT{
1 CELLS FIELD: P.X
1 CELLS FIELD: P.Y
}STRUCT POINT \ -- len
```

I.3.3 Discussion

The name first implementation of `STRUCT` and `END-STRUCT` is not difficult.

```
: STRUCT \ -- addr 0 ; -- size
  CREATE HERE 0 0 , DOES> @ ;

: END-STRUCT \ addr n --
  SWAP ! ; \ set size

: FIELD: \ n1 n2 "name" -- n3 ; addr -- addr+n1
  CREATE OVER , + DOES> @ + ;
```

The name last implementations of STRUCT{ and }STRUCT are trivial even for native code compilers:

```
: STRUCT{ \ -- 0
  0 ;

: }STRUCT \ size "name" --
  CONSTANT ;

: FIELD: \ n1 n2 "name" -- n3 ; addr -- addr+n1
  CREATE OVER , + DOES> @ + ;
```

In both cases the definition of FIELD: is the same. Further words that operate on types can all use FIELD: and so can be common to both camps.

For many modern compilers, implementing FIELD: to provide efficient execution requires carnal knowledge of the system.

Compatibility between the name first and name last camps is by defining the stack item *struct-sys*, which is implementation dependent and can be 0 or more cells. In the name-first example above, *struct-sys* corresponds to the "*addr*" stack items in the stack comments of STRUCT and END-STRUCT above. After some discussion on newsgroups and mailing lists, the consensus is that *struct-sys* should not appear in the specification of FIELD:.

Mitch Bradley wrote the following about structure alignment:

> In practice, structures are used for two different purposes with incompatible requirements:
>
> 1. For collecting related internal-use data into a convenient "package" that can be referred to by a single "handle". For this use, alignment is important, so that efficient native fetch and store instructions can be used.
> 2. For mapping external data structures like hardware register maps and protocol packets. For this use, automatic alignment is inappropriate, because the alignment of the external data structure often doesn't match the rules for a given processor.
>
> C caters to requirement (1) but essentially ignores (2). Yet people use C structures for external data structures all the time, leading to various custom macro sets, usage requirements, portability problems, bugs, etc.
>
> Since I do much more (2) than (1), I prefer a structure definition basis that is fundamentally unaligned. It is much easier to add alignment than to undo it.

I.3.4 Solution

The intention of this proposal is to find a portable notation for defining data structures using word names that do not conflict with existing systems, and to provide easy portability of code between systems.

System implementors can then define these words in terms of their existing structure definition words to enhance performance.

Authors of portable code will have a common point of reference.

The approach is to define FIELD: so as to enable both camps to co-exist. Since the name last approach is the simplest and only requires synonyms to define the start and end words, we simply recommend that these users write portable code in the form:

```
0
a FIELD: b
c FIELD: d
CONSTANT somestruct
```

We now turn to considering name first implementations, with the objective of providing a simple means of porting to name last implementations. Using the usual point and rectangle example, we can define structures:

```
BEGIN-STRUCTURE point \ -- a-addr 0 ; -- lenp
1 CELLS FIELD:  p.x   \ -- a-addr cell
1 CELLS FIELD:  p.y   \ -- a-addr cell*2
END-STRUCTURE

BEGIN-STRUCTURE rect   \ -- a-addr 0 ; -- lenr
point FIELD:    r.tlhc \ -- a-addr cell*2
point FIELD:    r.brhc \ -- a-addr cell*4
END-STRUCTURE
```

The proposal text below does not require `FIELD:` to align any item. This is deliberate, and allows the construction of groups of bytes. Because the current size of the structure is available on the top of the stack, words such as `ALIGNED` (**6.1.0706**) can be used.

Although this is a sufficient set, most systems provide facilities to define field defining words for standard data types. Note that these also satisfy the alignment requirements of the host system, whereas `FIELD:` does not.

The recommended field type definers are:

`CFIELD:`	1 character
`IFIELD:`	native integer (single cell)
`FFIELD:`	native float
`SFFIELD:`	32 bit float
`DFFIELD:`	64 bit float

The following cannot be done until the required addressing has been defined. The names should be considered reserved until then.

`BFIELD:`	1 byte (8 bit) field.
`WFIELD:`	16 bit
`LFIELD:`	32 bit field
`XFIELD:`	64 bit field

Name first minimalists are reminded that it is adequate to provide the source code (even by reference) in this proposal.

I.4 Proposal

10.6.2.aaaa `FIELD:` "field-colon" FACILITY EXT

(n_1 n_2 "⟨spaces⟩*name*" -- n_3)

Skip leading space delimiters. Parse name delimited by a space. Create a definition for name with the execution semantics defined below. Return $n_3 = n_1 + n_2$ where n_1 is the offset in the data structure before `FIELD:` executes, and n_2 is the size of the data to be added to the data structure. n_1 and n_2 are in address units.

***name* Exection:**
(*addr* -- *addr* + n_1)

Add n_1 from the execution of `FIELD:` above to *addr*.

10.6.2.cccc `BEGIN-STRUCTURE` "struct" FACILITY EXT

("⟨spaces⟩*name*" -- *struct* − *sys* 0)

Skip leading space delimiters. Parse name delimited by a space. Create a definition for name with the execution semantics defined below. Return a *struct* − *sys* (zero or more implementation dependent items) that will be used by `END-STRUCTURE` (**10.6.2.dddd**) and an initial offset of 0.

***name* Execution:** (-- +*n*)

+*n* is the size in memory expressed in address units of the data structure.

Ambiguous conditions: If *name* is executed before the closing `END-STRUCTURE` has been executed.

10.6.2.dddd `END-STRUCTURE` "end-structure" FACILITY EXT

(*struct* − *sys* +*n* --)

Terminate definition of a structure started by `BEGIN-STRUCTURE` (**10.6.2.cccc**).

10.6.2.eeee `CFIELD:` "c-field-colon" FACILITY EXT

The semantics of `CFIELD:` are identical to the execution semantics of the phrase `1 CHARS FIELD:`

10.6.2.ffff `IFIELD:` "i-field-colon" FACILITY EXT

The semantics of `IFIELD:` are identical to the execution semantics of the phrase `ALIGNED 1 CELLS FIELD:`

12.6.2.gggg `FFIELD:` "f-field-colon" FLOATING EXT

The semantics of `FFIELD:` are identical to the execution semantics of the phrase `FALIGNED 1 FLOATS FIELD:`

12.6.2.hhhh `SFFIELD:` "s-f-field-colon" FLOATING EXT

The semantics of `SFFIELD:` are identical to the execution semantics of the phrase `SFALIGNED 1 SFLOATS FIELD:`

12.6.2.iiii `DFFIELD:` "d-f-field-colon" FLOATING EXT

The semantics of `DFFIELD:` are identical to the execution semantics of the phrase `DFALIGNED 1 DFLOATS FIELD:`

I.5 Labelling

- `ENVIRONMENT?` impact — table 3.5 in Basis1
  name stack condition
- `THROW`/ior impact — table 9.2 in Basis1
  value text

I.6 Reference Implementation

Modified three times from original code in VFX Forth for Windows.

```
: begin-structure \ -- addr 0 ; -- size
\ *G Begin definition of a new structure. Use in the form
\ ** *\fo{BEGIN-STRUCTURE <name>}. At run time *\fo{<name>}
\ ** returns the size of the structure.
create
here 0 0 , \ mark stack, lay dummy
does> @ ; \ -- rec-len

: end-structure \ addr n --
\ *G Terminate definition of a structure.
swap ! ; \ set len

: field: \ n <"name"> -- ; Exec: addr -- 'addr
\ *G Create a new field within a structure definition of size n
bytes.
create
over , +
does>
@ +
;

: cfield: \ n1 <"name"> -- n2 ; Exec: addr -- 'addr
\ *G Create a new field within a structure definition of size 1
CHARS.
1 chars field:
;

: ifield: \ n1 <"name"> -- n2 ; Exec: addr -- 'addr
\ *G Create a new field within a structure definition of size 1
CELLS.
\ ** The field is ALIGNED.
aligned 1 cells field:
;

: ffield: \ n1 <"name"> -- n2 ; Exec: addr -- 'addr
\ *G Create a new field within a structure definition of size 1
FLOATS.
\ ** The field is FALIGNED.
faligned 1 floats field:
;

: sffield: \ n1 <"name"> -- n2 ; Exec: addr -- 'addr
\ *G Create a new field within a structure definition of size 1
SFLOATS.
\ ** The field is SFALIGNED.
sfaligned 1 sfloats field:
;

: dffield: \ n1 <"name"> -- n2 ; Exec: addr -- 'addr
\ *G Create a new field within a structure definition of size 1
```

```
DFLOATS.
\ ** The field is DFALIGNED.
dfaligned 1 dfloats field:
;
```

I.7 Test Cases

TBD.

J Synonyms

J.1 Author

Stephen Pelc

J.2 Change History

2006-09-22 Enhanced current practice section.
Fixed some typos.

2006-08-21 First draft.

J.3 Rationale

J.3.1 Problem

Various words have been used to generate a new name for an existing word. This required when porting code and when generating application wordlists that contain a reference to an existing word, e.g. when providing limited access to Forth system kernel words.

Especially with native code compiling Forth systems and cross compilers, these words have not provided full access to the required behaviour. The behaviour may require carnal knowledge of the underlying system, which is one reason why `SYNONYM` should be standardised.

J.3.2 Current practice

The proposed form `SYNONYM` has been in use at MPE with cross compilers and VFX Forth since 1998. It is also implemented in Win32Forth and PFE.

Many people have suggested that we stay with words such as `AKA`, `ALIAS` or `ALIAS:`, usually of the form

' ⟨oldname⟩ `ALIAS` ⟨newname⟩

This has merit in terms of common practice, but will break code for several systems. Some systems, e.g. cross compilers, cannot generate enough information using the *xt* of a word alone. All surveyed systems can implement `SYNONYM`.

J.3.3 Solution

Although many people have objected to parsing words, parsing permits the host system the most flexibility in implementation and is thus the preferred solution.

The syntax is:

`SYNONYM` ⟨newname⟩ ⟨oldname⟩

where ⟨newname⟩ will behave identically to ⟨oldname⟩.

Note that ⟨newname⟩ may be the same as ⟨oldname⟩.

J.4 Proposal

10.6.2.xxxx `SYNONYM` FACILITY EXT

(“⟨spaces⟩*newname*” “⟨spaces⟩*oldname*” --)

For both strings skip leading space delimiters. Parse *newname* and *oldname* delimited by a space. The definition for *newname* with the semantics defined below. *newname* may be the same as `oldname`.

***newname* interpretation:** ($i * x$ -- $j * x$)

Perform the interpretation semantics of *oldname*.

***newname* compilation:** ($i * x$ -- $j * x$)

Perform the compilation semantics of *oldname*.

Ambiguous conditions:

- The word *newname* is parsed by `'` or `[']` or `POSTPONE`.
- *oldname* is not found.
- `IMMEDIATE` is used for a word defined by `SYNONYM`.

J.5 Labelling

TBD

J.6 Reference Implementation

The implementation of `SYNONYM` requires carnal knowledge of the host implementation, which is one reason why it should be standardised. The implementation below is imperfect and specific to VFX Forth.

```
: Synonym \ <"new-name"> <"curdef"> --
\ *G Create a new definition which redirects to an existing one.
  create immediate
    hide  ' ,  reveal
  does>
    @  state @ 0=  over immediate? or
    if  execute  else  compile,  then
;
```

J.7 Test Cases

TBD

K THROW codes and IORs

K.1 Author

Stephen Pelc

K.2 Change History

2006-08-21 First draft.

K.3 Rationale

K.3.1 Problem

Error codes returned by some words, e.g. ALLOCATE are not specified, and an application has no entitlement to use them as THROW codes. The leads to very clumsy code of the form:

```
ALLOCATE IF <lit> THROW ENDIF
```

or

```
: ?THROW \ ior throwcode --
  SWAP IF THROW ELSE DROP THEN ;
ALLOCATE <lit> ?THROW
```

However, we also see many instances of code such as

```
ALLOCATE THROW
```

which leads to silent aborts when a system issues -1 THROW (as it is currently entitled to) or incorrect error messages.

K.3.2 Current practice

As far as possible within historical and commercial constarints, MPE has attempted to make iors THROWable. The only downside has been some necessary conversion of operating system error codes to ANS or application error codes.

Some years ago, some people objected to making iors the same as THROW codes because of the documentation overhead. This proposal is made to sample opinion again, particularly among Forth system implementers.

K.3.3 Solution

All words which return an *ior* should have one value assigned in the THROW code table (Table 9.2 in **9.3.5**). This table reserves values $-1..-255$ for system-defined exceptions. Systems that ignore this proposal are unaffected if they already avoid these values, and systems that implement this proposal gain use of these new fixed *ior*s.

The only downside is that we have to define some new THROW codes.

K.4 Proposal

Extend the `THROW` code table (Table 9.2 in **9.3.5**) so that there is a separate `THROW` code for each word that returns an *ior*.

K.5 Labelling

- `ENVIRONMENT?` impact — table 3.5 in Basis1

name	stack	conditions

- `THROW`/ior impact — table 9.2 in Basis1

value	text
-1	CLOSE-FILE
-2	CREATE-FILE
-3	DELETE-FILE
-4	FILE-POSITION
-5	FILE-SIZE
-6	OPEN-FILE
-7	READ-FILE
-8	READ-LINE
-9	REPOSITION-FILE
-10	RESIZE-FILE
-11	WRITE-FILE
-12	WRITE-LINE
-13	FILE-STATUS
-14	FLUSH-FILE
-15	RENAME-FILE
-16	ALLOCATE
-17	FREE
-18	RESIZE

L Extended character wordset (for UTF-8 and alike)

L.1 Author

Bernd Paysan

L.2 Change History

2005-09-25 First draft.

L.3 Problem

ASCII is only appropriate for the English language. Most western languages however fit somewhat into the Forth frame, since a byte is sufficient to encode the few special characters in each (though not always the same encoding can be used; latin-1 is most widely used, though). For other languages, different char-sets have to be used, several of them variable-width. Most prominent representant is UTF-8. Let's call these extended characters XCHARs. Since ANS Forth specifies ASCII encoding, only ASCII-compatible encodings may be used.

L.4 Proposal

L.4.1 Datatypes:

xc is an extended character on the stack. It occupies one cell, and is a subset of unsigned cell. **Note:** UTF-8 can not store more that 31 bits; on 16 bit systems, only the UCS16 subset of the UTF-8 character set can be used.

xc-addr is the address of an extended character in memory. Alignment requirements are the same as *c-addr*. The memory representation of an extended character differs from the stack location, and depends on the encoding used. An extended character may use a variable number of address units in memory.

L.4.2 Common encodings:

Input and files commonly are either encoded iso-latin-1 or utf-8. The encoding depends on settings of the computer system such as the LANG environment variable on Unix. You can use the system consistently only when you don't change the encoding, or only use the ASCII subset.

L.4.3 Words:

`XC-SIZE` "??" XCHAR

(*xc* -- *u*)

Computes the memory size of the extended character *xc* in address units.

`XC@+` "??" XCHAR

(*xc-addr*$_1$ -- *xc-addr*$_2$ *xc*)

Fetchs the extended character *xc* at *xc-addr*$_1$. *xc-addr*$_2$ points to the first memory location after *xc*.

`XC!+` "??" XCHAR

(*xc xc-addr*$_1$ -- *xc-addr*$_2$)

Stores the extended character *xc* at *xc-addr*$_1$. *xc-addr*$_2$ points to the first memory location after *xc*.

`XCHAR+` "??" XCHAR

(*xc-addr*$_1$ -- *xc-addr*$_2$)

Adds the size of the extended character stored at *xc-addr*$_1$ to this address, giving *xc-addr*$_2$.

`XCHAR-` "??" XCHAR

(*xc-addr*$_1$ -- *xc-addr*$_2$)

Goes backward from *xc-addr*$_1$ until it finds an extended character so that the size of this extended character added to *xc-addr*$_2$ gives *xc-addr*$_1$. Note: `XCHAR-` isn't guaranteed to work for every possible encoding.

`X-SIZE` "??" XCHAR

(*xc-addr u* -- *n*)

n is the number of monospace ASCII characters that take the same space to display as the extended character string starting at *xc-addr*, using *u* address units.

`XKEY` "??" XCHAR

(-- *xc*)

Reads an extended character *xc* from the terminal.

`XEMIT` "??" XCHAR

(*xc* --)

Prints the extended character *xc* on the terminal.

`CHAR` "??" XCHAR

("⟨spaces⟩*name*" -- *xc*)

Skip leading space delimiters. Parse *name* delimited by a space. Put the value of its first extended character onto the stack.

`[CHAR]` "??" XCHAR

Interpretation: Interpretation semantics for this word are undefined.

Compilation: ("⟨spaces⟩*name*" --)

Skip leading space delimiters. Parse *name* delimited by a space. Append the run-time semantics given below to the current definition.

Run-time: (-- *xc*)

Place *xc*, the value of the first extended character of *name*, on the stack.

L.5 Reference implementation:

Unfortunately, both the Gforth and the bigFORTH implementation have several system-specific parts.

L.6 Experience:

Build into Gforth (development version) and recent versions of bigFORTH.
Open issues are file reading and writing (conversion on the fly or leave as it is?).