EuroForth 2023

A proposed standard Forth style enumeration word set, using recognisers

Abstract

The lack of an ENUM word in the Forth standard has been previously noted.
The paper describes a solution that is fully faithful to Forth styling.
The implementation uses recognisers. Some potentially useful extensions will be discussed.

N.J. Nelson B.Sc. C. Eng. M.I.E.T.
Micross Automation Systems
Unit 6, Ashburton Industrial Estate
Ross-on-Wye, Herefordshire
HR9 7BW UK
Tel. +44 1989 768080
Email njn@micross.co.uk

1. Introduction

An enumeration is a data type which consists of a set of named members. Each member acts like a constant integer. A variable or value of the enumeration type can hold only integers within the named set. Enumerations are very useful, firstly for the elimination of "magic" numbers in code, and secondly for alerting any attempt to store an invalid number into an enumerated variable.

ANSI Forth does not include an enumeration word set.

The VFX implementation of Forth includes two enumeration words, but these are intended primarily for the import of "C" header files.

2. The existing VFX implementation

The two VFX enumeration words have the following form:

ENUM <enumname> {
  <membername>[=<integer>], [// <comment>]
  ...
};

For example:

```
ENUM GSignalMatchType {
  G_SIGNAL_MATCH_ID        = 1,     // The signal id must be equal
  G_SIGNAL_MATCH_DETAIL    = 2,     // The signal detail must be equal
  ...
  G_SIGNAL_MATCH_UNBLOCKED = 32,    // Only unblocked signals may be matched
               };
```

There is also a similar type without an enumeration name:

ENUM{
  <membername>[=<integer>], [// <comment>]
  ...
};

These enable code to be copied directly from a "C" header file and pasted into Forth code. They work reasonably well, though a certain amount of adjustment if often required. For example, in the original header file, the above example looked like:

```
/**
 * GSignalMatchType:
 * @G_SIGNAL_MATCH_ID: The signal id must be equal.
 * @G_SIGNAL_MATCH_DETAIL: The signal detail must be equal.
 ...
 * @G_SIGNAL_MATCH_UNBLOCKED: Only unblocked signals may be matched.

...

typedef enum
{
  G_SIGNAL_MATCH_ID        = 1 << 0,
  G_SIGNAL_MATCH_DETAIL    = 1 << 1,
  ...
  G_SIGNAL_MATCH_UNBLOCKED = 1 << 5
} GsignalMatchType;
```

But, even after adjustment, this does not look like Forth!
a) comma separators
b) equals sign assignment
c) right-to-left assignment
d) "C" style comments

This is not a problem when they are used as intended, for imported code.
But when creating one own enumerations, this is a distraction. We use enumerations extensively, and there are approximately 60 different types in the code of our main application.

When an enumeration name is supplied, this cannot be used in subsequent code. The name is placed in a dedicated dictionary. Its only purpose is so that all the enumerations can be listed, using the word .NAMEDENUMS.

3. A proposed enumeration in the Forth style

We aimed for a solution that is as concise and easy to type as possible:

ENUM<< <enumname>
  [<Forth expression>] <membername> [\ <comment>]
  ...
>>

The previous example then becomes:

```
ENUM<< GSignalMatchType
  1 0 LSHIFT G_SIGNAL_MATCH_ID           \ The signal id must be equal
  1 1 LSHIFT G_SIGNAL_MATCH_DETAIL       \ The signal detail must be equal
  ...
  1 5 LSHIFT G_SIGNAL_MATCH_UNBLOCKED    \ Only unblocked signals may be matched
 >>
```

Now we look more Forth like:

a) white space separators
b) no equals sign
c) left-to-right assignment
d) Forth style comments
e) any Forth expression may be used, thus the original intention can be made clear

4. Implementation idea

It will be seen in the example that all Forth words between enumerator name and the terminator >> need to function exactly as normal, in interpretation mode. Only the new member names need to be dealt with, by creating new words that act like constants.

In effect, we need to "recognise", and process, only the new words.

This led to the idea of creating a new recogniser, which acts in effect as an "unrecogniser".

5. Understanding recognisers

It is not clear how recognisers suddenly turned into recognizers, but it seems we have to put up with that.

The first description of recognisers at a Euroforth conference was by Bernd Paysan in 2012. They became the standard technique for text interpretation in VFX Forth as from version 5.1

The first thing to notice is which recognizers are normally at work. In our case, using the SSE64 floating point package, these are:

```
.recognizers REC-FIND  REC-VOCDOT  REC-NUM  REC-SSEFLOATS   ok
```

The text interpreter offers a parsed word to each of the recognisers in turn, until one of them accepts it. Any word that is not accepted by any of the recognisers throws an undefined word error.

So, in the list above, a word is offered first to REC-FIND which deals with predefined Forth words, then REC-VOCDOT which deals with words in a specified vocabulary, then REC-NUM which deals with single and double length integers, and finally REC-SSEFLOATS which deals with 64 bit floating point numbers.

Each recognisers has a set of three possible actions, one for interpretation, one for compilation and one for postponement.

The key feature of recognisers is that they can be dynamically attached and detached during compilation.

6. Starting and ending enumeration

What is needed therefore, is for ENUM<< and >> to attach and detach respectively, a recogniser that deals with the enumeration members.

```
: ENUM<< ( <name>--- ) \ Start an enumeration
  ?EXEC                                  \ Only when interpreting
  ['] REC-ENUM FORTH-RECOGNIZER +STACK-BOT \ Add enumeration to the recogniser stack
  0 -> ENUMVAL                           \ Initialise enumeration value
  PARSE-NAME ($CREATE)                   \ Name of the enumeration
  HERE -> ENUMLIST                       \ Set root address of list
  0 ,                                    \ Initialise list
  ['] ENUMCOMP, SET-COMPILER             \ When an enumeration is being compiled
  INTERP> ENUMINTERP                     \ Interpret action
;

: >> ( --- ) \ End an enumeration
  ['] REC-ENUM FORTH-RECOGNIZER -STACK     \ Remove from the recogniser stack
;
```

Looking at the above line by line:

a) We can clearly only define an enumeration while interpreting
b) In VFX, recognisers are handled using a standard stack mechanism
c) Our enumeration recogniser goes at the bottom of the stack, to be dealt with last
d) The value of enumeration members starts by default at zero
e) All enumerations should be named
f) We make provision for a list of the enumeration members
g) We make provision for special compiling and interpreting actions when using a child of ENUM<< i.e. an enumeration name.

7. The enumeration recogniser

```
: REC-ENUM ( ??,caddr,u---??,caddr,u,r:??? ) \ The enumeration recogniser
  DEPTH 2 3 WITHIN IF   \ Zero or one new enumeration values defined
    R:ENUM
  ELSE
    CR ." Error defining new enumeration value"
    R:FAIL
  THEN
;
```

Each recogniser receives the parsed word as a caddr,u pair. It returns the handle of a recogniser structure, preceded by any necessary parameters. This is an opportunity to check for the validity of the optional Forth expression (if any). The only restriction on this expression is that it must result in either zero or one item on the number stack.

8. The enumeration recogniser action

```
: ENUMINTERPACTION ( ??,caddr,u--- ) \ Interpreter action for enumeration recogniser
  DEPTH 3 = IF                       \ A new enumeration value defined
    ROT -> ENUMVAL                   \ Set it
  THEN
  ($CREATE)                          \ Create the enumerated name
  ENUMVAL ,                          \ Set the constant value
  INC ENUMVAL                        \ Next enumeration number
  LATEST-XT ENUMLIST ATEXECCHAIN     \ Add to list
  ['] ENUMVALCOMP, SET-COMPILER      \ When enumerated constant is being compiled
  INTERP> ENUMVALINTERP              \ When enumerated constant is being interpreted
;

' ENUMINTERPACTION  ' NOOP  ' NOOP  RECTYPE: R:ENUM   ( ---struct )
\ Contains the three recogniser actions for enumeration
```

Only one action is required in this case, for interpret.

Looking at the above line by line:

a) If an optional Forth expression exists, and it results in one item on the number stack, then that number becomes the value for the next enumerated member
b) The member is then created. and the value is set
c) The value is then incremented ready for the next member
d) The member is added to the list of members for that enumerations
e) We make provision for special compiling and interpreting actions when using a child of ENUMACTION i.e. an member name

The interpret and compilation actions are, for the time being, simply those of a constant i.e.

```
: ENUMVALCOMP, ( xt--- ) \ Compiling action of an enumeration value
  >BODY @ CLIT,
;

: ENUMVALINTERP ( addr--- ) \ Interpret action of an enumeration value
  @
;
```

9. Showing the list of enumeration members

Initially, I simply defined the interpret and compilation actions of an enumeration name as "do nothing" - they just return the address of the member list.

```
: ENUMCOMP, ( xt--- ) \ Compiling action of an enumeration
  >BODY CLIT,
;

: ENUMINTERP ( addr--- ) \ Interpret action of an enumeration
;
```

This means that the members of the enumeration can be easily shown, as in the example below:

```
ENUM<< TESTENUM         \ Name of the enumeration
            AZERO       \ By default, the enumeration starts at zero
            AONE        \ Standard Forth comments are allowed
  1 2 +     ATHREE      \ Any Forth expression can be used to set the enumeration
            AFOUR       \ The enumeration increments
>>                      \ Enumeration terminator

TESTENUM SHOWCHAIN
AFOUR
ATHREE
AONE
AZERO  ok
```

10. More useful ideas - to do

One possibility might be to make the enumeration name create a value that is only allowed to hold the member numbers, for example:

```
TESTENUM MYENUMVAL
1 -> MYENUMVAL ok
2 -> MYENUMVAL
Invalid member 2 for enumeration MYENUMVAL ok
```

The list could still be shown by adding a modifier, such as:

```
MEMBERSOF TESTENUM
AFOUR
ATHREE
AONE
AZERO  ok
```

Another modifier might check if a number is a member of the enumeration, e.g.

```
1 MEMBEROF TESTENUM . 1 ok
2 MEMBEROF TESTENUM . 0 ok
```

Alternatively, the enumeration name could specify the field of a STRUCT, e.g.

```
STRUCT MYSTRUCT
  VINT            my1
  VWORD           myword
  VBYTE           mybyte
  100  VFIELD     mystring
  TESTENUM        myenum
END-STRUCT
```

where the myenum field was only allowed to hold the members of TESTENUM.

11. Conclusion

The use of recognisers makes it easy to create a Forth friendly enumeration word set. The use of modifiers allows an enumeration name to carry out a variety of functions.