EuroForth 2025

Short paper

Improvements to enumeration

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

At Euroforth 2023 I proposed for standardisation a new enumeration wordset:

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

For example, one could do:

```
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
```

This was well received by my colleagues. But it wasn't long before requests for extra features came along.

2. Add translated descriptions to the enumeration

A translated description of an enumerated value is a frequent requirement, and this was normally done in a separate word e.g. for the above example, it might have been:

```
: TESTENUMDESCR ( enval---z$ ) \ Returns translated phrase describing enval
  CASE
    AZERO   OF P" Zero"       ENDOF
    AONE    OF P" One"        ENDOF
    ATHREE  OF P" Three"      ENDOF
    AFOUR   OF P" Four"       ENDOF
    ^NULL
  ENDCASE
;
```

Clearly it would have been a lot easier to define the description phrase from within the enumeration, rather than in a separate word. So now we have:

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

But now there are two optional items before the member name. How can we possibly tell them apart, give that Forth has no data types? In particular, <Description> cannot consist of a phrase number, because
a) The phrase number could theoretically be quite a small number, well within the likely range of enumeration numbers.
b) During the build process, some enumerations are needed before we build the database access wordset, so that translatable phrases are not available at the point of definition of the enumeration.

This was a challenge, until we realised that when a zero terminated string is defined using Z" , you always get an address that is nowhere near HERE, which is where it always used to be. Strings are in fact always presented on a recently invented space called SYSPAD.

Since SYSPADSTART is typically a very large number e.g.

```
SYSPADSTART . 140734512400720  ok
```

this now gives us a way of distinguishing the two data types int and string, in all cases of int that are likely to be enumerated.

We could now get as far as:

```
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 which has a stack effect...
                              \ ...( ---n ) can be used to set the enumeration
                  AFOUR       \ The enumeration increments
  Z" Customer" 11 AN11        \ Description and enumval
  Z" Category"    A12         \ Just a description
                  A13         \ Neither
>>                            \ Enumeration terminator
```

Our enumeration recogniser now looks like this:

```
: ENUMINTERPACTION ( ??,caddr,u--- ) \ Interpreter action for enum recogniser
  ^NULL -> ENUMZ$                                \ Assume no description
  DEPTH 2 - 0 ?DO                                \ Deal with any preceding values
    ROT DUP SYSPADSTART DUP /SYSPAD + WITHIN IF \ It is an address within the
                                                 \ strings buffer area
      -> ENUMZ$                                  \ Use it as a description
    ELSE                                         \ Probably not a string
      -> ENUMVAL                                 \ Use it as a new enum value
    THEN
  LOOP
  ($CREATE)                                      \ Create the enumerated name
  ENUMVAL ,                                      \ Set the constant value
  0 ,                                            \ Reserve space for phrase number
  ENUMZ$ ZCOUNT Z$,                              \ Compile description string
  INC ENUMVAL                                    \ Next enumeration number
  LATEST-XT ENUMLIST ATEXECCHAIN                 \ Add to list
  ['] ENUMVALCOMP, SET-COMPILER                  \ When an enumerated constant is
                                                 \ being compiled
  INTERP> ENUMVALINTERP                          \ When an enumerated constant is
                                                 \ being interpreted
;
```

We can still only save the original description though, not the translatable phrase number, which is not yet available. We've just left a space for it.

You will see that we create a list of all members of each enumeration, and there is a similar list of all the enumerations too.

It was not clear at the time precisely how these lists could be used - but now they proved to be really useful.

Right at the end of the build process, by which time all enumerations have been defined, and the database is up and running, we can execute a word that loops through all the enumerations and their members. It extracts the original description and matches it to a phrase number, creating new translatable phrases as necessary. It then pops the phrase number into the previously reserved space.

We have previously defined two new **modifiers** (I do wish they were not called operators in VFX), which enable us to easily access the original text and the phrase number of any enumerated member.

```
OPERATOR: ENUMPHRASE    \ Returns the phrase number of an enumerator
  OP# ENUMPHRASE CONSTANT OPENUMPHRASE
OPERATOR: ENUMDESCR     \ Returns the address of the description
  OP# ENUMDESCR  CONSTANT OPENUMDESCR
```

3. Making enumerated values available in external database queries

The second request from my colleagues was that enumerated values should be available automatically in the database. Generally, our main application, in Forth, is supported by several "dashboard" apps. The Forth program controls the system and places reportable information into the database. The dashboards, which require no programming, just configuration, display live data. Part of the configuration is the provision of an SQL statement that the dashboard can use to extract the data it needs. Previously, the SQL statements were littered with "magic numbers" representing our enumerated values. Every time a change was made to an ENUM<< in the Forth code, the dashboard configurations had to be checked in case any magic numbers had changed.

The solution was to create, automatically, a "loadable function" in the database, for each enumeration member. Then, a function can be used instead of a magic number inside an SQL query, and the results always match. For example

```
REPEV_CHCUS . 92  ok
SQL| SELECT REPEV_CHCUS() |SQL>>
+---------------+
| REPEV_CHCUS() |
+---------------+
| 92            |
+---------------+ ok
```

We can now take a look at a simplified version of the word which does all this, right at the end of the build.

```
: SETENUMPHRASES \ Place phrases for enumerations and create DB function
  { | penumname[ 255 ] pelementname[ 255 ] pelementnum plementdescr[ 255 ]
      pphrase -- }
  ENUMSLIST @ BEGIN                                    \ Anchor of enumerations
  DUP WHILE                                            \ Another enumeration
    DUP CELL+ @                                        \ Get xt of enumeration
    DUP IP>NFA 1+ penumname[ ZMOVE                     \ Get name
    EXECUTE @ BEGIN                                    \ Get anchor of elements
    DUP WHILE                                          \ Another element
      DUP CELL+ @                                      \ Get xt of element
      DUP IP>NFA 1+ pelementname[ ZMOVE                \ Get name of element
      >BODY                                            \ To element data
      DUP @ -> pelementnum                             \ Get element number
      DUP 2 CELLS+ plementdescr[ ZMOVE                 \ Get description
      SQL| DROP FUNCTION IF EXISTS                     \ Discard old function
           | pelementname[ >SQL |
      |SQL
      plementdescr[ C@ IF                              \ Description is defined
        SQL| CREATE FUNCTION                           \ Create new function
             | pelementname[ >SQL | ()
             RETURNS INT
             DETERMINISTIC                             \ If replication used
             RETURN | pelementnum FQL-N+ |
        |SQL
        plementdescr[ FINDPHRASE -> pphrase            \ Get phrase number
        pphrase SWAP CELL+ !                           \ Set in element data
      ELSE                                             \ No element description
        DROP                                           \ Address of data
      THEN
      @                                                \ Get next element
    REPEAT DROP                                        \ Discard element chain
    @                                                  \ Get next enumeration
  REPEAT DROP                                          \ Discard chain
;
```

4. Conclusion

A lot of this would have been easier if data types were more easily available - see my next paper "Forth 2025".