Notation Matters

Stephen Pelc
MicroProcessor Engineering
133 Hill Lane
Southampton SO15 5AF
England
t: +44 (0)23 8631 441
f: +44 (0)23 8033 9691
e: sfp@mpeforth.com
w: www.mpeforth.com

Abstract

Forth is an interactive extensible language. A consequence of this is that we can use the interactive nature of Forth both at compile-time and at run-time to produce our own notations. Good notation is very important when an application has a lifetime measured in decades.

During Forth Tagung 2012 at Beukenhof, I presented an improptu talk with this title. Michael Kalus then kept pestering me to write it up for *Vierte Dimensione*, the German Forth magazine. In the best impromptu talk tradition, I had kept no notes. Michael was good enough to transcribe the talk. Then I rewrote Michael's text to include examples and more text. And then I rewrote the paper again. Here it is.

The notation we use in Forth evolves over time. If we ever agree on a notation for OOP it will be a miracle. But, in order to achieve that, we have to remember some things about Forth that have nothing to do with **DUP** or **VARIABLE** but have a great deal to do with the way we use Forth. What we are using, what we are told Forth is really, is an interactive extensible language. And that means that we can change the notation of what we do. Which is actually one of the most powerful features of this class of language.

Using Forth at run-time

When I write a webserver in Forth, I don't have to write any scripting language. The server-side scripting in MPE's webservers has a mixture of HTML and scripting commands. First of all it tells us that it is going to use Forth as its scripting language:

```
<% language=forthscript %>
```

The **<%** and **%>** enclose scripting commands. I can include bits of Forth inside the web page to produce the output that I want, for example:
.
```
<% VAL @ . %>
```

That's an example of a script in Forth. We all know how to use it, and it cost almost nothing, either to implement or to use, and text interpretation happens at runtime. When I want to define how to open a COM port, I can write a peace of code that goes

```
COM1  9600 BAUD ... 8N1
```

and that is Forth source code. Even end-users can understand the code. What is important is to define a notation to perform the job. In both the examples above we are using the interactive capabilities of Forth at run-time, not just at compile-time. Quite often when we do this, we are providing a set of commands that an end-user needs to understand. For example, the people who write the web pages that use ForthScript need to know how to use it. We have to design the notation. These days, even embedded systems have enough memory to be able to do this.

The Forth community often tends to have an everlasting argument about the minutiae of the implementation, whereas what matters to users is the notation. This is the stuff that is on the paper or on the screen. When we design code for a Forth solution, design the notation before we design the implementation.

If the notation is clean to read, it is clean to use and we know we can write applications in it. Now, I am extremely biased. I am a vendor of Forth systems, which means I like to be paid. And the good thing about that is that I like big applications, because they cost a lot, and they become about writing maintaining lots of code, and that means when I come back to read the code in six month time, or in six years time, I still have to be able to understand what it does.

Application lifetime

You all know that you write comments on every line, and you still don't understand the code. The code should speak for itself. If you say something you just know what you mean. However, none of us are that good yet. You just hope that the person you are speaking to understands you. The maintenance programmer is the person you should be talking to when you write code. You don't know who he/she is or how clever he/she is.

I deal with applications that are more than 25 years old. The cross compiler that MPE sells is based upon code that we first saw in 1981. Very few lines of the original code remain unchanged, but the current code is a direct descendent of the original 1981 code. We have customers with code dating even further back. These people are dealing with code that is now over 30 years old, that started life on a HP desktop computer when it was the sexiest thing in computing. And we still have to maintain and extend those code bases.

What matters is that the code survives, and survives not in terms of its object code but in terms of its source code. The object code has gone from a 68000 to an 8086 to an 80386 to an ARM to something else. Every ten years it changes CPU or operating system. By the standards of the process, the application appears as source code on disc or paper. The the best form of preservation is what the developer sees.

Morse code application

So when we come back to applications, for example our OOP package or to Carsten Strotman's jobs for Morse code, the question becomes how to define a notation for the package. I have chosen to look at entering Morse code into a Forth application. You could set up dots and dashes, where you can enter a dot/period, and a dash could be either an underline or a minus sign. Source tokens that contain just these letters must be Morse code characters. The characters are going to be printable character from the ASCII set. So we can say, for example

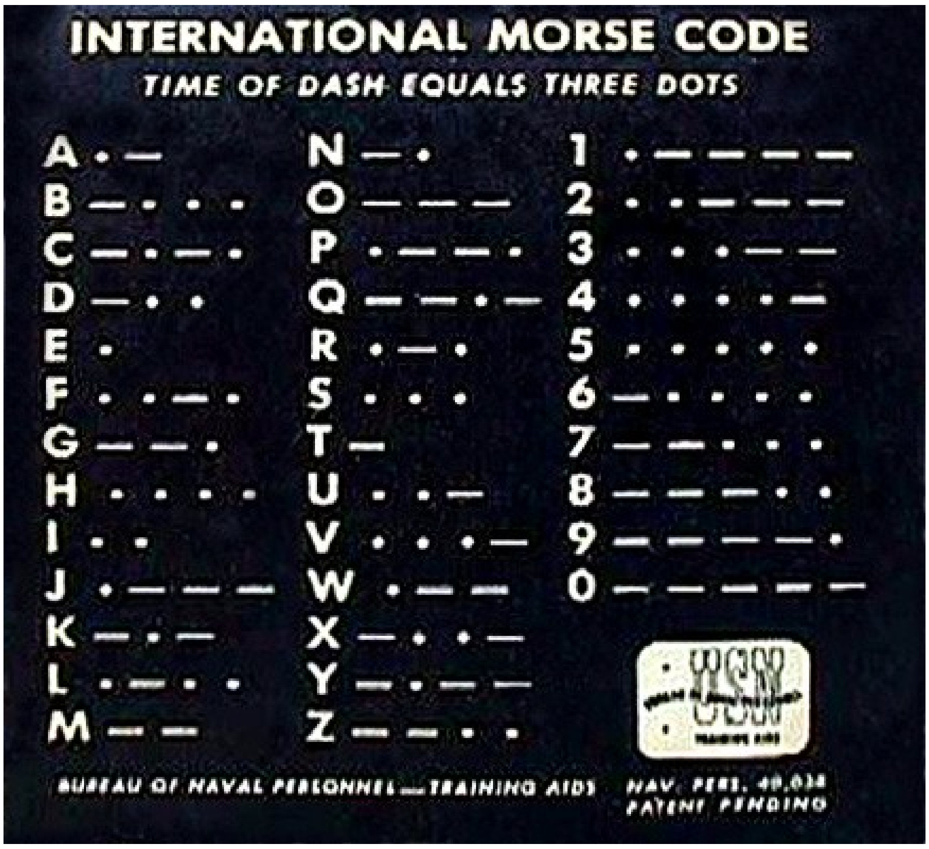
```
  morse-char A . -
  morse-char B - . . .
  morse-char C - . - .
```

or

```
  morse-char A .-
  morse-char B -...
  morse-char C -.-.
```

The second form avoids us having to know the end of the Morse sequence; it is just a space-delimited token containing dots and dashes.

We can translate this into some form that allows us to generate Morse code, either written in dots and dashes, or translated for transmission. The transmission side of the code will be responsible for knowing about inter-character and inter-word delays. For the moment, we are just interested in the notation, all the rest is implementation details. Our first job is to write the word **`MORSE-CHAR`** that generates one entry in our table. But that's not really enough, because the Morse code table has to be copied into the source code. And copying is really boring and really error-prone. What we want to do is to use existing text from a Morse code table. They typically look like these two.

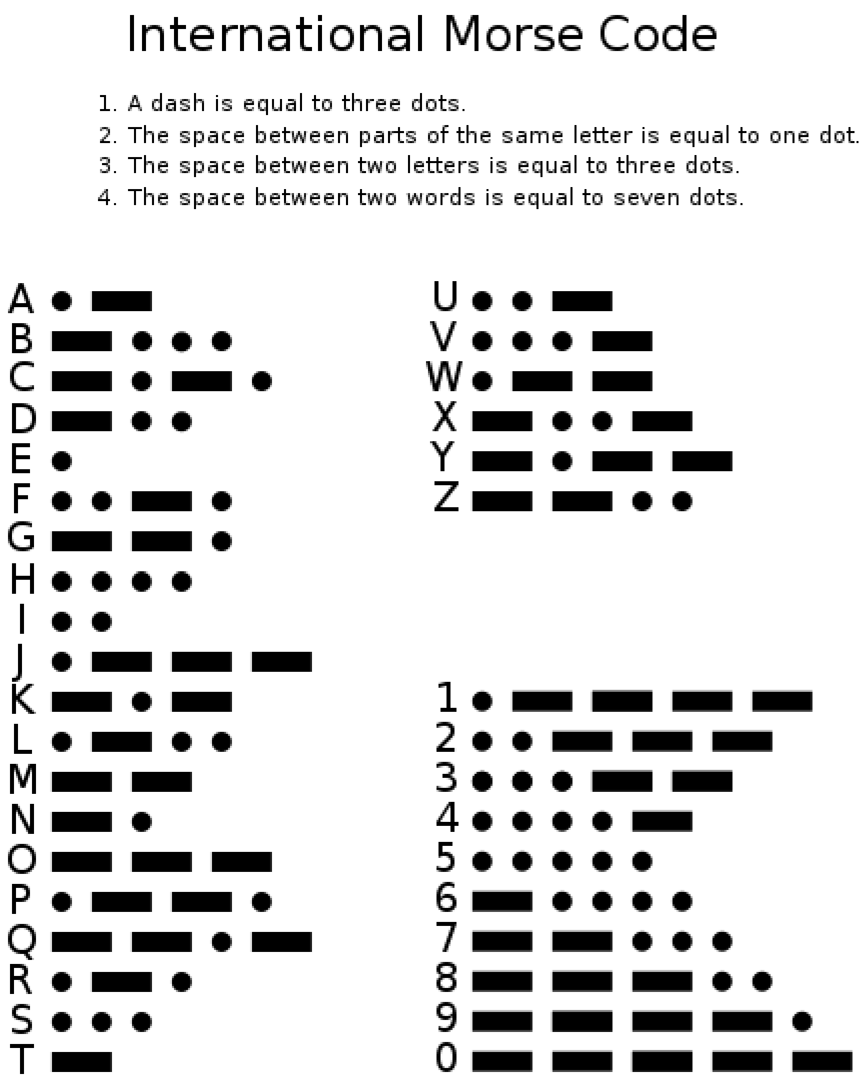


INTERNATIONAL MORSE CODE

TIME OF DASH EQUALS THREE DOTS

A	• —	N	— •	1	• — — — —
B	— • • •	O	— — —	2	• • — — —
C	— • — •	P	• — — •	3	• • • — —
D	— • •	Q	— — • —	4	• • • • —
E	•	R	• — •	5	• • • • •
F	• • — •	S	• • •	6	— • • • •
G	— — •	T	—	7	— — • • •
H	• • • •	U	• • —	8	— — — • •
I	• •	V	• • • —	9	— — — — •
J	• — — —	W	• — —	0	— — — — —
K	— • —	X	— • • —		
L	• — • •	Y	— • — —		
M	— —	Z	— — • •		

BUREAU OF NAVAL PERSONNEL — TRAINING AIDS NAV. PERS. 40,034 PATENT PENDING



International Morse Code

1. A dash is equal to three dots.
2. The space between parts of the same letter is equal to one dot.
3. The space between two letters is equal to three dots.
4. The space between two words is equal to seven dots.

A	● ▬	U	● ● ▬
B	▬ ● ● ●	V	● ● ● ▬
C	▬ ● ▬ ●	W	● ▬ ▬
D	▬ ● ●	X	▬ ● ● ▬
E	●	Y	▬ ● ▬ ▬
F	● ● ▬ ●	Z	▬ ▬ ● ●
G	▬ ▬ ●		
H	● ● ● ●		
I	● ●		
J	● ▬ ▬ ▬		
K	▬ ● ▬	1	● ▬ ▬ ▬ ▬
L	● ▬ ● ●	2	● ● ▬ ▬ ▬
M	▬ ▬	3	● ● ● ▬ ▬
N	▬ ●	4	● ● ● ● ▬
O	▬ ▬ ▬	5	● ● ● ● ●
P	● ▬ ▬ ●	6	▬ ● ● ● ●
Q	▬ ▬ ● ▬	7	▬ ▬ ● ● ●
R	● ▬ ●	8	▬ ▬ ▬ ● ●
S	● ● ●	9	▬ ▬ ▬ ▬ ●
T	▬	0	▬ ▬ ▬ ▬ ▬

Here is one in text form from
http://encyclopedia2.thefreedictionary.com/Morse+Code+%28table%29

```
International Morse Code

  A   .-        U   ..-
  B   -...      V   ...-
  C   -.-.      W   .--
  D   -..       X   -..-
  E   .         Y   -.--
                Z   --..
  F   ..-.
  G   --.       0   -----
  H   ....      1   .----
  I   ..        2   ..---
  J   .---      3   ...--
                4   ....-
  K   -.-       5   .....
  L   .-..      6   -....
  M   --        7   --...
  N   -.        8   ---..
  O   ---       9   ----.

  P   .--.      Period      .-.-.-
  Q   --.-      Comma       --..--
  R   .-.       ? Mark      ..--..
  S   ...       Hyphen      -....-
  T   -         Apostrophe  .----.
                Colon       ---...
  U   ..-       Quotation   .-..-.
  V   ...-      Slash       -..-.
  W   .--       @ sign      .--.-.
  X   -..-
  Y   -.--
  Z   --..
```

This is a good starting point for our Morse code table, so let us see what we have to change in the the table. We know that its going be a single character is followed by a sequence of dots and dashes. Looking at the table above, we see that we are looking for pairs of tokens. The first is the character and the second is its representation in dots and dashes. This is handled by the word **`MORSE-CHAR`** that we defined earlier. The effort of changing "Period" to "." is acceptable.

With a bit of syntactic sugar, we can generate a complete Morse code table that is easy to maintain. None of this is clever, it is all about achieving a result with the least effort, which in turn is just good engineering practice. The key decision is the decision to be lazy.

```
create MorseTable  \ -- addr
[morse
  A   .-       U   ..-
  B   -...     V   ...-
  C   -.-.     W   .--
  D   -..      X   -..-
  E   .        Y   -.--
               Z   --..
  F   ..-.
  G   --.      0   -----
  H   ....     1   .----
  I   ..       2   ..---
  J   .---     3   ...--
               4   ....-
  K   -.-      5   .....
  L   .-..     6   -....
  M   --       7   --...
  N   -.       8   ---..
  O   ---      9   ----.

  P   .--.     .   .-.-.-
  Q   --.-     ,   --..--
  R   .-.      ?   ..--..
  S   ...      -   -....-
  T   -        '   .----.
               :   ---...
  U   ..-      "   .-..-.
  V   ...-     /   -..-.
  W   .--      @   .--.-.
  X   -..-
  Y   -.--
  Z   --..
morse]
```

Using a good notation leads to reliability. Think about writing a 256 character font table. You are going to see an ASCII character followed by rows of bit on/off definitions. When you first see this data your eyes will glaze over at the potential boredom. If you think that you are going to translate this stuff to hexadecimal bytes without any errors, then think again. You will make mistakes. So the question is, should you spend a short time to write a notation for this and future font tables? Or should you spend a long time debugging?

Conclusion

What matters is notation. We are dealing with Forth which is a language in which we can manipulate our notation. We should remind ourselves to take advantage of this feature.

Acknowledgements

Elizabeth Rather never lets a notation go unchallenged.

Michael Kalus made me take this paper seriously.