# Filters for Unicode Markup, a work in progress report.

Bill Stoddart and Angel Robert Lynas
University of Teesside, UK

September 19, 2008

**Abstract**

The advent of Unicode extends the restrictive ASCII character set with millions of characters. However, we still have the same keyboards! Very often a document or program will require a limited set of Unicode characters in addition to characters available from the keyboard. We describe an approach which allows a user to provide a "markup" for each special character required, such as `\alpha` for the Greek letter $\alpha$, and in which the markup is replaced by the character it represents when typed at the keyboard or streamed from file. The techniques used include vectoring Forth's keyboard input to enable markup sequences to take effect during Forth command line input, and writing Unix filters in Forth. The latter allows the provision of wrappers that allow the use of Unicode markup with any Unix editor that can accept input from the standard input device.
**keywords. Forth, Unicode, Unix Filters, UTF-8, RVM-Forth**

## 1 Introduction

The Unicode Standard, available at `unicode.org`, provides a description of an extended character set encompassing mathematical symbols and the alphabets of most natural languages. The matter of character encodings is separated from the form of the characters themselves, and the encoding that has become dominant within Linux distributions is known as UTF-8, in which characters are represented within files or computer memory as a sequence of between one and four bytes. Support for Unicode is now becoming widely available, although it is still not included in some well known packages, for example Lesstif, the freeware version of Motif. Incorporation of a UTF-8 locale is non-trivial, as it breaks an implicit assumption under which almost all packages have been written, that each character occupies an equal amount of space. The Forth 93 Standard looked forward towards internationalisation, but assumed characters would remain of equal length but might become longer. Whilst such encodings are defined in Unicode, they have not proved to be the most popular ones, and anyone wishing to display a Unicode character on an Linux X Windows terminal or editing window will probably need to do it using the variable length encodings of UTF-8.

Unicode encoding issues were raised in 1998-9 by Stephen Pelc, Steve Coul and Peter Knaggs. An updated discussion from Stephen Pelc and Peter Knaggs appeared in 2001. An RFD on extended characters, designed to cope with all Unicode encodings, has been posted to the Forth Standards forum on forth200x.org by Bernd Paysan. A discussion paper "Xchars, or Unicode in Forth" by Anton Ertl and Bend Paysan appeared in EuroForth 2005 proceedings. Support for wide characters has been included in GForth and BigForth.

We are currently including UTF-8 support within our RVM-Forth (Reversible Virtual Machine Forth) environment, to gain access to mathematical symbols and other symbols commonly used in mathematics, such as letters from the Greek alphabet. We do this by providing a markup sequence for each special character required, e.g. `\alpha` for the Greek letter $\alpha$. When the markup sequence is entered from the keyboard, it is immediately replaced by the Unicode character it represents. We provide this facility at the Forth command line, and also make it available for use in editing via a Unix filter, also written in Forth.

## 2 Preparing RVM-Forth to act as a Unix Filter

RVM-Forth is a subroutine threaded Forth. It has a small nucleus generated by "meta compilation" of a description of Forth written in Forth. Metacompilation of the nucleus generates a Gnu Assembler file which is linked with object code generated from parts of the nucleus which are written in C. This provides an executable Forth Nucleus. When the Forth Nucleus is invoked, command line arguments are provided in the usual C style, and these are converted to a single string which is interpreted as though it where Forth terminal input. The usual startup sequence is:

```
RVM-FORTH HI
```

Where `HI` is a command that loads additional utilities, provided as Forth source code files, which are incrementally compiled. This process appears to the user to take virtually no time.

To use RVM-Forth to provide a UTF-8 markup filter when editing the file `example.r` with the Nano editor, we invoke it from a script file `NANO` as:

`NANO example.r`

Where `NANO` contains:

```
#! /usr/bash
RVM_FORTH UTILS ALSO UTF8 FILTER | nano -c -O -E -T 3 $1
stty echo icanon
```

Here the command `UTILS` loads the same utilities as `HI`, but does not print a sign on message. Forth is taking its input from the keyboard, and piping its output to Nano.

Unix, by default, uses buffered keyboard input in which a whole line is input and echoed to the terminal before the characters are seen by a program. RVM-Forth configures Unix keyboard input to be unbuffered, so that it receives one character at a time without that character being echoed to the screen. The second line of the file is included to restore normal Unix terminal handling when the filter terminates.

The commands `ALSO UTF-8 FILTER` add the Forth wordlist `UTF-8` to Forth's search order, then invokes the command `FILTER`, which is from this wordlist. `FILTER` records keyboard input in a circular "markup buffer", as well as generally passing it through to standard output. Whenever a defined markup sequence matches with the characters most recently received into the markup buffer, `FILTER` reinitialises the markup buffer, outputs characters to delete the markup sequence from the edit screen, and replaces it with the associated Unicode character.

The Forth side of the filter is non-terminating, and overall termination occurs when the user exits from the editor.

A second possible use of the same filter is as

`RVM_FORTH UTILS ALSO UTF8 FILTER < raw.r > cooked.r`

In this case Forth is receiving its input from file and must leave configuration of terminal input to the Unix filter mechanism. To achieve this Forth needs to test whether its input is coming from a keyboard during its configuration sequence, which it does by issuing a Unix system call via the C function:

```
int stdin_is_kbd() /*return -1 (Forth true flag) if stdin is the
keyboard, and 0 if stdin has been redirected to a pipe or file */
{ if ( system("[ -t 0 ]") == 0 ) return -1; else return 0 ;
}
```

The functionality of this command is made available as a Forth command by the following code[1]

```
CODE STDIN_IS_KBD ( -- f, returns true if stdin is the keyboard )
  xchg %esp,%esi
  call stdin_is_kbd
  push %eax
  xchg %esp,%esi
  ret
ENDCODE
```

A second issue is that, from our observations of Unix filter behaviour, it seems that `FILTER` will need to terminate when the end of the input file is reached. The different termination behaviours required of `FILTER` are provided by the loop test in the following definition.

```
: FILTER ( --, pre: stdin is the keyboard or a file.
Filter input, replacing markup sequences with the
corresponding UTF-8 character codes )
  INITIALISE-BUFFER
  BEGIN ?KEY STDIN_IS_KBD OR WHILE FILTER-KEY REPEAT BYE ;
```

When receiving from a file, `FILTER` will terminate when no further bytes are available, which occurs at he end of the file. At this point (and not before) `?KEY` returns a false flag.

Summarising, to enable RVM-Forth for Unix filter applications we have had to control the output of the RVM-Forth sign on message to ensure it does not occur during Filter deployment, and make standard input configuration dependant on whether the system finds it is receiving input from a keyboard or a file. When running RVM-Forth as a filter which pipes its output to an editor, the Forth filter need not explicitly terminate as it will be terminated externally when the editor terminates. When Forth runs as a filter which receives its input from a file, it must terminate when the end of the file is reached, and this will correspond to `?KEY` returning a false flag.

[1]We do not use the elegant postfix syntax of the classical Forth assemblers because our assembly code is passed through to the Gnu assembler after processing for control structures and code definitions, making it convenient to use the Gnu AT&T syntax for our assembler commands.

## 3 Recognising and acting upon markup sequences

The association between markup sequences and UTF-8 character encodings is recorded as a sequence of ordered pairs using the RVM-Forth Sets Package.

Part of the sequence which records Greek alphabetic characters, for example, is:

```
STRING INT PROD [ " \GAMMA " CE93 |-> ,  " \DELTA " CE94 |-> ,
" \THETA " CE98 |-> ,  " \LAMBDA " CE9B |-> , " \XI " CE9E |-> ,
" \PI " CEA0 |-> ,  " \SIGMA " CEA3 |-> ,  " \PHI " CEA6 |-> ,
.. ]
```

Here we are constructing a sequence of string integer pairs. The open square bracket is a start sequence bracket. Strings are enclosed by quotes, and `|->` is the maplet symbol, which removes two elements from the stack and constructs an ordered pair. The following comma compiles the ordered pair as the next sequence element.

"`\GAMMA` " is the markup string for a capital gamma ($\Gamma$) character, and `CE93` is the UTF-8 hexadecimal encoding for the character, which occupies two bytes. As each keyboard character is entered, it is added to the circular markup buffer. The text in the markup buffer is then compared with the markup sequences to see if any markup sequence matches the most recent text in the buffer. If it does, we re-initialise the markup buffer, backspace and erase the markup sequence on the screen, and output the UTF-8 character.

Some input keys require special treatment. A backspace results in the last character in the markup buffer being removed if the buffer is not empty. A new line will clear the markup buffer.

No attempt is made to handle escape sequences. These are recorded in the markup buffer and passed though to standard output like normal key strokes. This means, for example, that using the cursor movement keys during entry of a markup will prevent that markup being recognised. This can actually be useful, as it provides a way of entering a markup sequence without having it transformed into its corresponding Unicode character. It also means, however, that spurious markups can occur. For example, the left arrow key generates the hex byte sequence `1B 5B 44`. Since `5B 44` corresponds to `[D`, then if we include `[D` as a markup sequence, it will be spuriously recognised when a left arrow key is entered.

## 4 Experience with Unix editors

We have previously integrated Nedit into our Forth IDE. Typing `SEE <word>` at the Forth command prompt will cause Nedit to open the source file containing the definition of `<word>` in read only mode at the line where it is defined. `EDIT <word>` will open the file at the same place but in read/write mode. Nedit has the advantage of supporting a read only mode and of displaying line numbers. Unfortunately it is not suitable for UTF-8 encodings as it is based on Lesstif, which currently has no UTF-8 locale support.

Gedit handle files with extended characters, and some old versions of Gedit claim to have options for accepting input from STDIN. This option is not recorded on current documentation however, and we have not been able to make Gedit work with our UTF-8 filter. Another disadvantage is that there is no explicit read only mode. We have nevertheless implemented an option for Gedit to be the associated editor, and we use a Bash script to provide a read only wrapper for Gedit when a browsing mode is required.

Kate can also handle extended characters, and looked a promising candidate for use with a filter. In the Kate handbook,

(docs.kde.org/stable/en/kdesdk/kate/kate.pdf)

we read, under command line options:

```
kate --stdin
 Reads the document content from STDIN. This is similar to the
 common option - used in many command line programs, and allows
 you to pipe command output into Kate.
```

Whatever this might mean, it does not allow a filter configuration similar to the one used with Nano above. Standard output can be piped into Kate with commands such as:

```
 ls | kate --stdin
```

but we have not been able to use Kate with our UTF-8 filter in such a way that user input to Kate is passed through the markup filter.

Nano needs no special option to accept input from STDIN, and it works perfectly with the filter when the script NANO is invoked from the Unix Shell. However, when Nano is invoked from a system command sent by Forth, as when using `EDIT`, it sometimes fails to hit the correct line in the source file and also does not always record new lines correctly during the subsequent editing session.

## 5 Markup filtering of the Forth command line

The markup of utf-8 characters is achieved at the Forth command prompt by vectoring the execution of the Forth Standard `ACCEPT` to `UTF8-ACCEPT`. `ACCEPT` accepts the input of a given number of characters from the console into an input buffer, displaying characters as they are entered and terminating on receiving a new line or obtaining the given number of characters. `UTF8-ACCEPT` has the additional behaviour of storing received characters in the markup buffer and checking the markup buffer for a match against the possible markup sequences, such as `\alpha` . When a match is found, backspaces are output to delete the markup sequence from the screen, and the corresponding Unicode character is output in its place. Termination occurs on receipt of a new line character, or when the given maximum number of bytes have been received into the input buffer. Input of a backspace character causes the last character in the markup buffer to be removed, if it exists, and causes the last character of the input buffer to be removed. This latter may occupy between one and four bytes.

## 6 Adapting RVM-Forth to UTF-8 Unicode

Switching character representation to UTF-8 revokes the assumption under which RVM-Forth was written, that each character occupies one byte, and also the assumption, taken in the Forth Standard, that each character occupies an equal amount of memory space known as a "char location". With UTF-8 encoding a string of n characters may occupy more than n address units, leading to possible buffer overruns if we continue to rely on our old assumptions.

We use the abbreviation *pchar*, introduced Stephen Pelc and Peter Knaggs. A *pchar* refers to a "primitive character", from the ASCII character set, and requiring one char location of storage. In the UTF-8 encoding, other, "extended characters", occupy between 2 and 4 character locations.[2]

The current Standard description of `ACCEPT` has the signature:

```
( c-addr +n1 -- +n2 )
```

and a description that begins with: "Receive a string of at most +n1 characters..". This needs to be re-expressed as "Receive a string of length at most +n1 address units". There are further issues if markup is used to enter extended characters. After entering "`\alpha \beta` " we have the two char-

[2],This is not quite accurate, since, in some alphabets, characters may be written with diacritical marks which themselves are encoded as separate characters. However, we already have enough to deal with here..

acter string $\alpha\beta$, which of four address units in length. However we have to allow a buffer length of ten in order to enter the string. Where a *pchar* occupies one byte and UTF-8 encoding is used, the maximum character length is 4 bytes. If $m$ is the length of the longest markup sequence and $m > 4$, then to allow entry of $n$ characters it is sufficient to accept a string of length $4 * (n - 1) + m$ address units.

`ACCEPT` is typical of a number of Standard words which deal with characters and whose glossary definitions require rewording in terms of character locations rather than characters. A new wordset is required to deal with the special operations required of wide characters, and here the current RFD provides valuable suggestions.

## 7 Conclusions and future work

Using markup to access a limited range of special characters allows us to write Forth code which can makes use of classical mathematical symbols. This is particularly beneficial to RVM-Forth which has an extensive set package and supports $\lambda$ notation. Future work includes rewriting the mathematical parts of the RVM Forth source code using the extended character set where appropriate, and writing infix expression compilers for set expressions using the techniques outlined in our EuroForth 2008 paper "Using Forth in a Concept Oriented Computer Language Course." Further work is also envisaged to get the Forth IDE words `SEE` and `EDIT` working with more editors, and combining this functionality with that of the markup filter.