

# QL Assembly Language Mailing List

Issue 8

Norman Dunbar

PUBLISHED BY MEMYSELF EYE PUBLISHING ;-)

**Download from:**

[https://github.com/NormanDunbar/QLAssemblyLanguageMagazine/releases/tag/Issue\\_8](https://github.com/NormanDunbar/QLAssemblyLanguageMagazine/releases/tag/Issue_8)

**Licence:**

Licensed under the Creative Commons Attribution-NonCommercial 3.0 Unported License (the “License”). You may not use this file except in compliance with the License. You may obtain a copy of the License at <http://creativecommons.org/licenses/by-nc/3.0>. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

This pdf document was created on 7/10/2019 at 17:48.

Copyright ©2019 Norman Dunbar



# Contents

<b>1</b>	<b>Preface .....</b>	<b>7</b>
1.1	Feedback	7
1.2	Subscribing to The Mailing List	7
1.3	Contacting The Mailing List	8
<b>2</b>	<b>Feedback on Issue 7 .....</b>	<b>9</b>
2.1	Feedback from Wolfgang Lenerz	9
2.2	Feedback from Marcel Kilgus	9
2.3	More Feedback from Wolfgang Lenerz	11
2.4	Even More Feedback from Wolfgang Lenerz	13
2.5	A Better Ql2utf8	16
2.6	A Better Utf82ql	22
<b>3</b>	<b>Image Credits .....</b>	<b>23</b>







## Listings

2.1	Wolfgang's improved ql2utf8 Utility . . . . .	16
2.2	Wolfgang's improved utf82ql Utility . . . . .	22





# 1. Preface

## 1.1 Feedback

Please send all feedback to [assembly@qdosmsq.dunbar-it.co.uk](mailto:assembly@qdosmsq.dunbar-it.co.uk). You may also send articles to this address, however, please note that anything sent to this email address may be used in a future issue of the eMagazine. Please mark your email clearly if you do not wish this to happen.

This eMagazine is created in  $\text{\LaTeX}$  source format, aka plain text with a few formatting commands thrown in for good measure, so I can cope with almost any format you might want to send me. As long as I can get plain text out of it, I can convert it to a suitable source format with reasonable ease.

I use a Linux system to generate this eMagazine so I can read most, if not all, Word or MS Office documents, Quill, Plain text, email etc formats. Text87 might be a problem though!

## 1.2 Subscribing to The Mailing List

This eMagazine is available by subscribing to the mailing list. You do this by sending your favourite browser to <http://qdosmsq.dunbar-it.co.uk/maillinglist> and clicking on the link “Subscribe to our Newsletters”.

On the next screen, you are invited to enter your email address *twice*, and your name. If you wish to receive emails from the mailing list in HTML format then tick the box that offers you that option. Click the Subscribe button.

An email will be sent to you with a link that you must click on to confirm your subscription. Once done, that is all you need to do. The rest is up to me!

### 1.3 Contacting The Mailing List

I'm rather hoping that this mailing list will not be a one-way affair, like QL Today appeared to be. I'm very open to suggestions, opinions, articles etc from my readers, otherwise how do I know what I'm doing is right or wrong?

I suspect George will continue to keep me correct on matters where I get stuff completely wrong, as before, and I know George did ask if the list would be contactable, so I've set up an email address for the list, so that you can make comments etc as you wish. The email address is:

[assembly@qdosmsq.dunbar-it.co.uk](mailto:assembly@qdosmsq.dunbar-it.co.uk)

Any emails sent there will eventually find me. Please note, anything sent to that email address will be considered for publication, so I would appreciate your name at the very least if you intend to send something. If you do not wish your email to be considered for publication, please mark it clearly as such, thanks. I look forward to hearing from you all, from time to time.

If you do have an article to contribute, I'll happily accept it in almost any format - email, text, Word, Libre/Open Office odt, Quill, PC Quill, etc etc. Ideally, a  $\text{\LaTeX}$ source document is the best format, because I can simply include those directly, but I doubt I'll be getting many of those! But not to worry, if you have something, I'll hopefully manage to include it.





## 2. Feedback on Issue 7

Well now, here's a thing. Very quickly after Issue 7 "hit the streets" I got feedback from two different people. Thanks very much to Wolfgang and to Marcel for their input, and their permission to publish.

### 2.1 Feedback from Wolfgang Lenerz

[WL] Just a little comment: there is a typo on page 16, in the third code extract at line 1: Tobias makes a MOVEM to ...a2-a7 : it should be to ...a2-a6.

[ND] Thanks. I don't have a Q68 (yet?) and I really didn't have much to do with Tobias's article to get it into the eComic, so I didn't notice that slight error. I fixed it in the PDF download on 1st October 2019 at around 19:00 BST (UTC + 01:00) - so anyone who downloaded prior to that time might wish to download again to get the correction.

[WL] Also a more general comment, which I offer as constructive criticism: in the utf82q1 routine, when handling values over 127 (i.e. at least 2 bytes), why check for the special cases first (arrows, pound etc) before getting the values from the table? Wouldn't it be better to leave their place in the table at 0 as well, and every time you hit a 0 in the table you check for the exception?

[ND] Good point, thanks. That would have made more sense as the processing is more likely to be processing valid characters than the exceptions. I thought I was doing well getting the exceptions in what I thought was the most likely order!

[WL] Oh, and this probably doesn't get said often enough : *really enjoy reading your prose!*

[ND] Thanks. It's nice to get feedback, but much nicer to get compliments.

### 2.2 Feedback from Marcel Kilgus

[MK] As a pedantic ass I have to object so sentences like these:

*The UK Pound symbol is character 96 (\$60) on the QL, but in ASCII it is character 163 (\$A3)" (etc.)*

[ND] I like pedants! My wife says I am one, then she corrects me at every available opportunity!

[MK] ASCII is, by definition, 7-bit, so it cannot contain a character with the number 163. The tale of characters 128-255 is one fought in many battles. Linux tended to be "ISO 8859-1" and later "ISO 8859-15" before they adopted UTF-8, on Windows you will mostly find the "Windows-1252" encoding. These are very similar, but differ when it comes to the Euro sign for example (ISO 8859-1 is too old to have a Euro sign and the others have adopted it in different places).

[ND] Technically, I agree, ASCII is indeed 7 bit and 163 is definitely not 7 bit. But let's face it, there have been 8 bit "ASCII" characters for many years, even when I was at college back in the, ahem, early eighties, ASCII was (at least, considered) 8 bit - whether pedantically correct or not. However, true ASCII is 7 bit.

[ND] I remember many occasions, back when `config.txt` was still a thing, trying to set up the correct code page for a system. A nightmare as there was no Google back then to help out, just the manual for whatever system I was installing or working with.

[ND] I am led to understand, however, that ISO/IEC 4873 introduced some extra control codes "characters", in the \$80 to \$9F hexadecimal range, as part of extending the 7-bit ASCII encoding to become an 8-bit system.<sup>1</sup> However, I sit corrected on the 7/8 bit point. Thanks.

[MK] But, and that is the important thing, Unicode was made to unify them all. And UTF-8 is a pretty darn cool invention, unfortunately it came too late for Windows, which was a very early adopter of Unicode at a time when everybody thought "65536 characters ought to be enough for everyone!". So Windows started to use 16-bits for every character ("UCS-2" encoding), which makes coding somewhat weird, and then they found out that 65536 characters are not enough after all, so now Windows uses UTF-16, which is UTF-8's big brother, with sometimes 2 bytes per character and sometimes 4. What a mess. But when it comes to data storage UTF-8 is the way to go these days, always!

[ND] It sure is a mess, and yes, UTF-8 is the way to go. As I mentioned XML files depend on it, the web is pretty much full of it in all those HTML files etc. And, once you get your head around the difference between a "code point" and the character's actual bytes, it's pretty easy to understand.

[ND] I'm not so sure that Windows is missing out or behind the times though. At work, my files are all pretty much UTF-8 (I write my documents in `ASCIIDOC`<sup>2</sup> format and convert them to PDF files using `asciidoc-to-pdf` - if I need Office flavoured docs, I use `pandoc` to convert to something in `DOCX` format - but I almost never use those. `Asciidoctor` files are plain text, and very easy to version control! `Notepad++` or `VSCodium` are my text editors of choice and both save in UTF-8 with no problems. Even `Notepad` itself can read the files - and I suspect Windows 10 will be better, I'm on Windows 7. (Currently)

[ND] Mind you, those damned so-called "smart" quotes that Office documents insist on using mess things up truly. It's the first thing I turn off with my Office stuff, and every slight update or patch seems to turn them back on! So annoying.

[MK] For QPC I already implemented these translations 20 years ago when copying text to/from

<sup>1</sup>The Unicode Consortium (October 27, 2006). "Chapter 13: Special Areas and Format Characters" (PDF). In Allen, Julie D. (ed.). *The Unicode standard, Version 5.0*. Upper Saddle River, New Jersey, US: Addison-Wesley Professional. p. 314. ISBN 978-0-321-48091-0. Retrieved March 13, 2015.

<sup>2</sup>Now that's ironic!

the clipboard. But well done for bringing UTF-8 to the QL

[ND] Well, thanks for the reminder of how old I'm getting! The reason I did the utilities was simple, I had one of those itches to scratch. When I did a bit of work with Jan on his updated QL Monitor, I used a Linux system to do the typing - it's what I'm used to - and those arrow characters caused me no end of grief, as did the copyright and the pound signs. I messed about there using actual, ahem, ASCII codes (sorry!) but now, I don't have to.

[ND] Oh, and *thank you* for QPC2, it's my favourite QL program of all time, and it simply "just" works on Linux under Wine. I did have some problems recently with it not working, but I traced that to a mix and match installation with bits of Wine 3 and bits of Wine 4 living together in sin.

QPC2 is what has kept me in the QL scene for as long as I can remember - I always got somewhat tired of the QL, the cables, the hard drive, the noise, the length of table I needed with limited space in my flat (apartment) and so on. With QPC2 it's all on my laptop. Nice and compact.

[ND] And, by the way, *I am a pedant's baddest nightmare!*

## 2.3 More Feedback from Wolfgang Lenerz

[WL] I had a longer look at ql2utf8. I hope you don't mind a few more comments.

[ND] No, I like getting comments - most people do assembly better than I do!

[WL] When I tried to compile the source file, I couldn't, as the different traps weren't defined.

```
io_fbyte      equ 1
io_sbyte      equ 5
mt_frjob      equ 5
```

[ND] Were you using QMAC by any chance? I know that's a recurring problem for QMAC as GWASS and GWASL come with the various traps and vectors "automatically" included. If I include them in the source, then they won't assemble for me, I get an error about duplicate definitions.

[WL] Moreover, I got a few errors that some bra.s were out of reach.

[ND] Hmmm, I just recompiled with GWASS and got no errors at all. However, I did get one error with GWASL. Looking at the listing file, it's complaining that the label oneByte is an "illegal instruction" - weird. I remember GWASL doing that on a few occasions in the past. I used to edit the sources, rub out the label, and type it in again, that usually worked. I could never trace it to hidden characters etc as a hex dump of the source showed nothing out of the ordinary.

I don't however, get any errors about short branches being out of reach.

[WL] It seems to me that the two lines of code at label TestBit7 are superfluous: you are doing the exact same test just beneath it, at label Testpound.

[ND] That's a typing error. Originally I only had the BVS.S instruction rather than the BTST #7 so in theory, if D1 was loaded with a byte >= \$80 the V flag would be set. Unfortunately it didn't work. I traced the code under QMON2 and by the time we get to that point, the V flag is clear, always. I obviously forgot to remove the BVS when I edited the code to add in the BTST #7 instruction. My mistake.

[WL] You could replace the two instructions at Label OneByte with the single instruction PEA readLoop.

[ND] I see what you mean, if I do that replacement, then instead of branching to `writeByte` and returning and then branching off to `readLoop`, just drop in to `writeByte` and return automatically to the top of the loop. Nice!

[WL] Then you could also just delete the last two instructions of label `gotCopyright` (no need to `bsr.s writebyte`, you just fall through) and you could also replace, in the different `gotxxxx` routines (e.g. `gotEuro`, `gotGrave` etc) the two instructions:

```
bsr.s writeByte
bra readLoop
```

with a simple:

```
bra.s oneByte
```

and it might be able to use a `bra.s` rather than a `bra` somewhere in the changed code.

[ND] Yes, that all makes perfect sense given the above changes to `oneByte`.

[WL] At label `notArrows`, you might want to replace these 4 lines

```
addq.b #1,d2
move.b 0(a2,d2.w),d1 ; Second byte
bsr writeByte ; Send it out
bra readLoop ; Go around.
```

with these two

```
1 move.b 1(a2,d2.w),d1 ; Second byte
2 bra oneByte
```

[ND] Yes, that makes perfect sense too. Thanks.

[WL] I would set the output & input channel IDs into two registers (eg A4, A5) and move them into A0 when needed in the byte read/write subroutines, instead of accessing the stack (and thus memory) every time with a LEA.

[ND] I used A4 and A5 for that very purpose in the following chapter, in the `Utf82q1` code, and forgot to go back and fix this code to do the same.

[WL] Finally, I would also include test at label `notArrows` to make sure that the byte in D1 doesn't exceed the max value of your table. I know that values above that are not printable characters, but it is possible to include them in a text file. You might want to tell the user that some characters couldn't be translated...

[ND] Yes indeed, that was an oversight. Thanks for pointing it out.

[WL] Hope you don't mind the above.

[ND] Not at all, many thanks indeed.

So, given all those amendments, here for your delectation is the latest version of the `Q12utf8` code, incorporating all of Wolfgang's changes and corrections.



## 2.4 Even More Feedback from Wolfgang Lenerz

[WL] I also had a look through the `utf2q1` code. Some of the comments made for the other routine (`q12utf8` ND) may also apply here, no need to go through them again. Here I have some more comments on this routine.

[ND] Ok, I'm sitting comfortably ....

[WL] The first one is not really about the code itself, but the way you structured it. Of course, this is much of a personal preference, so please take this with a pinch (or even a spoonful) of salt. Leaving out the exception and read/write routines, your code is structured thus:

```
readLoop
    get byte
    leave if EOF

Multibytes
    is it two bytes?
    yes -> jump to handle_two

NotTwo
    is it three bytes?
    yes -> jump to handle_three

Error
    not three bytes , return error

handle_two
    treat two bytes
    bra    readloop

handle_three
    treat three bytes
    bra    readLoop
```

[WL] For me, you have 6 different blocks of code. I would prefer the following structure with 4 blocks (making the code less "spaghetti"):

```
readLoop
    get byte
    leave if EOF

handle_two
    is it two bytes?
    no-> jump to handle three
    treat two bytes
    bra    readloop

handle_three
    is it 3 bytes?
    no -> jump to error
    treat three bytes
    bra    readLoop

Error
```

```
not three bytes , return error
```

[ND] Yes, I admit that sometimes my structure leaves a lot to be desired and you are correct in what you say above - I must try harder!

[WL] Leaving out the branches to the loop, basically your way of doing it is:

```
is it something?
  yes, go off , handle it_1
is it something else?
  yes, go off and handle it_2
error
handle_it1
handle_it2
```

[WL] Whereas mine is:

```
is it something?
  no, go off to next check

handle it1
  is it something else?
    no, go to error

handle it2

error
```

[WL] Again, this is a personal preference: There is no functional difference, but I, personally, find the second one easier to read if you want to follow the flow of the code.

[ND] Agreed.

[WL] But in doing so it will allow you to write the code at the multiBytes label so:

```
multiBytes
  move.b    d1,d2
  andi.b    #%11100000,d2      ; <--- BUG HERE? [ND]
  cmp.b     #%11000000,d2      ; 2 bytes?
  bne.s     threebytes        ; ... no->

twoBytes
  (treat 2 bytes including exceptions)

testThree   (no need to copy d1 into d2 again)
  cmp.b     #%11100000,d2      ; 3 bytes?
  bne       invalidUTF8       ; ... no ->
  ...
```

[ND] Hmm, I think you have a bug there. For three byte characters the top nibble should be 1110, so your mask is missing a '1' bit. I suspect you intended to type the following for multiBytes:

```

multiBytes
    move.b    d1, d2
    andi.b    #%11110000, d2

```

Otherwise you are forcing bit 4 of D2 to always be a zero. However, that minor niggle aside, I like your version better than mine as I/we only need a single ANDI instruction which keeps the top nibble, which can then be compared to check for two byte (110x) or three byte (1110) characters. Far more efficient indeed.

[WL] The scanTable routine is probably the most time consuming part of the code, so I'd have written it as follows:

```

scanTable
    move.l    a2, a3                ; point to table
    move.w    #59, d0              ; there are 60 words to compare

scanLoop
    cmp.w     (a3)+, d2             ; is it a match?
    beq.s     scanDone             ; ...yes ->
    dbf       d0, scanLoop         ; try all permitted values
    rts                          ; no match found, return NZ from cmp

scanDone
    move.l    a3, d0               ; where we found it (+2)
    sub.l     a2, d0               ; index into table
    subq.w    #2, d0               ; but we overshot by 2 bytes

    lsr.w     #1, d0               ; offset into index
    add.w     #$80, d0             ; convert to character code
    cmp.w     d0, d0               ; see below
    rts                          ; the condition code Z is set by the cmp

```

[ND] Curses, I've been found out! My way was easier for me as I didn't have to count up however many two byte characters there were! However, as they say about Unix/Linux, there's more than one way to skin a cat, but again, I prefer your method.

[WL] There are a few more instructions when you find the correct value, but the search loop itself is smaller and will be faster (unless the value searched for is the very first in the table, and even then it'll be a close match). The CMP D0,D0 is there so that the routine returns with the Z flag set, without affecting any other register by zeroing it.

[ND] I wasn't fond of the non-standard way of detecting an error in my version, I have to admit. This is far far better.

[WL] So, coming back from calling the routine at label doScan, a simple BNE.S ERROR will do:

```

doScan
    bsr.s     scanTable
    bne.s     invalidUTF8
    (... success in d0 ...)

```

[ND] Agreed, this is better and resembles more a standard error return, zero is good, non-zero is not good.

[WL] At label twoBytes, you should be able to write:

```
twoBytes
    lsl.w    #8,d1        ; move byte up
    bsr     readByte      ; get next byte into LSB of D1
```

[WL] You should now have the correct word in D1. Remember, though, as of then to test on D1, not D2, for valid utf, even in the scanTable loop.

**Note**, this presumes that the trap handler does its work correctly and *only* modifies the LSB<sup>3</sup> of D1 to put the returned value in there. (Unlike, e.g. some early versions of SMSQulator which just reset the *entire* register to 0 and then sets the byte. Ouch!)

[ND] I think that I shall leave the code using D2, just in case it causes problems elsewhere then. Better safe than sorry.

[WL] Most of these comments go a little beyond just checking the code itself, I hope you don't mind.

[ND] No, I don't mind and in fact I welcome comments on anything printed in this ePeriodical. If you have a problem with my writing style, code etc, I'm happy to hear from you. From anyone that is!

## 2.5 A Better Ql2utf8

```
1 ;
2 ; QL2UTF8:
3 ;
4 ; This filter converts QL text files to UTF8 for use on Linux, Mac or
5 ; Windows where most modern editors etc, default to UTF8.
6 ;
7 ;
8 ; EX ql2utf8_bin, input_file, output_file_or_channel
9 ;
10 ;
11 ; 26/09/2019 NDunbar Created for QDOSMSQ Assembly Mailing List
12 ; 07/10/2019 WLenerz Many improvements.
13 ;
14 ; (c) Norman Dunbar, Wolfgang Lenerz 2019. Permission granted for
15 ; unlimited use or abuse, without attribution being required.
16 ; Just enjoy!
17 ;
18 ;
19 ; How many channels do I want?
20 numchans equ 2 ; How many channels required?
21
22
23 ; Stack stuff.
24 sourceId equ $02 ; Offset(A7) to input file id
25 destId equ $06 ; Offset(A7) to output file id
26
27 ; Other Variables
```

<sup>3</sup>LSB = Lowest Significant Byte.



```

28 pound      equ      96          ; UK Pound sign.
29 copyright  equ      127         ; (c) sign.
30 grave      equ      159         ; Backtick/Grave accent.
31 euro       equ      181         ; Euro symbol
32 err_bp     equ      -15         ; Bad parameter
33 err_eof    equ      -10         ; End of file
34 err_or     equ      -4          ; Out of range
35 me         equ      -1          ; This job's id
36 timeout    equ      -1          ; Infinty, and beyond!
37
38 ;-----
39 ; Uncomment the following if you are using QMAC as your assembler.
40 ;-----
41 ; io_fbyte   equ      1          ; Fetch one byte
42 ; io_sbyte   equ      5          ; Send one byte
43 ; mt_frjob   equ      5          ; Force remove a job
44 ;-----
45
46 ;=====
47 ; Here begins the code.
48 ;-----
49 ; Stack on entry:
50 ;
51 ; $06(a7) = Output file channel id.
52 ; $02(a7) = Source file channel id.
53 ; $00(a7) = How many channels? Should be $02.
54 ;=====
55 start
56     bra.s    checkStack
57
58     dc.l     $00
59     dc.w     $4afb
60 name
61     dc.w     name_end-name-2
62     dc.b     'QL2UTF8'
63 name_end    equ      *
64
65 version     dc.w     vers_end-version-2
66             dc.b     'Version 1.01'
67 vers_end    equ      *
68
69
70 bad_parameter
71     moveq    #err_bp,d0          ; Guess!
72     bra      errorExit          ; Die horribly
73
74
75 ;-----
76 ; Check the stack on entry. We only require NUMCHAN channels - any
77 ; thing other than NUMCHANS will result in a BAD PARAMETER error on
78 ; exit from EW (but not from EX).
79 ;-----
80 checkStack
81     cmpi.w   #numchans,(a7)      ; Two channels is a must
82     bne.s    bad_parameter       ; Oops
83

```

```

84 ;-----
85 ; Initialise a couple of registers that will keep their values all
86 ; through the rest of the code.
87 ;-----
88 ql2utf8
89     lea     utf8 ,a2          ; Preserved throughout
90     moveq   #timeout ,d3      ; Timeout, also Preserved
91     move.l  sourceID(a7),a4    ; Input channel id
92     move.l  destId(a7),a5      ; Output channel id
93
94 ;-----
95 ; The main loop starts here. Read a single byte, check for EOF etc.
96 ;-----
97 readLoop
98     moveq   #io_fbyte ,d0      ; Fetch one byte
99     move.l  a4,a0              ; Channel to readLoop
100    trap     #3                ; Do input
101    tst.l    d0                ; OK?
102    beq.s    testBit7          ; Yes
103    cmpi.l   #err_eof ,d0      ; All done?
104    beq      allDone           ; Yes.
105    bra      errorExit         ; Oops!
106
107 testBit7
108     bst     #7,d1              ; Bit 7 set?
109     bne.s   twoBytes          ; Multi Byte character if so
110
111 ;-----
112 ; The UK Pound and copyright signs are exceptions to the "bytes
113 ; less than $80 are the same in UTF8 as they are in ASCII" rule as
114 ; Sir Clive didn't follow ASCII 100%. Both characters are multi-byte
115 ; in UTF8.
116 ;-----
117 testPound
118     cmpi.b  #pound,d1         ; Got a UK Pound sign?
119     bne.s   testCopyright     ; No.
120
121 gotPound
122     move.b  #$c2,d1           ; Pound is $C2A3 in UTF8.
123     bsr.s   writeByte         ; Write first byte
124     move.b  #$a3,d1           ;
125     bra.s   oneByte           ; Write out & carry on.
126
127 ;-----
128 ; Here we repeat the same check as above, in case we have the
129 ; copyright sign.
130 ;-----
131 testCopyright
132     cmpi.b  #copyright,d1     ; Got a copyright sign?
133     bne.s   oneByte           ; No.
134
135 gotCopyright
136     move.b  #$c2,d1           ; Copyright is $C2A9 in UTF8
137     bsr.s   writeByte         ; Write first byte
138     move.b  #$a9,d1           ; Then drop in to write & carry on
139

```

```

140 ;-----
141 ; All other ASCII characters , below $80 , are single byte in UTF8 and
142 ; are the same code as in ASCII. Stack the address of readLoop and
143 ; drop into writeByte. On RTS, we will hit the top of the loop again.
144 ; (Courtesy Wolfgang Lenerz.)
145 ;-----
146 oneByte
147     pea    readLoop
148
149 ;-----
150 ; A small but perfectly formed subroutine to send the byte in D1 to
151 ; the output channel.
152 ;-----
153 writeByte
154     moveq   #io_sbyte ,d0      ; Send one byte
155     move.l  a5,a0              ; Output channel id
156     trap    #3
157     tst.l   d0                 ; OK?
158     bne     errorExit          ; Oops!
159     rts
160
161 ;-----
162 ; ASCII codes from $80 upwards require multiple bytes in UTF8. In the
163 ; case of the QL, these are mostly 2 bytes long. I could use IO_SSTRG
164 ; here , I know.
165 ; However, as ever, there are exceptions. The grave accent (backtick)
166 ; is a single byte on output , while the 4 arrow keys are three bytes.
167 ; The bytes to be sent are read from a table because , again , the QL
168 ; is not using the full set of accented characters – so there is
169 ; mucking about to be done.
170 ;-----
171 twoBytes
172     cmpi.b  #grave,d1          ; Backtick/Grave accent?
173     bne.s   testEuro           ; No.
174
175 ;-----
176 ; We are dealing with a backtick character (aka Grave accent)?
177 ;-----
178 gotGrave
179     move.b  #pound,d1          ; Grave in = pound out!
180     bra.s   oneByte            ; Write out & carry on
181
182 ;-----
183 ; Here we repeat the same check as above , in case we have the
184 ; Euro sign.
185 ;-----
186 testEuro
187     cmpi.b  #euro,d1           ; Got a Euro sign?
188     bne.s   testArrows         ; No.
189
190 gotEuro
191     move.b  #$e2,d1             ; Euro is $E282AC in UTF8
192     bsr.s   writeByte           ; Write first byte
193     move.b  #$82,d1             ;
194     bsr.s   writeByte           ; Write second byte
195     move.b  #$ac,d1             ;

```

```

196     bra.s    oneByte          ; Write out and carry on
197
198 ;-----
199 ; The arrows are $BC, $BD, $BE and $BF (left, right, up, down). These
200 ; are three bytes in UTF8, $E2 $86 $9x where 'x' is 0, 2, 1 or 3.
201 ;-----
202 testArrows
203     move.b   d1,d2            ; Copy character code
204     subi.b   #$bc,d2          ; Anything lower = C set
205     bcs.s    notArrows        ; And is not an arrow
206     subq.b   #4,d2            ; Arrows = 0-3. C clear is bad
207     bcc.s    notArrows        ; Still not an arrow.
208
209 gotArrows
210     subi.b   #$bc,d1          ; Correct arrow code, 0 - 3
211     lea       arrows,a3        ; Arrow table
212     move.b   d1,d2            ; Save index into table
213     ext.w     d2              ; Need word not byte
214
215     move.b   #$e2,d1          ; First byte
216     bsr.s    writeByte
217     move.b   #$86,d1          ; Second byte
218     bsr.s    writeByte
219     move.b   0(a3,d2.w),d1     ; Third byte
220     bra.s    oneByte          ; Write it & go around again.
221
222 ;-----
223 ; We need this as arrows in the QL are Left, Right, Up, Down but in
224 ; UTF8 they are Left, Up, Right, Down. Sigh.
225 ;-----
226 arrows
227     dc.b     $90,$92,$91,$93  ; Awkward byte order!
228
229 ;-----
230 ; Now we are certain, everything is two bytes. Read them from the
231 ; table and write them out. However, there are only 60 entries in the
232 ; table - best we check!
233 ;-----
234 notArrows
235     cmpi.b   #59,d1           ; Are we in range for the table?
236     bcc.s    inRange          ; Yes
237
238 outOfRange
239     moveq     #err_or,d0       ; Out of range
240     bra.s     errorExit        ; Oops!
241
242 inRange
243     move.b   d1,d2            ; D2 = byte just read
244     subi.b   #$80,d2          ; Adjust for table index
245     ext.w     d2              ; Word size needed
246     lsl.w     #1,d2           ; Double D2 for Offset
247     move.b   0(a2,d2.w),d1     ; First byte
248     bsr.s    writeByte        ; Send it output
249     move.b   1(a2,d2.w),d1     ; Second byte
250     bra      oneByte          ; Write it and go around
251

```



```

252 ;
253 ;-----
254 ; No errors , exit quietly back to SuperBASIC .
255 ;-----
256 allDone
257     moveq    #0,d0
258
259 ;-----
260 ; We have hit an error so we copy the code to D3 then exit via a
261 ; forcible removal of this job. EXEC_W/EW will display the error in
262 ; SuperBASIC , but EXEC/EX will not.
263 ;-----
264 errorExit
265     move.l    d0,d3                ; Error code we want to return
266
267 ;-----
268 ; Kill myself when an error was detected , or at EOF.
269 ;-----
270 suicide
271     moveq     #mt_frjob,d0        ; This job will die soon
272     moveq     #me,d1
273     trap      #1
274
275 ;-----
276 ; The following table contains the two byte sequences required for
277 ; QL characters above $80. These are all 2 bytes in UTF8, so quite a
278 ; simple case. (Not when converting UTF8 to QL though!) There are 60
279 ; QL characters which convert to two byte UTF8 characters .
280 ;-----
281 utf8
282     dc.w      $c3a4                ; a umlaut
283     dc.w      $c3a3                ; a tilde
284     dc.w      $c3a2                ; a circumflex
285     dc.w      $c3a9                ; e acute
286     dc.w      $c3b6                ; o umlaut
287     dc.w      $c3b5                ; o tilde
288     dc.w      $c3b8                ; o slash
289     dc.w      $c3bc                ; u umlaut
290     dc.w      $c3a7                ; c cedilla
291     dc.w      $c3b1                ; n tilde
292     dc.w      $c3a6                ; ae ligature
293     dc.w      $c593                ; oe ligature
294     dc.w      $c3a1                ; a acute
295     dc.w      $c3a0                ; a grave
296     dc.w      $c3a2                ; a circumflex
297     dc.w      $c3ab                ; e umlaut
298     dc.w      $c3a8                ; e grave
299     dc.w      $c3aa                ; e circumflex
300     dc.w      $c3af                ; i umlaut
301     dc.w      $c3ad                ; i acute
302     dc.w      $c3ac                ; i grave
303     dc.w      $c3ae                ; i circumflex
304     dc.w      $c3b3                ; o acute
305     dc.w      $c3b2                ; o grave
306     dc.w      $c3b4                ; o circumflex
307     dc.w      $c3ba                ; u acute

```

```

308      dc.w      $c3b9      ; u grave
309      dc.w      $c3bb      ; u circumflex
310      dc.w      $ceb2      ; B as in ss (German)
311      dc.w      $c2a2      ; Cent
312      dc.w      $c2a5      ; Yen
313      dc.w      $0000      ; Grave accent – single byte
314      dc.w      $c384      ; A umlaut
315      dc.w      $c383      ; A tilde
316      dc.w      $c385      ; A circle
317      dc.w      $c389      ; E acute
318      dc.w      $c396      ; O umlaut
319      dc.w      $c395      ; O tilde
320      dc.w      $c398      ; O slash
321      dc.w      $c39c      ; U umlaut
322      dc.w      $c387      ; C cedilla
323      dc.w      $c391      ; N tilde
324      dc.w      $c386      ; AE ligature
325      dc.w      $c592      ; OE ligature
326      dc.w      $ceb1      ; alpha
327      dc.w      $ceb4      ; delta
328      dc.w      $ceb8      ; theta
329      dc.w      $cebb      ; lambda
330      dc.w      $c2b5      ; micro (mu?)
331      dc.w      $cf80      ; PI
332      dc.w      $cf95      ; o pipe
333      dc.w      $c2a1      ; ! upside down
334      dc.w      $c2bf      ; ? upside down
335      dc.w      $0000      ; Euro
336      dc.w      $c2a7      ; Section mark
337      dc.w      $c2a4      ; Currency symbol
338      dc.w      $c2ab      ; <<
339      dc.w      $c2bb      ; >>
340      dc.w      $c2ba      ; Degree
341      dc.w      $c3b7      ; Divide

```

Listing 2.1: Wolfgang's improved ql2utf8 Utility

## 2.6 A Better Utf82ql

```

1  ;
2  ; UTF82QL:
3  ;
4  ; This filter converts UTF8 text files from Linux, Mac or Windows to
5  ; to the SMSQ character set.
6  ;
7  ;
8  ; EX utf82ql2_bin , input_file , output_file_or_channel
9  ;
10 ;
11 ; 28/09/2019 NDunbar Created for QDOSMSQ Assembly Mailing List.
12 ; 07/10/2019 WLenerz Many improvements.
13 ;
14 ; (c) Norman Dunbar, Wolfgang Lenerz, 2019. Permission granted for
15 ; unlimited use or abuse, without attribution being required.
16 ; Just enjoy!
17 ;
18 ;

```

```

19 ; How many channels do I want?
20 numchans     equ     2           ; How many channels required?
21
22
23 ; Stack stuff.
24 sourceId     equ     $02         ; Offset(A7) to input file id
25 destId       equ     $06         ; Offset(A7) to output file id
26
27 ; Other Variables
28 utf8Pound    equ     $c2a3       ; UTF8 Pound sign
29 qlPound      equ     96          ; QL Pound sign
30
31 utf8Grave     equ     96          ; UTF8 Grave code
32 qlGrave      equ     159         ; QL Grave code
33
34 utf8Copyright equ     $c2a9       ; UTF8 copyright
35 qlCopyright  equ     127         ; QL copyright sign
36
37 qlEuro       equ     181         ; SMSQ Euro symbol
38
39 err_exp      equ     -17
40 err_bp       equ     -15
41 err_eof      equ     -10
42 err_or       equ     -4
43 me           equ     -1
44 timeout      equ     -1
45
46 ;-----
47 ; Uncomment the following if you are using QMAC as your assembler.
48 ;-----
49 ; io_fbyte    equ     1          ; Fetch one byte
50 ; io_sbyte    equ     5          ; Send one byte
51 ; mt_frjob    equ     5          ; Force remove jobs
52 ;-----
53
54
55 ;=====
56 ; Here begins the code.
57 ;-----
58 ; Stack on entry:
59 ;
60 ; $06(a7) = Output file channel id.
61 ; $02(a7) = Source file channel id.
62 ; $00(a7) = How many channels? Should be $02.
63 ;=====
64 start        bra.s    checkStack
65
66             dc.l      $00
67             dc.w      $4afb
68 name         dc.w      name_end-name-2
69             dc.b      'UTF82QL'
70 name_end     equ      *
71
72 version      dc.w      vers_end-version-2
73             dc.b      'Version 1.00'
74 vers_end     equ      *

```

```

75
76
77 bad_parameter
78     moveq    #err_bp,d0        ; Guess!
79     bra      errorExit        ; Die horribly
80
81
82 ;-----
83 ; Check the stack on entry. We only require NUMCHAN channels – any
84 ; thing other than NUMCHANS will result in a BAD PARAMETER error on
85 ; exit from EW (but not from EX).
86 ;-----
87 checkStack
88     cmpi.w    #numchans,(a7)    ; Two channels is a must
89     bne.s     bad_parameter     ; Oops
90
91 ;-----
92 ; Initialise a couple of registers that will keep their values all
93 ; through the rest of the code.
94 ;-----
95 ql2utf8
96     lea       utf8,a2          ; Preserved throughout
97     moveq     #timeout,d3       ; Timeout, also Preserved
98     move.l    sourceId(a7),a4   ; Channel ID for UTF8 input file
99     move.l    destId(a7),a5     ; Channel ID for QL output file
100
101 ;-----
102 ; The main loop starts here. Read a single byte, check for EOF etc.
103 ;-----
104 readLoop
105     bsr       readByte         ; Read one byte
106     beq.s     testBit7         ; No errors is good.
107     cmpi.l    #err_eof,d0      ; All done?
108     beq       allDone          ; Yes.
109     bra       errorExit        ; Oops!
110
111 ;-----
112 ; Test the top bit here. If it is zero, we are good for most single
113 ; byte characters, otherwise it is potentially multi-byte.
114 ;-----
115 testBit7
116     btst      #7,d1            ; Bit 7 set?
117     bne.s     multiBytes       ; Multi Byte character if so
118
119 ;-----
120 ; In UTF8, the Grave accent (backtick) is a single byte character but
121 ; the byte value doesn't correspond to that on the QL. On UTF8 it is
122 ; $60 (96) but on the QL it is $9F (159) so, this is another Sir
123 ; Clive induced exception!
124 ;-----
125 testGrave
126     cmpi.b    #utf8Grave,d1     ; Got a grave!
127     bne.s     oneByte          ; Must be a single byte if not a pound.
128
129 gotGrave
130     move.b    #qlGrave,d1       ; Write a grave character

```



```

131
132 ;-----
133 ; The byte read is a valid single byte character so it has the exact
134 ; same code in the QL's variation of ASCII, just write it out.
135 ;-----
136 oneByte
137     bsr      writeByte      ; Write the byte out
138     bra.s    readLoop       ; And continue.
139
140
141 ;-----
142 ; Most of the remaining characters will be two bytes in UTF8 and one
143 ; byte on the QL. There are a few exceptions though – the Euro and
144 ; the four arrow keys are three bytes long in UTF8.
145 ;-----
146 multiBytes
147     move.b   d1,d2           ; Copy character code
148     andi.b   #%11110000,d2   ; Keep top four bits
149     cmpi.b   #%11000000,d2   ; Two bytes?
150     bne.s    testThree      ; Yes.
151
152 ;-----
153 ; At this point we should have a UTF8 two byte character but we only
154 ; have the first byte in D1. We need the second byte also, so read it
155 ; and check that it is indeed valid.
156 ;-----
157 twoBytes
158     move.b   d1,d2           ; Save the leading byte
159     bsr      readByte        ; Read the second byte
160     lsl.w    #8,d2           ; Shift first byte upwards
161     or.b     d1,d2           ; And add the new byte
162
163 ;-----
164 ; Exception checking. UTF8 codes $C2A3 for the UK Pound and $C2A9 for
165 ; copyright, are not in the table. They are QL codes $60 (96) and $7F
166 ; (127) and are exceptions to the rule that a QL code less than 128
167 ; always has a one byte code in UTF8 – they are both two bytes.
168 ;-----
169 testPound
170     cmpi.w   #utf8Pound,d2   ; Got a UK Pound?
171     bne.s    testCopyright   ; No
172
173 gotPound
174     move.b   #qlPound,d1     ; QL Pound code
175     bra.s    oneByte         ; Write it out & loop around
176
177 testCopyright
178     cmpi.w   #utf8Copyright,d2 ; Got a copyright?
179     bne.s    doScan          ; No
180
181 gotCopyright
182     move.b   #qlCopyright,d1 ;
183     bra.s    oneByte         ; Write it out & loop around
184
185 ;-----
186 ; Ok, exceptions processed, do the remaining two byte characters.

```

```

187 ;
188 doScan
189     bsr     scanTable      ; Is this valid UTF8?
190     bne.s   invalidUTF8    ; Nope
191
192 validUTF8
193     move.b  d0,d1          ; Get the character code
194     bsr.s   writeByte      ; Write it out
195     bra     readLoop       ; And continue.
196
197 invalidUTF8
198     moveq   #err_exp,d0     ; Error in expression
199     bra     errorExit       ; Bale out.
200
201 ;
202 ; We are interested in a few three byte characters , so we check those
203 ; next. These are identified by the top nibble of the first character
204 ; read in being 1110.
205 ;
206 testThree
207     cmpi.b  #%11100000,d2   ; Three bytes?
208     bne.s   invalidUTF8    ; No.
209
210 ;
211 ; At this point we should have a UTF8 three byte character but we
212 ; only have the first byte in D1. We need the second byte also , so
213 ; read it and check that it is indeed valid. Then get the third byte.
214 ; All our three byte characters should have $E2 in the first byte.
215 ;
216 ; The Euro is $E282AC.
217 ; The Arrows are $E2869x where 'x' is 0,1,2 or 3.
218 ;
219 threeBytes
220     cmpi.b  #$e2,d1         ; Valid three byte?
221     bne.s   invalidUTF8    ; Looks unlikely.
222
223     move.b  d1,d2          ; Save the first byte
224     bsr.s   readByte       ; Get the second byte
225     cmpi.b  #$82,d1         ; Euro second byte?
226     beq.s   threeValid     ; Yes
227     cmpi.b  #$86,d1         ; Arrow second byte?
228     bne.s   invalidUTF8    ; Sadly , no, error out.
229
230 threeValid
231     lsl.w   #8,d2           ; Shift first byte upwards
232     or.b    d1,d2          ; And add the new byte
233     bsr.s   readByte       ; Get the third byte
234     cmpi.w  #$e282,d2       ; Euro possibly?
235     bne.s   threeArrows    ; No, try arrows
236
237 ;
238 ; We have read $e282 so if we get $ac next , we have the euro. If not
239 ; it's an error in the UTF8 characters that the QL understands.
240 ;
241 threeEuro
242     cmpi.b  #$ac,d1         ; Need this for the Euro

```

```

243     bne.s    invalidUTF8      ; No, error out again.
244     move.b   #qlEuro,d1      ; QL Euro code
245     bsr.s    writeByte       ; Write it out
246     bra      readLoop        ; And continue.
247
248
249 ;-----
250 ; The QL arrows are $BC, $BD, $BE and $BF (left, right, up, down).
251 ; The UTF8, $E2869x where 'x' is 0, 2, 1 or 3 to correspond with the
252 ; order of the QL arrow codes.
253 ;-----
254 threeArrows
255     cmpi.w    #$e286,d2      ; Got a potential arrow code?
256     bne.s     invalidUTF8    ; Fraid not, error out.
257     subi.b    #$90,d1        ; D1 is now 0-3 for valid arrows
258     bmi.s     invalidUTF8    ; Oops, it went negative
259     cmpi.b    #3,d1          ; Highest arrow code
260     bhi.s     invalidUTF8    ; Oops, invalid arrow code.
261     addi.b    #$bc,d1        ; Convert to QL arrow code.
262     bsr.s     writeByte      ; Write it out
263     bra      readLoop        ; And continue.
264
265
266 ;-----
267 ; A small but perfectly formed subroutine to send the byte in D1 to
268 ; the output QL file.
269 ; On Entry, A0 = input channel ID and A3 = output channel ID.
270 ; On exit, D0 = 0, Z set.
271 ; On error, never returns.
272 ;-----
273 writeByte
274     move.l    a5,a0          ; Get the correct channel ID
275     moveq     #io_sbyte,d0   ; Send one byte
276     trap      #3
277     tst.l     d0             ; OK?
278     bne.s     errorExit      ; Oops!
279     rts
280
281 ;-----
282 ; Another perfectly formed subroutine to read one byte into D1
283 ; from the input UTF8 file.
284 ; On Entry, A0 = output channel ID and A3 = input channel ID.
285 ; On exit, error codes in D0, Z set if no error and D1.B = character
286 ; just read.
287 ;-----
288 readByte
289     move.l    a4,a0          ; Get the correct channel ID
290     moveq     #io_fbyte,d0   ; Fetch one byte
291     trap      #3             ; Do input
292     tst.l     d0             ; OK?
293     rts
294
295 ;-----
296 ; Scan the UTF8 table looking for the word in D2. If found, we have
297 ; the table offset in D0 and that is then halved to get the index which
298 ; is still $80 below the correct character code — we add to convert.

```

```

299 ; Returns with D0 = the character code , or $FFFF to show the end was
300 ; reached and we appear to have an invalid two byte character. A2
301 ; holds the table address. D7 is a working register.
302 ;
303 scanTable
304     move.l    a2,a3                ; Get start of table
305     move.w    #59,d0              ; There are 60 entries in the table
306
307 scanLoop
308     cmp.w     (a3)+,d2            ; Found it yet?
309     beq.s     scanDone           ; Yes
310     dbf       d0,scanLoop        ; No, try again
311     rts                          ; Not found, Z not set.
312
313 scanDone
314     move.l    a3,d0              ; Address in table + 2
315     sub.l     a2,d0              ; Address now the Offset + 2
316     subq.w    #2,d0              ; Adjusted to correct offset
317     lsr.w     #1,d0              ; Conver to index
318     add.w     #$80,d0            ; Now correct character code
319     cmp.w     d0,d0              ; Sets Z flag
320     rts
321
322 ;
323 ; No errors , exit quietly back to SuperBASIC.
324 ;
325 allDone
326     moveq     #0,d0
327
328 ;
329 ; We have hit an error so we copy the code to D3 then exit via a
330 ; forcible removal of this job. EXEC_W/EW will display the error in
331 ; SuperBASIC, but EXEC/EX will not.
332 ;
333 errorExit
334     move.l    d0,d3              ; Error code we want to return
335
336 ;
337 ; Kill myself when an error was detected , or at EOF.
338 ;
339 suicide
340     moveq     #mt_frjob,d0       ; This job will die soon
341     moveq     #me,d1
342     trap      #1
343
344 ;
345 ; The following table contains the two byte sequences required for
346 ; QL characters from character $80 onwards. Those flagged as $FFFF
347 ; are exceptions , dealt with in the code. There are no entries for
348 ; the arrow keys as they would simply be zero words at the end of the
349 ; table.
350 ;
351 utf8
352     dc.w      $c3a4              ; a umlaut
353     dc.w      $c3a3              ; a tilde
354     dc.w      $c3a2              ; a circumflex

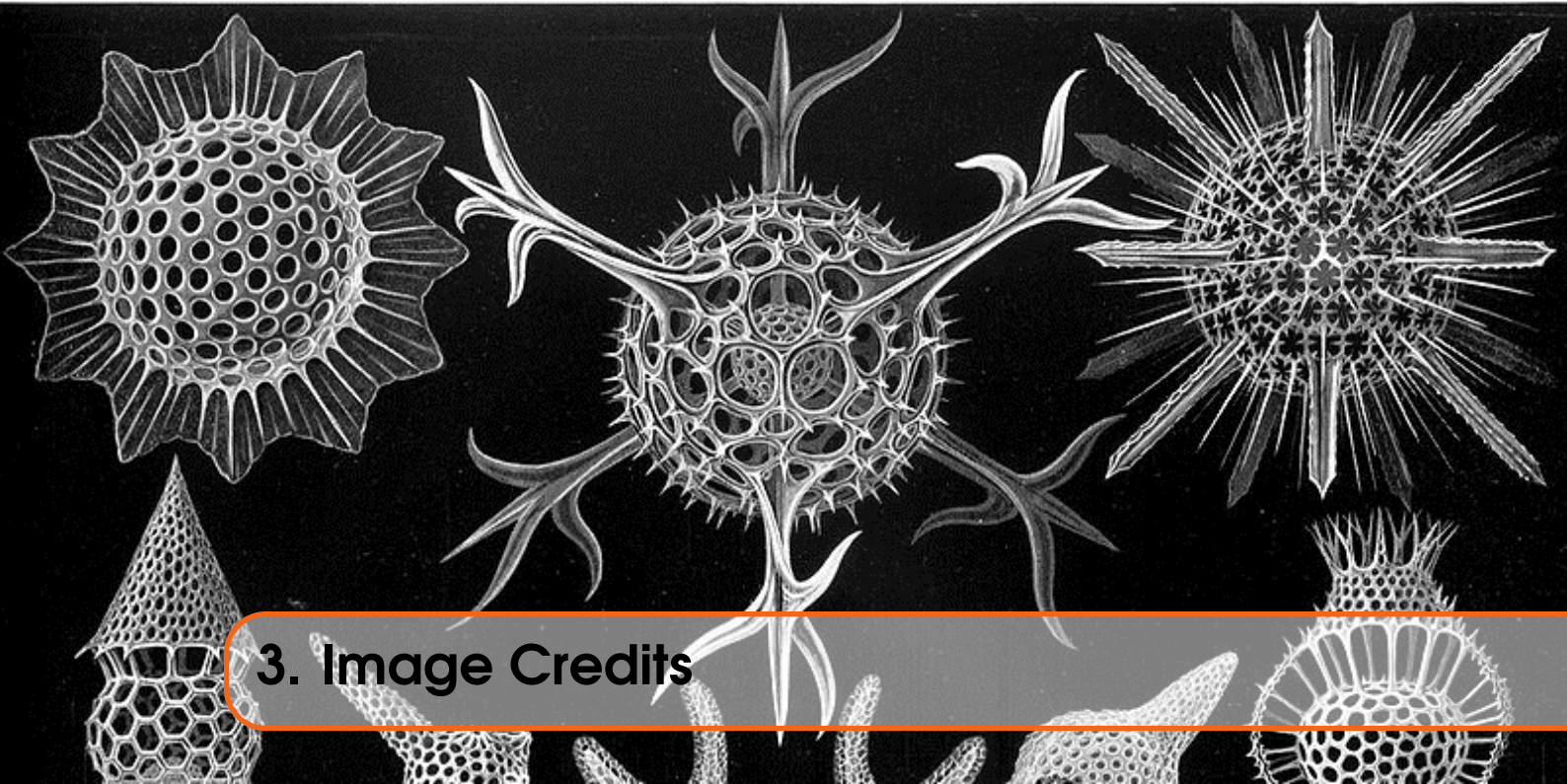
```

355	dc.w	\$c3a9	; e acute
356	dc.w	\$c3b6	; o umlaut
357	dc.w	\$c3b5	; o tilde
358	dc.w	\$c3b8	; o slash
359	dc.w	\$c3bc	; u umlaut
360	dc.w	\$c3a7	; c cedilla
361	dc.w	\$c3b1	; n tilde
362	dc.w	\$c3a6	; ae ligature
363	dc.w	\$c593	; oe ligature
364	dc.w	\$c3a1	; a acute
365	dc.w	\$c3a0	; a grave
366	dc.w	\$c3a2	; a circumflex
367	dc.w	\$c3ab	; e umlaut
368	dc.w	\$c3a8	; e grave
369	dc.w	\$c3aa	; e circumflex
370	dc.w	\$c3af	; i umlaut
371	dc.w	\$c3ad	; i acute
372	dc.w	\$c3ac	; i grave
373	dc.w	\$c3ae	; i circumflex
374	dc.w	\$c3b3	; o acute
375	dc.w	\$c3b2	; o grave
376	dc.w	\$c3b4	; o circumflex
377	dc.w	\$c3ba	; u acute
378	dc.w	\$c3b9	; u grave
379	dc.w	\$c3bb	; u circumflex
380	dc.w	\$ceb2	; B as in ss (German)
381	dc.w	\$c2a2	; Cent
382	dc.w	\$c2a5	; Yen
383	dc.w	\$ffff	; Grave accent – single byte
384	dc.w	\$c384	; A umlaut
385	dc.w	\$c383	; A tilde
386	dc.w	\$c385	; A circle
387	dc.w	\$c389	; E acute
388	dc.w	\$c396	; O umlaut
389	dc.w	\$c395	; O tilde
390	dc.w	\$c398	; O slash
391	dc.w	\$c39c	; U umlaut
392	dc.w	\$c387	; C cedilla
393	dc.w	\$c391	; N tilde
394	dc.w	\$c386	; AE ligature
395	dc.w	\$c592	; OE ligature
396	dc.w	\$ceb1	; alpha
397	dc.w	\$ceb4	; delta
398	dc.w	\$ceb8	; theta
399	dc.w	\$cebb	; lambda
400	dc.w	\$c2b5	; micro (mu?)
401	dc.w	\$cf80	; PI
402	dc.w	\$cf95	; o pipe
403	dc.w	\$c2a1	; ! upside down
404	dc.w	\$c2bf	; ? upside down
405	dc.w	\$ffff	; Euro
406	dc.w	\$c2a7	; Section mark
407	dc.w	\$c2a4	; Currency symbol
408	dc.w	\$c2ab	; <<
409	dc.w	\$c2bb	; >>
410	dc.w	\$c2ba	; Degree

```
411 | dc .w    $c3b7          ; Divide
412 |
413 | dc .w    $0000          ; End of table
```

Listing 2.2: Wolfgang's improved utf82ql Utility





### 3. Image Credits

The front cover image on this ePeriodical is taken from the book *Kunstformen der Natur* by German biologist Ernst Haeckel. The book was published between 1899 and 1904. The image used is of various *Polycystines* which are a specific kind of micro-fossil.

I have also cropped the image for use on each chapter heading page.

You can read about Polycystines on [Wikipedia](#) and there is a brief overview of the above book, also on [Wikipedia](#), which shows a number of other images taken from the book. (Some of which I considered before choosing the current one!)

Polycystines have absolutely nothing to do with the QL or computing in general - in fact, I suspect they died out before electricity was invented - but I liked the image, and decided that it would make a good cover for the book and a decent enough chapter heading image too.

Not that I am suggesting, *in any way whatsoever*, that we QL fans are ancient.