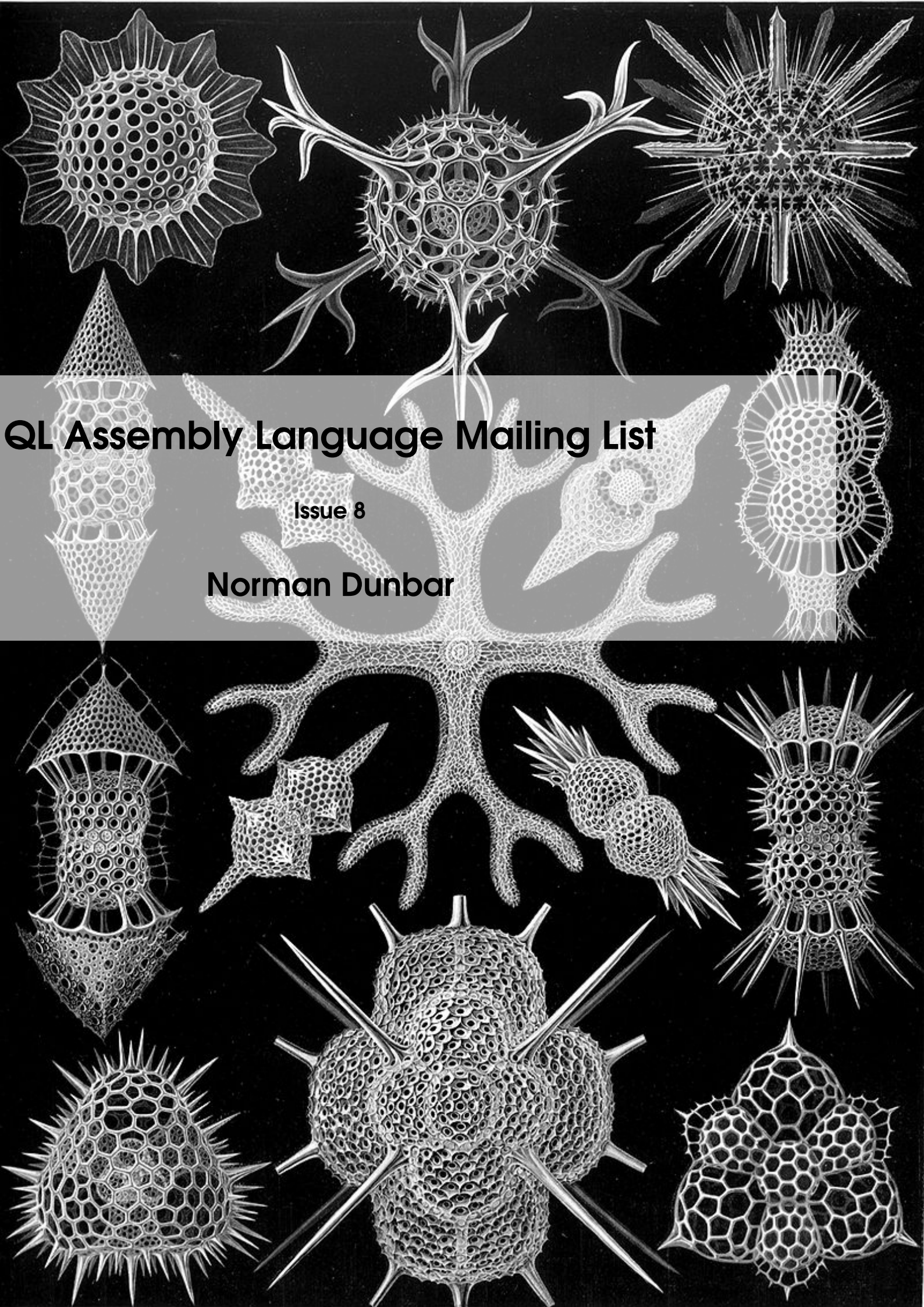


QL Assembly Language Mailing List

Issue 8

Norman Dunbar

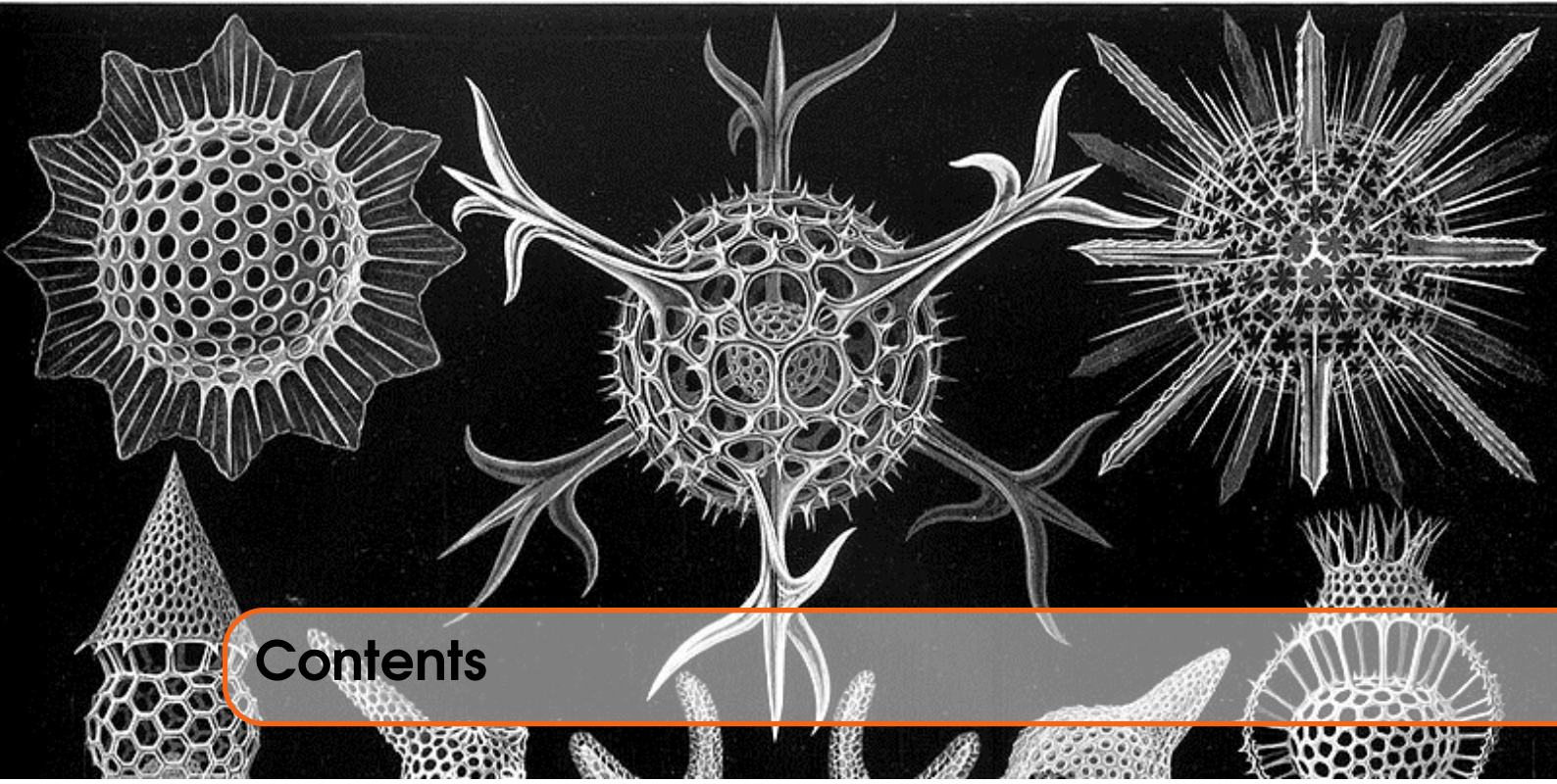
PUBLISHED BY MEMYSELFEYE PUBLISHING ;-)

Download from:
https://github.com/NormanDunbar/QLAssemblyLanguageMagazine/releases/tag/Issue_8

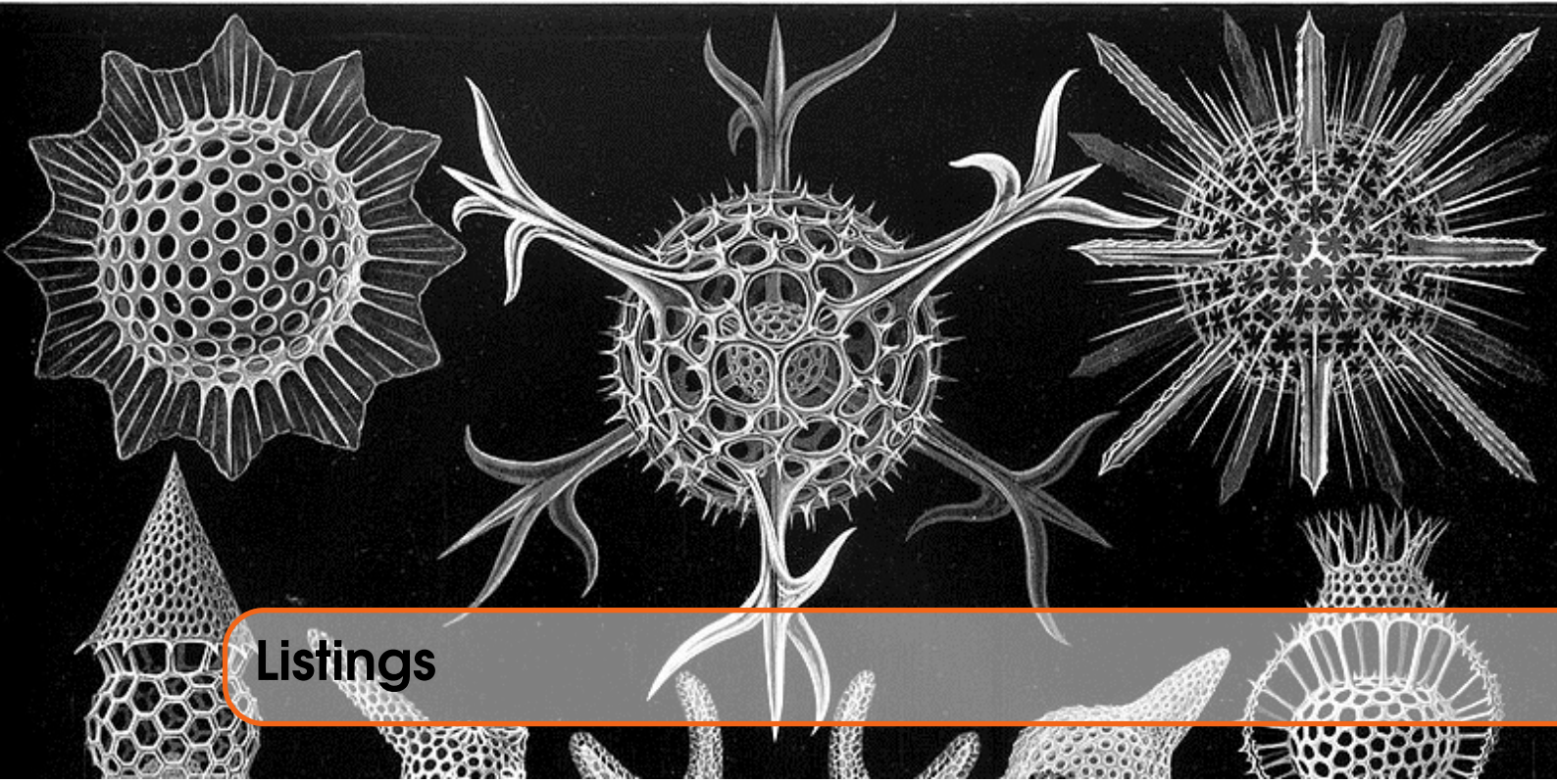
Licence:
Licensed under the Creative Commons Attribution-NonCommercial 3.0 Unported License (the "License"). You may not use this file except in compliance with the License. You may obtain a copy of the License at http://creativecommons.org/licenses/by-nc/3.0. Unless required by applicable law or agreed 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 *6/10/2019* at *19:19*.

Copyright ©2019 Norman Dunbar



Contents



Listings

1. Preface

1.1 Feedback

Please send all feedback to 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 LaTeXsource 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/mailinglist 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

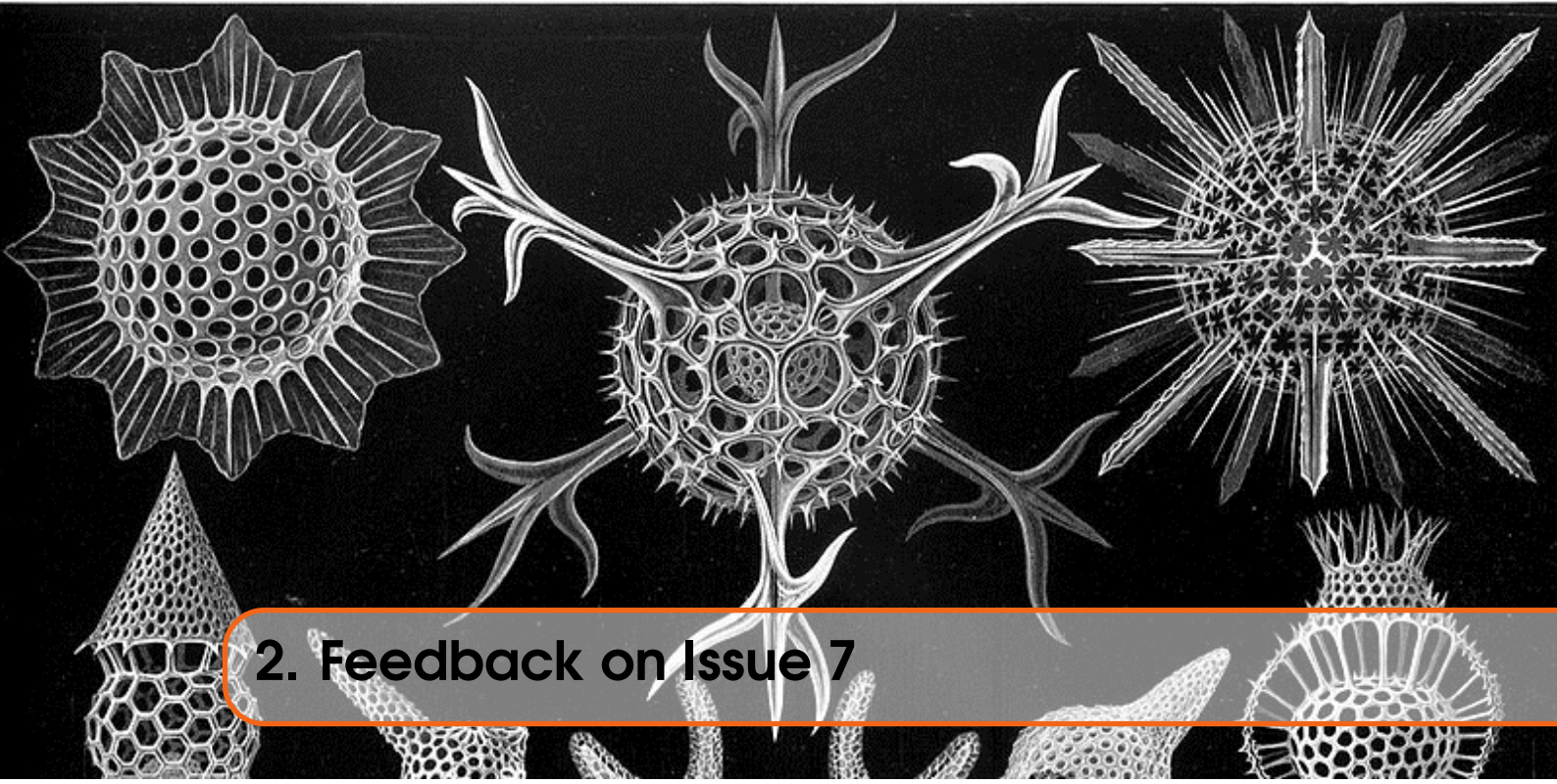
I'm rather hoping that this mailing list will not be a one-way affair, like QL Today appeared to be. I'm very open to suggestions, opinions, articles etc from my readers, otherwise how do I know what I'm doing is right or wrong?

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

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 LaTeXsource 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 `utf82ql` 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.[1] 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 used 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 `ASCIIDOCTOR`[2] format and convert them to PDF files using `asciidoctor-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

[1] 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.

[2] 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 automagically 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 `Utf82ql` 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 `Ql2utf8` code, incorporating all of Wolgang's changes and corrections.

2.4 *Even More* Feedback from Wolfgang Lenerz

[WL] I also had a look through the `utf2ql` code. Some of the comments made for the other routine (`ql2utf8` 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[3] of D1 to put the returned value in there. (Unlike, e.g. some early versions of SMSQmulator which just reset the *entire* register to 0 and then sets the byte. Ouch!)

[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 ... NORM - DO THE ABOVE CHANGES AND PASTE HERE ...
```

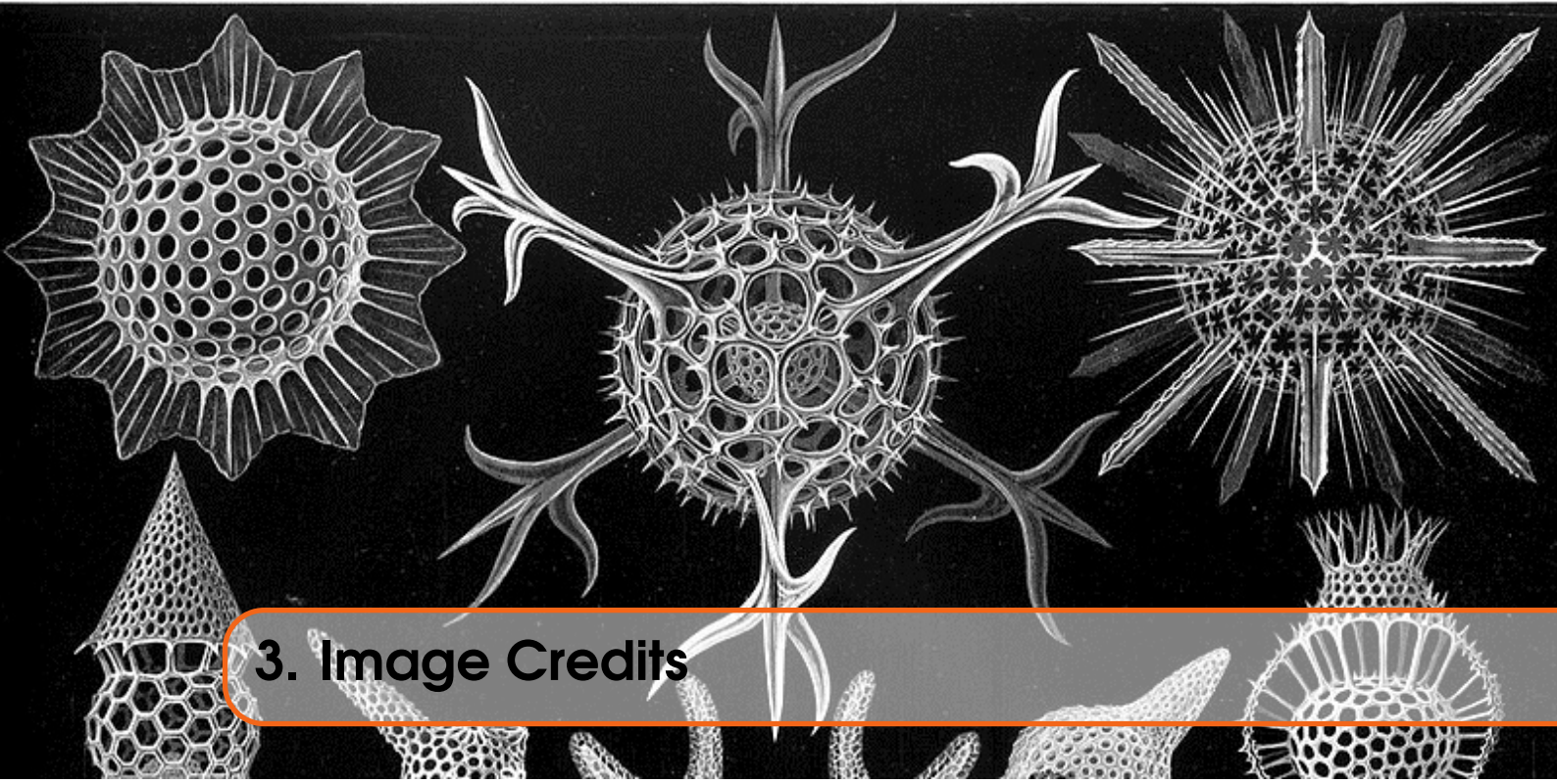
Listing 2.1: Wolfgang's improved ql2utf8 Utility

2.6 A Better Utf82ql

```
1
2 ... NORM - DO THE ABOVE CHANGES AND PASTE HERE ...
```

Listing 2.2: Wolfgang's improved utf82ql Utility

[3] LSB = Lowest Significant Byte.



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.