Typesetting the Byzantine *Cappelli*

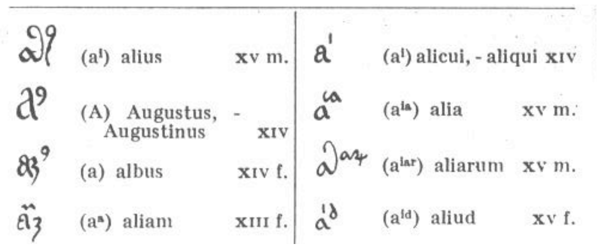
Philip Taylor
The Computer Centre, Royal Holloway,
University of London, TW20 0EX,
United Kingdom
mailto:P.Taylor@Rhul.Ac.Uk

Abstract

An overview of the author's rôle in the preparation of the forthcoming *Lexicon of Abbreviations & Ligatures in Greek Minuscule Hands*, with particular emphasis on two challenges: sorting TEX markup for polytonic Greek using multiple concurrent keys, and deriving statistical data which could be used to provide input to the book design.

Introduction

One of the greatest pleasures that I get from my position as Webmaster at Royal Holloway, University of London, is the only-too-rare opportunity to work with truly gifted and dedicated scholars. For the last few years, I have been truly privileged to be able to work with Miss Julian Chrysostomides, Director of our Hellenic Institute, and with Dr Charalambos Dendrinos, a Research Fellow within the same Institute. These two extraordinary scholars have both devoted a considerable portion of their lives to the collection, collation and preparation of material for a *Lexicon of Abbreviations & Ligatures in Greek Minuscule Hands* which is intended to do for Byzantine scholarship what Adriano Cappelli's *Dizionario di Abbreviature latine ed italiane* has been doing for Latin scholarship for the past 100 years.



(a^{l}) alius	xv m.		(a^{l}) alicui, - aliqui	xiv
(A) Augustus, - Augustinus	xiv		(a^{ia}) alia	xv m.
(a) albus	xiv f.		(a^{iar}) aliarum	xv m.
(a^{a}) aliam	xiii f.		(a^{id}) aliud	xv f.

Figure 1: A fragment from Cappelli's *Dizionario*

For both Latin and Byzantine scholars, the task of deciphering manuscripts which may be more than a thousand years old is not simply one of reading a long-dead scribe's handwriting: far more difficult is the task of identifying and correctly interpreting the various abbreviations, ligatures and other scribal shorthand notations that he or she may have used. Even a skilled palæographer may have difficulty in deciphering these, although for Latin scholars Cappelli's *Dizionario* provides an invaluable tool.

Only too aware of the difficulties that their students were experiencing in attempting to decipher Byzantine manuscripts, Julian & Charalambos decided to compile a Byzantine dictionary that would provide their students, and future scholars, with a key to those scribal notations which were most likely to cause problems in interpretation. For over five years, these two scholars have been painstakingly researching and deciphering hundreds if not thousands of individual manuscripts and recording the results of their work, initially using fairly primitive technology such as Windows 3.1's *Cardfile* and Eberhard Mattes' *emTEX* but more recently using spreadsheet technology (Microsoft's *Excel*) and the TEX Live Windows implementation of Hàn Thế Thành's *Pdf(LA)TEX* by Fabrice Popineau.

The Work of the Scholars

Although locating and obtaining copies of the manuscripts requires a not-insignificant amount of time, I will concentrate here on the tasks which the scholars undertake once the copies have been received. Each scribal notation that is potentially of interest is identified and scanned, and any artifacts that might serve to confuse are eliminated using a light pen and suitable software (JASC's *Paintshop PRO*). The resulting "clean" image is then stored as a PDF file using a fixed naming convention, and a corresponding entry made in an *Excel* spreadsheet: this entry contains the filename, the transcription, an explanation (if the notation is an abbreviation or similar) and the provenance (typically the date, but occasionally a more detailed provenance where this is felt to be important). Lest this create the impression that the rôle of the scholars is trivial, let me emphasise that the task of deciphering and interpreting the scribal notations is one requiring much skill and many many years of experience!

Scanning and transcription take place in batches, following which proofing takes place. During the proofing stage, the size and relative position of each scanned image is adjusted to ensure that all scanned images reproduce at approximately the same height and with the same vertical offset from the notional baseline. As will be seen later, ensuring that all images reproduce at the same height has only a limited effect on their *widths*: as a result, a statistical analysis of the widths of the scanned images will later be needed to allow an accurate assessment of the proportion of images that would fit without problem were a particular book design to be adopted. The scaling and offset parameters are stored within the *Excel* spreadsheet.

As far as is possible, syntax errors are identified and corrected at the proofing stage, although some may sneak through and require further intervention at the galley or page-proof stages. Because of the complexity of the markup used to represent transcriptions and explanations, such errors can easily creep in — these frequently involve braces, either as mismatched pairs or through the accidental use of non-brace symbols such as parentheses or brackets.

Markup and Syntax

Each entry in the spreadsheet consists of a number of fields, most of which are destined to become incorporated in a TEX document. Each record in the resulting TEX file conforms to the following pattern:

```
\Byzantine
    <scale>
        <y-offset>
            <filename>
                <transcription>
                    <explanation>
                        <provenance>
```

of which an example might read

```
\Byzantine
    -0.2
        -0.5
            abr-qwrafion2
                {{q\raise {f}}}
                    {{qwr'afion}}
                            {{1430}}
```

The `\Byzantine` command is used to introduce each record, and its meaning is redefined in various programs used to process the data. The first two parameters represent the scaling to be used (a negative value implies shrinkage rather than magnification) and the offset from the horizontal axis (a negative value implies lowering rather than raising). The third parameter is the filename, the portion preceding the hyphen indicating into which of about ten general classifications the record should be subsumed. The fourth parameter is the transcription, marked up according to Silvio Levy's encoding scheme for polytonic Greek with additional commands required to indicate scribal ornamentations such as `\raise {}`, `\overbar {}`, etc. The fifth parameter is the explanation, again marked up using Levy's scheme (this time with no extensions); and the sixth and final field is the provenance.

Sorting the Data

Although the data are coarsely pre-sorted by virtue of the prefix element of the filename, the actual lexicographic sorting needed before the data can be incorporated in the final lexicon is *considerably* more complex. Not only is it necessary to sort — by Greek collating rules — the latin transliteration of the Greek characters originally used, it is also necessary to ensure that the sorting takes into account all of the additional orthographic devices which may occur: breathings, accents, iota subscripts, ornaments (raised [groups of] letters, overbars), diareses and of course case-differences themselves. Furthermore, it is necessary to sort initially by transcription, but if — for a given record — the transliteration is absent, or identical to another transcription, then sorting must instead be by explanation, and if two or more records are found *still* to be identical after all of these criteria have been considered, then the date (provenance) and finally the original order in the spreadsheet must be taken into account (this last fallback key ensures that no manual sorting will ever be required once the data have been correctly entered in the spreadsheet).

Needless to say, sorting of this complexity is a task for which TEX is rather less than ideally suited. Even though earlier workers (e.g., Kees van der Laan, 1993; Bernd Raichle, 1994) have shewn that TEX is perfectly *capable* of performing sorting, the magnitude of the data (some 4000 records) and the complexity of the sorting required suggest that a more appropriate tool be used. The problem, however, is that the TEX markup used is fairly complex, and in order to parse it effectively, TEX itself is really required. Thus we appeared to be on the horns of a dilemma: on the one hand, TEX was felt to be unsuitable for the task of sorting, yet on the other TEX was considered to be essential if the markup were to be correctly parsed and interpreted. This *impasse* was finally resolved during discussions at a EuroTEX conference with Professor Klaus Lagally, who had experience of similar problems when trying to sort Arabic text in TEX : his solution, which proved to be absolutely ideal, was that we should treat the task as two separate problems — (1) parsing the TEX markup, and (2) sorting the data. The key to the solution lay in his suggestion that, during the sorting phase, TEX be asked to output far simpler keys (e.g., purely numeric) which could then be easily interpreted by any conventional sorting routine.

Parsing in TEX

With Klaus's suggestions firmly in mind, work started on writing the TEX parser. Although it would have been possible to write all keys to a single file, it was decided to associate each key with a unique file :

```
\immediate \openout  \TRAccents
    = Transcription.accents
    . . .
\immediate \openout  \TROrnaments
    = Explanation.ornaments

\immediate \openout  \EXAccents
    = Transcription.accents
    . . .
\immediate \openout  \EXOrnaments
    = Explanation.ornaments

\immediate \openout  \Dates
    = Byz-data.dates
\immediate \openout  \Sequence
    = Byz-data.sequence
```

Note that accents, breathings, cases, diareses, iotas, letters and ornaments are replicated for transcription and explanation but that dates and sequence numbers require only a single instance of each.

The main loop of the program iterates over its input file :

```
\loop
    \read \source to \buffer
    \ifeof \source
        \repeatfalse
    \else
        \expandafter
                \parse \buffer \endparse
        \advance \entry by 1
        \repeattrue
    \fi
\ifrepeat
\repeat
```

Before parsing the transcription and the explanation, we re-define the output files as being `\TR...` or `\EX...` as appropriate :

```
\def \Usetranscription
    {%
     \let \Accents = \TRAccents
      . . .
     \let \Ornaments = \TROrnaments
    }

\def \Useexplanation
    {%
     \let \Accents = \EXAccents
      . . .
     \let \Ornaments = \EXOrnaments
    }
```

Remembering that each input record consists of the control word `\Byzantine` followed by six parameters, we define `\parse` to call the analysis routine twice, passing first the transcription and then the explanation :

```
\def \parse \Byzantine
            #1 #2 #3-#4 #5#6#7\endparse
    {%
     \reset
     \transcription = {#5}
     \Usetranscription
     \analyse #5\endparse \endanalyse
     \print

     \reset
     \Useexplanation
     \explanation = {#6}
     \analyse #6\endparse \endanalyse
     \print
    }
```

The apparent difference between the earlier statement that the transcription forms the fourth parameter (and the explanation the fifth parameter) and the code above, which appears to refer to them as the fifth and sixth parameters respectively, is explained by the fact that during parsing we treat the filename as two separate parameters separated by a hyphen : this allows the prefix (representing the general category into which the entry fits) to be extracted and used to qualify the date, since the lexicon is macro-ordered by general category, and micro-ordered by the sorting criteria currently being described.

The analysis code itself is fairly straightforward :

```
\def \analyse #1#2\endanalyse
    {\ifx #1\endparse
     \else
```

```
        \def \flag {#2}
        \advance \index by 1
        \process {#1}
        \analyse #2\endanalyse
     \fi
    }
```

all of the complexity being delegated to the `\process {}` routine:

```
\def \process #1%
    {
        \csname +\string #1\endcsname
    }
```

Before `\process {}` can be understood, it is first necessary to explain how the parser identifies into which category each token fits. The program starts by listing the various categories:

```
\newentity {accent}
\newentity {breathing}
\newentity {diaresis}
\newentity {grouping}
\newentity {iota}
\newentity {letter}
\newentity {ornamentation}
```

Then, for each category, the tokens which compose that category are enumerated. Here, for example, the possible accents are enumerated:

```
\newaccent `
\newaccent '
\newaccent ~
```

To avoid the risk of human error, we interrogate an internal counter to find how many elements there are in each category:

```
\numberof \accents = \valueof ~
\advance \accents by 1 %%% null accent
```

Now take a deep breath, because we need to discuss how `\newentity {}` is defined:

```
 1 \def \newentity #1
 2   {\expandafter \NewCount
 3                    \csname #1\endcsname
 4    \expandafter \def \csname
 5                    new#1\endcsname ##1%
 6     {\advance \csname #1\endcsname by 1
 7      \expandafter \edef \csname
 8                   +\string ##1\endcsname
 9        {\expandafter \noexpand
10           \csname #1token\endcsname
11            {\string ##1}
12            {\the \csname #1\endcsname}%
14        }
15     }
16   }
```

As this code is somewhat opaque, let's make life simpler by considering what happens when `\newentity {}` is called with a parameter, as in `\newentity {accent}`. Lines 2 to 3 expand to yield `\NewCount \accent`. `\NewCount` can be used in macro expansions but is otherwise identical to Plain's `\newcount`. Lines 4 onwards expand to yield a definition for the single-parameter macro `\newaccent {}`; the definition is equivalent to the following pseudo-TEX code:

```
\def \newaccent #1
    {\advance \accent by 1
     \edef \+#1%
        {\accenttoken {#1}{\the \accent}
    }
```

Again it will be simpler to understand through the medium of an example, so we will consider what happens when `\newaccent {}` is called with parameter `~`, as in `\newaccent ~`. The counter `\accent` is incremented by one (it starts life at zero), and the control sequence `\+~` is defined to expand to `\accenttoken {~}{<current value of \accent>}`. The sole function of the `+` prefix used in constructing the name of the control sequence is to reduce the risk of a namespace clash.

We will next need to look at `\accenttoken {}`, which as we see below is just one of a family of identically treated control sequences:

```
\def \accenttoken #1#2%
                {\do {Accent}{#1}{#2}}
\def \breathingtoken #1#2%
                {\do {Breathing}{#1}{#2}}
\def \diaresistoken #1#2%
                {\do {Diaresis}{#1}{#2}}
\def \groupingtoken #1#2%
                {\do {Grouping}{#1}{#2}}
\def \iotatoken #1#2%
                {\do {Iota}{#1}{#2}}
\def \lettertoken #1#2%
                {\do {Letter}{#1}{#2}}
\def \ornamentationtoken #1#2%
                {\do {Ornament}{#1}{#2}}
```

after which we need to examine `\do {}`:

```
\def \do #1#2#3%
    {%
     \csname #1\endcsname {#2} {#3}
    }
```

Since parameters 1 & 2 of `\accenttoken {}` become parameters 2 & 3 of `\do {}`, we can see that when `\do {}` is launched from `\accenttoken {}` the expansion is:

```
\Accent {<accent-token>}{<numeric-value>}
```

Remembering that TeX is case-sensitive, it should be clear that `\accent` and `\Accent {}` are totally different entities — the former is an integer register (declared with `\NewCount {}`), whilst the latter is explained below:

```
\def \Accent #1#2%
   {
   \lastaccent = #2
   }
```

Thus the sole effect of `\Accent {}` is to store the numeric value associated with the accent (its ordinal) in `\lastaccent`; most of the entities analogous to `\Accent {}` behave in a similar way, with the key exception of `\Letter {}` at which we must next look:

```
 1 \def \Letter #1#2%
 2     {%
 3      \lettervalue = #2
 4      \ifodd \lettervalue
 5         \edef \Caseskey {\Caseskey 1}
 6         \advance \lettervalue by 1
 7      \else
 8         \edef \Caseskey {\Caseskey 0}
 9      \fi
10      \edef \Accentskey
11         {\Accentskey \the \lastaccent}
12      \edef \Breathingskey
13        {\Breathingskey
14                  \the \lastbreathing}
15      \edef \Diareseskey
16        {\Diareseskey
17                  \the \lastdiaresis}
18      \edef \Iotaskey
19         {\Iotaskey 0}
20      \edef \Letterskey
21         {\Letterskey \expandafter
22              \expandafter \expandafter
23                   \twodigits \expandafter
24                        0\the \lettervalue
25                             \sentinel
26         }
27      \edef \Ornamentskey
28         {\Ornamentskey
29                   \the \lastornament}
30      \lastaccent = 0
31      \lastbreathing = 0
32      \lastdiaresis = 0
33     }
```

It is, in fact `\Letter {}` (which is triggered by the parser detecting a letter, as opposed to any diacritic or other orthographic mark) that is at the heart of the TeX parser under discussion. Remember that `\Letter {}` will be called with the actual letter as parameter 1 and the ordinal of that letter as parameter 2. At line 3, the ordinal is saved in `\lettervalue`. At lines 4 to 9, a test is made to see whether this is odd or even, a simple test which discriminates between upper- and lower-case letters. If the result is odd (uppercase), `\lettervalue` is rounded upwards to renormalise it as lowercase after noting the fact that it was originally uppercase. Note carefully the `\edef`s at lines 5 & 9, which append a zero or a one to the current value of `\Caseskey`: this same mechanism is used from lines 10 to 29 to append the last (accent, breathing, diaresis, or ornament) ordinal to the corresponding key.

Here at last we begin to see the results of all of our efforts: the various keys are extended (by a fixed amount) each time a letter is encountered in the input record to capture, as a set of sequences of fixed-length integers, the possible features which may be used to differentiate each letter from an otherwise identical letter when sorting finally takes place.

After parsing the transcription, and again after parsing the explanation, we write each of the seven keys to the associated file:

```
\def \print
    {
     \immediate \write \Accents
        {\Accentskey}
         ...
     \immediate \write \Ornaments
        {\Ornamentskey}
    }
```

After the keys for the transcription and explanation have been written to file, the date (with filename prefix prepended) and sequence number are similarly recorded:

```
\immediate \write \Dates {#3-#7}
\immediate \write \Sequence {\the \entry}
```

The Results

The end result of all of this is a series of files, each of which consist of n records, where n is the number of records in the original data set. Each record in each file will be of length $K \times l$, where l is the number of letters in the corresponding input record and K is a constant which varies from file to file (some keys can be represented as a single digit per character, some require two digits per character, and so on). A short fragment of a typical input file, and the corresponding extracts from sample key files, are shewn on the following pages; the samples are intended to illustrate most of the scribal devices used.

byz-data.dat

```
\Byzantine -0.5 0.4 abr-adelfou
    {{>ad\raise {e}}} {{>adelfo~u}}
        {{Thebes}}
\Byzantine -0.4 0.4 abr-adelfous
    {{>ad\raise {o‘us}}} {{>adelfo‘us}}
        {{1374}}
\Byzantine -0.3 0 abr-adelfwn
    {{>adelf}} {{>adelf~wn}}
        {{1374}}
\Byzantine -0.3 -0.2 abr-aer
    {{a\raise {e}r}} {{>’aer}}
        {{15\th c.}}
\Byzantine -0.2 -0.2 abr-aer1
    {{a\raise {e}r}} {{>’aer}}
        {{1492}}
\Byzantine -0.3 -0.5 abr-afierwthrion
    {{>af"I\raise {e}rw\raise {tr}}}
        {{>afierwt’hrion}} {{1420}}
\Byzantine -0.3 -0.2
    abr-afrodith-fwsforos-qalkos {{}}
        {{>Afrod’ith fwsf’oros / qalk‘os}}
            {{16\th c.}}
```

transcription.letters

```
041012
0410324638
0410122448
041236
041236
0448201236544436
<blank>
```

transcription.accents

```
000
00010
00000
000
000
00000000
<blank>
```

transcription.breathings

```
100
10000
10000
000
000
10000000
<blank>
```

transcription.cases

```
000
00000
00000
000
000
00100000
<blank>
```

transcription.diareses

```
000
00000
00000
000
000
00100000
<blank>
```

transcription.ornaments

```
001
00111
00000
010
010
00010011
<blank>
```

explanation.letters

```
04101224483246
0410122448324638
04101224485428
041236
041236
0448201236544441636203228
0448363210204416485438483236323850042422...
```

explanation.accents

```
0000003
00000010
0000030
200
200
000000020000
0000020000002000000010
```

explanation.breathings

```
1000000
10000000
1000000
100
100
100000000000
1000000000000000000000
```

explanation.cases

```
0000000
00000000
0000000
000
```

```
000
000000000000
1000000000000000000000
```

explanation.diareses

```
0000000
00000000
0000000
000
000
000000000000
0000000000000000000000
```

explanation.ornaments

```
0000000
00000000
0000000
000
000
000000000000
0000000000000000000000
```

Sorting in Perl

After the complexities of the TEX coding required to implement the parser, the matching Perl code will come as something of a relief! We start by opening or creating a few files:

```
$Data = "Byz-Data.dat" ;
open Data or die
    "File $Data cannot be opened :
        $ !\n" ;
@data = <Data> ;

$TRAccents = "Transcription.Accents" ;
open TRAccents or die
    "File $TRAccents cannot be opened :
        $ !\n" ;
@TRaccents = <TRAccents> ;

. . .

$EXOrnaments = "Explanation.Ornaments" ;
open EXOrnaments or die
    "File $EXOrnaments cannot be opened :
        $ !\n" ;
@EXornaments = <EXOrnaments> ;

$Dates = "Byz-Data.dates" ;
open Dates or die
    "File $Dates cannot be opened :
        $ !\n" ;
@dates = <Dates> ;

$Sequence = "Byz-Data.Sequence" ;
open Sequence or die
    "File $Sequence cannot be opened :
        $ !\n" ;
@sequence = <Sequence> ;

$Sink = ">Byz-Data.Srt" ;
open Sink or die
    "File $Sink cannot be created :
        $ !\n" ;
```

We then enumerate the keys that will be used for sorting:

```
@key11 = @TRletters ;
@key12 = @TRbreathings ;
@key13 = @TRaccents ;
@key14 = @TRiotas ;
@key15 = @TRornaments ;
@key16 = @TRdiareses ;
@key17 = @TRcases ;

@key21 = @EXletters ;
@key22 = @EXbreathings ;
@key23 = @EXaccents ;
@key24 = @EXiotas ;
@key25 = @EXornaments ;
@key26 = @EXdiareses ;
@key27 = @EXcases ;

@key31 = @dates ;
```

Then we perform a detached key sort and output the results:

```
@keys = sort polytonically @sequence ;
foreach $key (@keys)
            {print Sink $data [$key]} ;
```

All that remains is to define the comparison algorithm:

```
sub polytonically
    {if     (($key11 [$a]
                cmp $key11 [$b]) != 0)
                    {return $key11 [$a]
                        cmp $key11 [$b]}
     elsif  (($key12 [$a]
                cmp $key12 [$b]) != 0)
                    {return $key12 [$a]
                        cmp $key12 [$b]}
     . . .

     elsif  (($key27 [$a]
                cmp $key27 [$b]) != 0)
                    {return $key27 [$a]
                        cmp $key27 [$b]}
     elsif  (($key31 [$a]
                cmp $key31 [$b]) != 0)
                    {return $key31 [$a]
```

```
                      cmp $key31 [$b]}
    else   {print Errors "Warning :
                 duplicate entry : \n",
                   $sequence [$a]+1,
                      " : ", $data [$a],
                        $sequence [$b]+1,
                            " : ", $data
                                [$b], "\n"}
   }
```

Statistical Analysis of Field Widths

As explained above, although the scanned images are normalised for height and position, it is impossible to normalise them for width since some are inherently narrow and others are inherently wide. The transcriptions, explanations and provenances, too, vary widely in length. In order to gain an insight into the best distribution of the available space between the various fields (and, indeed, in order to determine the minimum page size which would accommodate the longest possible entry for one-, two- and three-column designs), it was decided to perform a statistical analysis of the variation in minimum width of each of the fields. For the scanned images, all that was necessary was to record the width of each (after scaling) and to use *Excel* to analyse these (the actual analysis techniques used are discussed below), but for the textual fields there was an additional and very interesting problem : how does one decide what is the minimum width that can be used to typeset a given stretch of text ?

Obtaining the Statistics for Text Fields

One possible approach is to typeset the text in a `\vbox {}` with `\hsize = 0pt`. This will forcibly hyphenate every possible word, but the problem is knowing how to access the results : if one examines the dimensions of the `\vbox {}` after this operation, it will have finite height and depth, but the width will still be `0pt`, and even if one unboxes it and reboxes it, one finds that the internal `\hbox {}`es that TEX constructs whilst paragraph building also have zero width. However, all is not lost : if one uses TEX's box destructor primitive `\lastbox`, one can gain access to the last line of the paragraph ; unboxing and reboxing this line yields an `\hbox {}` with finite width as well as finite height and depth. Applying this technique iteratively allows access to (the widths of) all the lines of the paragraph, and if one then selects the largest of these, one then knows the narrowest measure within which the stretch of text could be typeset.

In reality, of course, this may be far too narrow to be usable, but once one has established a lower bound one is as least on the way to determining the optimal width.

Sample code which can be used to perform this operation is shewn below :

```
\toks 0 = {Research by the School of
           Management is not confined
           to for-profit corporations,
           it is also a leading centre
           for research into public
           sector organisations.  In
           recognition of the rising
           impact of sustainability for
           business and society, the
           School, together with the
           Department of Geography,
           have created an
           inter-disciplinary Centre for
           Research into Sustainability
           (CRIS).
           }

\newif \ifloop
\newdimen \minwidth

\def \findminwidth #1#2%
    {%
     \minwidth = 0 pt
     \setbox 0 =
     \vtop \bgroup
           \hsize = 0 pt
           \hfuzz = \maxdimen
           #1 \noindent #2 \par
           \loop
               \setbox 2 = \lastbox
               \ifvoid 2
                   \loopfalse
               \else
                   \setbox 4 = \hbox
                           {\unhbox 2}
                   \ifdim \wd 4 > \minwidth
                       \global \minwidth
                                    = \wd 4
                   \fi
                   \unskip
                   \unpenalty
                   \looptrue
               \fi
           \ifloop
           \repeat
           \egroup
     \message {min-width : \the \minwidth}
    \global \setbox 0 = \vtop
            \bgroup
            \hsize = \minwidth
            \rightskip = 0 pt
               plus \minwidth
            #1 \noindent #2 \par
            \egroup
   }

\findminwidth {\uchyph = 1}{\the \toks 0}
\setbox 2 = \box 0
\findminwidth {\uchyph = 0}{\the \toks 0}
\setbox 4 = \box 0
\findminwidth {\hyphenchar \font = -1 }
                              {\the \toks 0}
\setbox 6 = \box 0
\leftline
      {\box 2 \quad \box 4 \quad \box 6}
\end
```

The test lines at the end demonstrate that different minimum widths can be achieved depending on whether or not hyphenation is permitted, and if permitted, whether upper-case words may legitimately be hyphenated. For the sample stretch of text used, the three minima were `50.36124pt`, `60.05573pt` and `74.00015pt` for the uc & lc case, the lc-only case, and the no-hyph case respectively. Figure 2 shews the results of typesetting the sample text to these three measures:

Research	Research by	Research by
by the	the School of	the School of
School of	Management	Management
Manage-	is not	is not confined
ment is not	confined to	to for-profit
confined to	for-profit	corporations,
for-profit	corporations,	it is also a
corpora-	it is also	leading centre
tions, it	a leading	for research into
is also a	centre for	public sector
leading	research into	organisations. In
centre for	public sector	recognition of
research	organisa-	the rising impact
into public	tions. In	of sustainability
sector	recognition	for business
organisa-	of the rising	and society,

Figure 2: Typesetting to the narrowest measure: uc & lc hyphenation, lc-only, and no hyphenation

Analysing the Statistics

For many years, I eschewed spreadsheets completely, believing that they were yet another manifestation of "The Emperor's New Clothes" and offering nothing whilst promising everything ... It was only when I started chairing the TUG Bursary Committee that I began to realise that spreadsheets did indeed have something to offer, and so when I began to search for a tool to help with the analysis of field widths for the current project I started by looking at the possibilities of *Excel*.

At first I was defeated by little things: *Excel*, for example, seems unfamiliar with the concept of the point as a unit of measure, so it was unable to deal with TeX's `58.88902pt` notation. This was very easily dealt with once I realised that *Excel*'s `Data/Text to Columns.../Delimited/Other/"p"` would do exactly what I needed — strip off the trailing `pt` leaving only the unitless number in the source column.

The second task was considerably more difficult: given a set of some 4000 real numbers (representing the widths of one of the fields in the Lexicon), I wanted to (a) sort them, and (b) derive statistics which would shew what percentage were less than each unique value. I was convinced that *Excel* could manage this, but none of my *Excel*-literate colleagues (including my wife!) could tell me how to persuade *Excel* to do the necessary.

In the end, a Google search led me to the solution: the necessary statistical tools are *not* installed by default, and it is first necessary to install the appropriate options pack. `Tools/Add-Ins.../Analysis ToolPak` proved to be the required incantation, after which `Tools/Data Analysis.../Histogram/Cumulative Percentage` provided the exact statistics that I needed. Figures 3 – 4 shew a sample of the output from *Excel* covering the range from 85% to 98,5%. Access to statistics such as these for each of the four fields of the Lexicon will prove invaluable when putting the finishing touches to the book design, since the authors will be able to see for themselves what fraction of the entries would fit without compromise were a particular design to be selected.

Conclusions

TeX is a superb program, capable of producing the finest quality typeset output; however, it is neither the ideal tool for sorting, nor for producing statistical analyses. When used in conjunction with other tools such as Perl (sorting) and *Excel* (statistics), the combined power far exceeds the sum of the parts. Synergies such as these are surely the key to the rôle of TeX in the future: TeX should no longer be perceived as a tool in isolation, but rather as a partner in a whole suite of tools, each perfectly adapted to the task for which it is used.

43.51428	31	83.30%
44.06853	92	85.42%
44.62279	16	85.79%
45.17705	18	86.21%
45.73131	10	86.44%
46.28556	25	87.02%
46.83982	19	87.46%
47.39408	22	87.96%
47.94834	18	88.38%
48.50260	24	88.94%
49.05685	20	89.40%
49.61111	19	89.84%
50.16537	15	90.18%
50.71963	8	90.37%
51.27389	13	90.67%
51.82814	34	91.45%
52.38240	13	91.75%
52.93666	14	92.08%
53.49092	12	92.35%
54.04517	19	92.79%
54.59943	33	93.56%
55.15369	8	93.74%
55.70795	9	93.95%
56.26221	12	94.22%
56.81646	6	94.36%

Figure 3: Width, frequency & cumulative %age

57.37072	8	94.55%
57.92498	15	94.89%
58.47924	7	95.06%
59.03349	8	95.24%
59.58775	1	95.26%
60.14201	7	95.43%
60.69627	34	96.21%
61.25053	4	96.30%
61.80478	3	96.37%
62.35904	4	96.47%
62.91330	6	96.60%
63.46756	7	96.77%
64.02181	12	97.04%
64.57607	2	97.09%
65.13033	4	97.18%
65.68459	4	97.27%
66.23885	21	97.76%
66.79310	4	97.85%
67.34736	3	97.92%
67.90162	7	98.08%
68.45588	4	98.18%
69.01014	8	98.36%
69.56439	3	98.43%
70.11865	3	98.50%
70.67291	1	98.52%

Figure 4: Width, frequency & cum. %age (cont)

Acknowledgements

The author would like to acknowledge the unwavering and unstinting affection, dedication and patience of Miss Julian Chrysostomides and Dr Charalambos Dendrinos, with whom it is truly a joy to work. He would also like to thank Professor Klaus Lagally for his advice on sorting non-Latin scripts, and Claudio Beccari for his courtesy and patience whilst the author was trying to effect the transition from Silvio Levy's polytonic Greek fonts to the "CB" series. Finally he would like to thank the TUG Bursary Committee for financial support which enabled him to present this paper at Wu Han (China) during the TUG 2005 conference.

Bibliography

van der Laan, C.G. ("Kees") 1993: *Sorting in BLUe*; Minutes and Appendices (MAPS) 10, Nederlandstalige TEX Gebruikersgroep (NTG).

Raichle, Bernd 1994: *Sorting in TEX's Mouth*; Proceedings of the 1994 EuroTEX Conference.