BLUe's Format Databases

Stay organized

Kees van der Laan

Hunzeweg 57,
9893 PB Garnwerd, The Netherlands
cgl@rc.service.rug.nl

Abstract

The backgrounds, use, and coding of BLUe's Format Databases have been discussed. Two kinds of databases have been introduced class I (data, such as addresses, references, and script parts for example pictures), and class II (macros, such as variant formats and tools).
At the heart lies the selective loading TEXnique, which allows that only what is needed will be loaded on-the-fly. The data structures and operations on them have been treated. The use within the context of typesetting scripts have been elucidated via examples. At the end the coding is explained.

Keywords: Active list separators, addresses, comment blocks, compatible extension, data integrity, databases, education, fifo, lazy evaluation, list element tag, macro writing, mail-merge, no-nonsense, number ranges, pattern matching, pictures, plain TEX, references, reusable software parts, search, selective loading, set macros, software engineering, table of contents, variant document parts.

1 Introduction

Why couple the buzzword database to TEX? What has TEX got to do with it? Vaguely the answer is that we like to store addresses, references, copy parts, tools and formats outside of TEX—or one of its flavours—and only borrow what we need. Important is also the data integrity aspects which can be achieved via databases. We only store the information once for use in different contexts. At the heart is the process of selective loading.

Let us go back to Knuth and survey how he coped with using macros (formats and tools), addresses, references, and copy parts.

With respect to formats Knuth provided and discussed: concert format, letter format, and *The TEXbook* format, i.e., the manmac collection of macros. How did he store those macros? The answer is: simply in separate files. Is that good enough? The answer is: no, IMHO, with all respect, assumed we only need part of it. Similarly, he used gkpmac, as an independent collection.

How to handle macros has been treated in *The TEXbook* Appendix D *4. Selective loading of macros.* For references he mentioned at the end of that section '. . . to prepare a bibliography for a paper, by reading a suitably arranged bibliography file; only the entries that correspond to defined control sequences will be loaded.' No further details did he supply for how he coped with a database of addresses, or (prefab) parts. However, in manmac Knuth elaborated on how to handle the document parts

- answers, and
- index reminders, IRs for short.

Although the answers and IRs are very loosely connected to the database idea I'll survey Knuth's approach nevertheless, because they give insight how to cope with copy parts, especially how to create such files and how to process them efficiently. This elucidates the use of Knuth's \\, the what I call list element tag.

1.1 Answer elements

in the file answers obey the syntax

```
<answerelement> is
   \ansno <chapnumber>.<exnumber>
   <answerproper>
```

Example *(An answer, what is the question :-))*

```
\ansno 3.16:
The Bible Illuminated
```

The file answers contains separator lines which begin with \ansno—the list element tag—and contain the separators period and colon. Very close to natural markup. Only \ansno is extra. Apparently we can't do without something like that.

In an abstract sense the collection of answers forms a list. Lists in TEX are processed via some sort of active list separator, as explained in Appendix D of *The TEXbook*. How did Knuth cope with the list of answers[1]? For this case he used \ansno as active list 'separator,' i.e., \ansno processes each answer. Perhaps we should call this tag the *list element tag*, because it marks (the start of) a list element.

[1] Let us assume for simplicity that the answers are already stored in the file answers.

The name active list separator is not accurate enough, although conceptually it comes close. It is just the first element which is inconsistent, the element is preceded by the tag. That tag separates partially, let us say.

1.2 Index reminders

in the file index obey the syntax `<IR> is`

```
<indexelement> !<classificationnumber>
    <pagenumber>.
```

Knuth did not process this file within TEX. Ultimately, he set the sorted and enriched index file via TEX, of course.[2]

1.3 And what about addresses?

A letter is provided with the address included, no use of databases, see *The TEXbook* 404, 405.

1.4 Knuth's `\ifoption`

This is about optional inclusion of macros, and in general document parts, as mentioned in the infamous Appendix D of *The TEXbook*. Assume that a file macs has been created with as structure a sequence of `\ifoption`s.

```
\ifoption A <Apart>\fi
\ifoption B <Bpart>\fi
\ifoption C <Cpart>\fi
%et cetera
```

Knuth's selective loading comes down to

- the user supplies for example `\load{macs}{AC}`
- create the list `\options` with for example as replacement text `\\A\\C`, with `\\` the function to process the next token as argument (done by `\fifo`...`\ofif` below)
- define `\ifoption`, and therein `\\`, such that the appropriate parts of the file, macs let us say, will be loaded.

This is achieved by the following from *The TEXbook* 384, transcribed into my fifo notation

```
\def\load#1#2{%#1 file, #2 options
%Purpose: #2 elaborated into list of options
%          as replacement text of \options.
   \let\options\empty\let\\\relax\fifo#2\ofif
   \input#1 }
\def\fifo#1{\ifx#1\ofif\ofif\fi
   \process#1\fifo}        \def\ofif#1\fifo{\fi}
\def\process#1{\edef\options{\options\\#1}}
%
\def\ifoption#1{%Purpose: to locate #1
%and to load the corresponding part.
  \def\\##1{\if##1#1\resulttrue\fi}%
  \resultfalse \options \ifresult}
%with driver, for example
\load{macs}{AC}
%and the file macs with for example
\ifoption A The A part\fi
\ifoption B The B part\fi
\ifoption C The C part\fi
\endinput
```

My approach differs in that I don't have two list element tags, such as `\ifoption` and `\\`, but only `\lst`, respectively `\tool`. My A, B etc. are names, control sequences, which can be initialized with an error message, with the advantage that this message will appear when the name does not match the name of the document element to be loaded. Furthermore, I used a token variable `\namelst` instead of the definition `\options`, to store the list of options, but that is not essential.

My begin and end of the optional part are braces to facilitate using it as a replacement text. (With tools and formats I don't have braces but terminate via `\endinput`.) Knuth's begin and end tags are `\ifoption` and `\fi`.

I won't detail further but will explain the superior method of Díaz.

1.5 Díaz fast selective loading

The Díaz process, mentioned in *The TEXbook* 384 as a superior method, reads essentially as follows.[3]

```
%The TeXbook, Appendix D 4.Selective loading.
%The Max Diaz fast selective loading process.
%(A little simplified, and combined with
% my list element tag, \lst.)
\def\lst#1{\catcode`\~=
   \ifx#1\undefined14    %comment
              \else9 \fi}%ignore char
%We want to load the cgl part.
\def\cgl{<anything>}

\lst\name
~a
~b
~c
\lst\cgl
~aa
~bb
~cc
\lst\erik
~aaa
~bbb
~ccc
\catcode`\~13
%
<Commonpart>
\bye
%Explanation: the list element tag toggles the
%catcode for ~ such that either the first
%character is ignored (and the rest of the line
%loaded) or the line is a comment line.
```

In the above one can replace `\lst\name...` by `\input <filetobeloadedfrom>`.

I did not elaborate much on Max Díaz approach via catcode changes, because I don't like as yet to insert alien first line characters in the document parts stored in the file to be selectively loaded from.[4]

[2] How to handle (modest) indexes completely within TEX has been treated in 'BLUe's Index.'

[3] Courtesy Erik van Eynde for posing the practical problem of variant document parts on the TEX-nl discussion list, which stimulated me to look closer to Knuth's and Diaz' approach.

[4] I do realize the gains in efficiency though, and more importantly there are less restrictions. Perhaps I'll use it in the next release of BLUe's Format System.

Loosely related to the above is the use of the so-called block comments, to treat parts of documents as comments. What I have seen and worked on myself so far is similar to verbatims except that it is not set. I rate Diaz approach as superior and won't elaborate on block comment further.

1.6 Notations and definitions

gkpmac stands for the macros used by Graham, Knuth, Patashnik to format Concrete Mathematics.

manmac denotes the macros used by Knuth to format *The TEXbook*, and the METAFONTbook.

Pascal denotes Wirth's programming language.

When I load from a database file the extension is already provided for and reads '.dat'.

2 Databases

With algorithms,[5] and similar with databases, we have two main aspects

- data structures, and
- operations on the data.

Up till now I have used the following two kinds of databases to cooperate smoothly with TEX.

I data, such as, addresses, references, and script parts
II tools, such as, variant formats, and various macros.

I call these class I and class II databases. Samelson's principle has been obeyed

'Don't pay for what you don't use,'

which comes down to selective loading.

The difference between class I and class II is that in class II only one element has to be loaded per scan.

Each file consists of a list of `<listelement>`s. The file names obey the syntax `<subjectindicator>`.dat. I consider it useful to attach a name to each list element.

Below I'll discuss the elements proper, deprived from the useful extras such as author information, markup for selective line numbering, contents, and history of changes, in order to convey the main idea.

2.1 Class I data structure

At the moment this is used in the files

- address.dat, for addresses
- lit.dat, for references, and
- pic.dat, for pictures.

Each list element is a triple and reads as follows.

- list element tag, `\lst`
- name of list element
- the list element proper, as group. that is enclosed by curly braces.

I chose `\lst` as the tag for list element tag. One could equally well have chosen another control symbol or sequence as list element tag, for example Knuth's `\\`. However, because `\\` is so heavily used for e-o-l I refrained from that.

Within the replacement text I adopted application specific conventions. For example with addresses I used `\\`—another lower level list element tag :-)—to facilitate formatting. I also used elements tagged by `\email`, and `\phone`. When the entry is used for an address label the latter tags are `\let`-equal to `\gobble`, i.e., they are neglected for the label.

For addresses the syntax of an entry reads

```
\lst\<name>{<salutename>
\\<fullname>
\\<affiliation>
\email{...}
\phone{...}
\fax{...}
}
```

Example *(Entry for address.dat)*

```
\lst\ntg{NTG
\\Nederlandstalige \TeX{} Gebruikersgroep
\\Postbus 394
\\1740 AJ Schagen
\\The Netherlands
\email{ntg@nic.surfnet.nl}
}
```

For references the replacement text starts with the name of the author with as few punctuation marks as possible, followed by the date, title, and information about the source. I also decided to include `\annotation`, which can gobble its argument when not needed, or anything you want. In the simplest case it can return 'its argument' via `\let\annotation\relax`.

Example *(Entry for references.dat)*

```
\lst\knuthded{Knuth D.E (1984):
   Computers and Typesetting.
   \TB. Addison-Wesley.
   ISBN 0-201-13447-0 (hard cover)
   ISBN 0-201-13448-9 (soft cover).
   \annotation{For the correct printing
    look in the index for \cs{language}.
    \TeX, is frozen in version $\pi$,
    3.1415\dots}
}
```

For entries of the picture database see 'BLUe's Graphs,' the ideas are similar.

The restrictions

are that as part of the replacement text tags are excluded which are not allowed in an argument of a `\def`.

2.2 Class II data structure

At the moment this is used in the files

- fmt.dat, for variant formats, and

[5] Courtesy Niklaus Wirth 'Algorithms = data + programs.'

- tools.dat, for tools, such as the index macros.

To facilitate handling I consider it useful to close each element by `\endinput`. As a result each list element is a quadruple

- list element tag, `\tool`
- name of list element
- the list element proper
- `\endinput`.

I chose `\tool` as the tag for list element tag. One could equally well have chosen another control symbol or sequence as list element tag, for example Knuth's `\\`. However, because `\\` is so heavily used for e-o-l I refrained from that.

The list element tag name ends either with tool or fmt. Within the replacemenet text no conventions are prescribed. The syntax for an entry reads `\tool\<name>tool`

```
\tool\<name>fmt
   <toolmacros>    or        <formatmacros>
\endinput                \endinput
```

Example *(Entry for tools.dat)*

```
\tool\englishtool
\message{ ---English , Jan 95, cgl--- }
\abstractname{Abstract}
\acknowledgementsname{Acknowledgements}
\contentsname{Contents}
\indexname{Index}
\examplename{Example}
\keywordsname{Keywords}
\referencesname{References}
\endinput
```

The restrictions

are that as part of the replacement text tags are excluded which are not allowed in an argument of a `\def`.

2.3 Operations

What operations do we need? With databases we have the structure definitions of the fields, to fill these in via a fill-in screen, the various queries—read operations—next to the reporting features. I defined already the structures of the elements. My operations are

- adding to the database goes without fill-in screens but just by extending the file[6]
- as queries I provided: selective loading,[7] and making a list of names
- the reporting is the printing of the database, in general, a part of the typesetting task.

Adding to the database

comes down to extending the file with the list element.[8]

Selective loading

The loading is namelist driven, that is, the names of the entries to be loaded have to be specified.

For class I databases the markup for the loading reads

$$\backslash\langle kindofdatabase\rangle\{\backslash\langle name_1\rangle\ldots\backslash\langle name_n\rangle\}$$

with `<kindofdatabase>` either addresses, pictures, or references.[9]

For class II databases the markup for the loading reads for example

`letter` or
`report` or
`transparencies`

for formats. For tools it is tool dependent, for example for typesetting Pascal it comes down to

```
\beginpascal
<pascaltext>
\endpascal
```

with the details of loading hidden from the user.

Listing of all the names

can be obtained after, for example

```
\contentsdatabase{address}
%or
\contentstoolsorfmt{fmt}
```

Listing of names selected via pattern matching

In order to browse the class I databases, especially address.dat and lit.dat, the macro `\search` has been provided in blue.tex. The idea is that as input we specify a pattern—without periods, alas—and as result a list of names will be obtained, of which the replacement text contains the pattern. Those names and associated replacement texts are stored as definitions, i.e., the name will become the control sequence of the definition.[10]

Example *(Search for addresses from RUSSIA)*

Below the input has been given for the search of the pattern RUSSIA in the database file address.dat.

```
\input blue.tex
\searchfile{address}
\search{RUSSIA}
\bye
```

This is handy for making address labels grouped per country.
`\input blue.tex \letter`

[6] Perhaps someone can provide the screen fill-in facility on top some day?

[7] To load all the items is trivial, but generally too much. The gains in storage which can be obtained by selective loading have been reported in 'BLUe's References, revisited.' It is convenient to specify the elements by name.

[8] A white lie, more has to be done.

[9] Specifying by pattern will be treated a little later.

[10] This is precisely the reason to supply curly braces in the database entry.

```
\searchfile{address}
\search{RUSSIA}
\makesearchlabels
\bye
```

A search pattern should not contain a period.

Listing of contents

There are various kinds of listings, such as

- verbatim (format and tools, total or selective), and
- typeset (address labels, total bibliography).

Verbatim listing of a tool.

This problem is induced by my multi-file approach. Of course one can extract the tool by an editor et cetera as alternative.

Example *(Verbatim listing of macros of pascaltool)*

In order to do these kinds of things I have added tags as comments to the file. For example `%!cgl;newcol`, with the intention to use `;newcol` for a new column in a pfile listing and as an end separator with `\cgl` to terminate reading. Quite confusing but once understood it is handy to have those hooks.

```
\input blue.tex \let\pascaltool=x
\long\def\tool#1{\ifx#1\undefined
   \bgroup\unouterdefs\ea\gobbletool
   \else\ea\toolverbatim\fi}
\def\cgl;newcol{\endverbatim\endinput}
\input tools.dat
\bye
```

Remark. For a convenient verbatim listing of fmt.dat or tools.dat use pgfile.tex, which is a generalization of pfile.tex to print verbatim blue.tex.[11]

```
%pgfile.tex cgl, March 1995
\input blue.tex \hfuzz=25pt
\lineskip1pt plus2pt minus1pt
\baselineskip12pt plus 2pt minus1pt
\message{Type name for file:}
\read16 to\namefile
\title{File: \namefile}
\issue{Version 1.0}
\beginscript
  \thisverbatim={\catcode`\;=0
  \catcode`\!=12
  \catcode`\|=12
  \input \namefile}
  \beginverbatim
  ;endverbatim
\endscript
```

A convenient 'listing' of the address database is a list of address labels.

Example *(All address labels)*

```
\input blue.tex \letter
\makealllabels
\bye
```

3 Use

The use has been treated by example and is part of the larger typesetting task. For class I databases the use after specification reads as follows.

```
\pasteupreferences                    references
\<picturename>                         a picture
\letterto{...}          address(es) & letter.tex
```

More details about the use of `\letter` and pictures are provided in the user guide 'Publishing with TEX,' or more specifically in 'BLUe's Letters' and 'BLUe's Graphs.'

Example *(Typesetting letter(s))*

I assume that the letter proper has been stored in letter.tex.[12]

```
\input blue.tex \letter
\subject{\TeX{} for BLU}
\ourreference{22 1 95}
\yourreference{\TB}
\letterto{\knuthde\ntg}
```

Example *(Typesetting references)*

```
...
\references{\knuthdea\laancgc}
\beginscript
...
\pasteupreferences
\endscript
```

4 Miscellaneous

4.1 Testing for integrity after extension

Because of the restrictions it is advisable to test a database for use when it has been extended. The simplest test I could think of is to provide the 'table of contents,' that is a list of all the names. The test job for lit.dat reads as follows.

```
\input blue.tex
\contentsdatabase{lit}
\bye
```

No pages of output will be produced (that is OK!). The file contentslit will contain the list of all names. When this job is successful we know that TEX will not complain while scanning.

Note that in blue.tex I provided `\newwrite \toclit` and the like.

4.2 Extension with a format or tool

With tools and formats we have to work a little harder. In order to extend the database of class II, the following must be done.

- provide for user-interface macros as part of the kernel
- guard against multiple loading

[11] Within the context of verbatim printing I use the semi-colon as escape character.

[12] It is also possible to supply the letter in the job, but that is not so relevant for the database aspects.

- allocate storage in the kernel, blue.tex, i.e., only once.[13]

Note that outer definitions can't be used as such.[14]

Example *(Addition of the report format to format.dat)*

In order to achieve the use of `\report` we have to add to blue.tex the following overhead, and to include in fmt.dat the macros proper, preceded by `\tool\reportfmt` and ended by `\endinput`.

```
%User interface in blue.tex
\def\report{\ifx\undefined\reportfmt
   \let\reportfmt=x\fi
   \ifnum\reportloadcnt=0 \ea\loadformat\fi
   \advance\reportloadcnt1
   \let\reportfmt\undefined}
%with auxiliaries
\def\loadformat{\input fmt.dat\relax}
%Storage allocation
\newtoks\chapternumbering
%et cetera
%And in fmt.dat the blue collar workers
\tool\reportfmt
<reportmacros>
\endinput
```

I did not supply a message when `\report` was already defined, on purpose.[15]

Note that I did not talk about the design of `\report`. The redesign and coding of `\beginscript`, for example, is another issue, and beyond the scope of this paper.

Conventions for entries of fmt.dat

come down to

- update the contents at the beginning of the file fmt.dat
- include in the beginning of the format `\message{ ---<...> format, <date>, <owner>--- }`
- adhere to visual layout of the code for the printout
- insert additional comments which contain markup for the line numbering
- provide of table of contents with line numbers
- add a history of changes log.

The layout is aimed at printing in two columns, with a new contribution starting a new column. The selective line numbering is controlled by the control sequences

- `\numvrb`, to start the line numbering
- `\vrblin` = `<number>`, to adjust the line counter, and
- `\nonum`, to switch off the line numbering.

Do remember that within the context of a printout the escape character is the semi-colon.

Example *(Template for an entry in fmt.dat)*

```
%end%<lastlinepreviousformat>     %;newcol
%
%
%begin%%%%%%%%%%%%%%%%%%%%%%%%%<...>%;numvrb
\tool\<...>fmt
\message{ ---<...> format, <date>,
                      <(cr)owner>--- }
%<authorinformation>
<theformatproper>
\endinput                         %;nonum
%Contents
%<item>...................<pagenumber(s)>
%etc
%History of changes
%<date> <change>
%etc
%end%%%%%%%%%%%%%%%%%%%%%%%%%<...>%;newcol
```

A similar template holds for tools.dat. See 'Publishing with TEX,' for the details.

5 Coding

In BLUe's format two-part macros are the starting point. A one-part macro is provided on top of it. That is one of the conventions I adopted, relevant for the coding of the handling of databases.

The two classes of databases have a different user interface, and therefore the coding proper is class-specific.

5.1 Addresses, references, and pictures

For specification of the addresses, the references or the pictures, assign the names to a toks variable.[16] The loading from the database is for all three applications the same. However, what we do with it further depends on the application. Just to give you a break I'll digress a little on the latter.

Pictures are special, because we also need to load the common macros for use within any picture, assumed we use pictures at all.

Addresses have as extra that they have to be merged with one or more letters, and with the background material such as logo, sender affiliation and the like.

With references the extra work comes from the need for cross-referencing, and to paste them up at a place at will.

In this note I'll restrict myself to the loading of list elements and leave open what to do with these elements in the script. The latter is treated in the specific articles 'BLUe's Graphs,' 'BLUe's Letters,' and 'BLUe's References.'

The coding for loading elements from a database contains the two major steps[17]

[13] Remember that TEX lacks a garbage collector.

[14] Replace the definition by a non-outer one, or use `\csname`. One can also add the def or symbol to the set of `\unouterdefs`.

[15] I usually start separate chapters with `\report` and then I don't like to have to remove the `\report` tags in the chapters, when the total will be processed. Nor do I like to get messages, because I know already that the tag is at the start of each chapter. Of course reloading is prevented.

[16] Well, . . . a white lie, it looks like it.

[17] Because of the freedom to give it any meaning it is handy to let the names stand for the error message at first. In case of a typo in the name it already contains its error message. Neat!

- define the specified names
- load the definitions associated with those names.

In other words, in contrast with the usual procedure — if something is already known don't load it again—the opposite has been applied here. When a name is known it will be overloaded, i.e., replaced by what we really want. As extra I created a `\namelst` with the specified names each preceded by `\lst`.[18] As abstraction I used in the coding below the root name 'kind of database, KoD for short' which could equally well have been addresses, pictures, or references, in principle.

```
\def\<KoD>#1{\begin<KoD>#1\end<KoD>}

\def\begin<KoD>#1\end<KoD>{%
  \let\process\processnames\fifo#1\ofif
  \loadselectivefrom{\the\<KoD>file}}

\def\processnames#1{\ifx\undefined#1
    \let#1\<KoD>error
  \else\message{\Dash\string#1
         already loaded.\Dash}%
  \fi\namelst\ea{\the\namelst\lst#1}}

\def\loadselectivefrom#1{%#1 <name>
 \def\lst##1{\ifx##1\undefined\ea\gobble
             \else \ea\gdef\ea##1\fi}
 \input #1.dat \relax}%e.g. lit.dat

\def\<KoD>error{listelement not
    in database, (Sorry.)
    \loaderror{<KoD>}}

\def\loaderror#1{\message{#1 not in
      database}}
```

Explanation. Remember that the file consists of triples: `\lst`, `\<name>`, and a group, 'the replacement text.' Are you ready? There we go. All the specified names are defined and obtain first their error message as replacement text. This is done via the use of the fifo paradigm. The loading overrides each (error message) replacement text with what we want. This is done via looking at `\lst` as the list element tag with as argument `\<name>` with the purpose to store or to gobble the third element of the triple.

As extra the names are stored in `\namelst`, each preceded by `\lst` to facilitate later processing, i.e., without recursion, just execute the `\namelst`, with an appropriately defined `\lst`.

The one-part macro for this simple case is trivial and added for consistency. The argument does not need processing on the fly.[19]

5.2 Formats and tools

For the loading it comes all down to what we assign as meaning to the list element tag, that is, what meaning we assign to `\tool`. In the same spirit as with references and the like, it comes down to gobble the third element of the quadruple if the `\<name>tool`, respectively `\<name>fmt`, is unknown. Otherwise all the stuff up to `\endinput` will be loaded, and that is it.

The entries from format.dat are not mentioned below, because that is not relevant any longer. They just have to obey the earlier given syntax.

General macros and basic tools for selective loading of formats are the following, borrowed form blue.tex.

```
\def\gobbletool#1\endinput{\egroup}
%
%Define the list element tag '\tool'
\long\def\tool#1{\ifx#1\undefined
   \bgroup\unouterdefs\ea\gobbletool\fi}
%
\def\unouterdefs{%List of defs which
                 %have to be neglected
  \ea\let\csname+\endcsname\relax
  \catcode`\^12
}
%
\def\toolverbatim{\thisverbatim{%
    \baselineskip12pt plus2pt minus1pt
    \lineskip1pt plus1pt minus1pt}%
    \beginverbatim}
%
\def\loadformat{\input fmt.dat\relax}
```

Note that `\unouterdefs` have been kept local.

The user-interface macro for `\letter`, as provided in blue.tex, reads as follows.

```
\def\letter{\ifx\undefined\letterfmt
   \let\letterfmt=x\fi
   \loadformat
   \let\letterfmt\undefined}
```

Multiple loading has been safe-guarded in `\report`, via the use of the counter `\reportloadcnt`, as follows.

```
\newcount\reportloadcnt
%Prevent double loading.
\def\report{\ifx\undefined\reportfmt
   \let\reportfmt=x\fi
   \ifnum\reportloadcnt=0 \ea\loadformat\fi
   \advance\reportloadcnt1
   \let\reportfmt\undefined}
```

5.3 List of names in the database

The following macro does the job for the class I databases. It creates a file contentsaddress, contentslit, or contentspic, with the list of all the names.

```
\def\contentsdatabase#1{%#1 address lit pic
\ea\let\ea\name\csname toc#1\endcsname
\immediate\openout\name=contents#1
\def\lst##1##2{\immediate\write\name{\nx##1}}
\input #1.dat\relax}
%with auxiliaries
\newwrite\tocpic
\newwrite\toclit
\newwrite\tocaddress
%test example
\contentsdatabase{address}
\bye
```

[18] I did not want to bother the user with preceding each name by `\lst`. I chose for alleviating the task for the user.

[19] Because I chose to typeset `\listelementerror` when not overwritten, it must obey the syntax of the typesetting macro.

Note that in order to write the (undefined) names no expansion must take place.

For databases of class II—with list element tag `\tool` and end separator `\endinput`—the following does the job. It creates a file `contentsfmt`, or `contentstools`, with the list of all the names.

```
\def\contentstoolsorfmt#1{{%#1 tools fmt
\ea\let\ea\name\csname toc#1\endcsname
\immediate\openout\name=contents#1
%Define list element tag '\tool'
\long\def\tool##1##2\endinput{%
    \immediate\write\name{\nx##1}}
\unouterdefs
\input #1.dat\relax}}
%with auxiliaries
\newwrite\tocfmt
\newwrite\toctools
%test example
\contentstoolsorfmt{fmt}
\bye
```

Note that in order to write the (undefined) names to a file no expansion must take place.

5.4 Search by pattern

Searching a database for a pattern with the aim to yield a list of names—each name preceded by `\lst` in the toks variable `\namelst`—has been coded as follows.

```
%Searching the database
\def\search#1{\def\loc##1##2{%
   \def\locate####1##1####2\end
    {\ifx\empty####2\empty\foundfalse
     \else\foundtrue\fi}%end locate
   \ea\locate##2.##1\end}%end loc
   \def\lst##1##2{\loc{#1}{##2}\iffound
    \immediate\write16{\nx##1}%log file
    \namelst\ea{\the\namelst\lst##1}\fi}
   \input \the\searchfile.dat\relax}
%with auxiliaries
\newtoks\searchfile
\newtoks\namelst
```

Explanation. The argument of `\search` is matched with the replacement text of each list element. When found the name of the list element is added to the token variable `\namelst`. For the time being the name is also written to the log file, for the purpose to verify what has been added to the token variable.[20]

For a pedagogical stepping stone see 'Syntactic Sugar' where I used the `\loc` macro to locate a letter in a string, and applied it to Appelt's problem of doing something special to capital characters.

6 Looking back

It all started with my surprise when studying AMS styles and formats that references have to be marked up over and over. The use of databases was badly needed. So I started to think about why it has been overlooked. I realized that from a database only a few references will be included in an article or so. On the other hand, the idea struck me that an author should not bother about formatting references at all. He should supply only names to entries already available, and appropriately marked up, in the publisher's database.

The overhead of other tools in use by the community was clear to me, so I started to work on 'BLUe's References.' During that work I found how to load selectively from a TEX database. That was a gem, a paradigm, and bound to be used elsewhere. Since then, especially while working on 'BLUe's Format,' I have used the mechanism for addresses—to be merged with letters—for pictures—to be stored scaling invariant, and to be invoked with a particular `\unitlength`.

When I faced the problem of how to store elegantly formats and tools in general, I combined my[21] selective loading approach with Knuth's lists. Et voilà.

7 Looking forward

I don't know as yet to what extent the above is useful for databases which are an order of magnitude larger, i.e., with thousands or tenthousands of entries.

8 Acknowledgements

Thank you Jos Winnink for your interest in the database approach with TEX, and for stimulating me a bit for writing this separate article on the issue.

9 Conclusion

It is so nice to watch the TEX log file scroll by slowly, witnessing the searched databases, smoothly and on-the-fly.

References

The TEXbook and LATEX user's guide are omni-present and not explicitly listed. For my works consult lit.dat, via the use of `\search` for example.

[20] The searching for a pattern within TEX has been used by me on several occasions already. To start with typesetting bridge, where the purpose was to locate a card in a hand, and if so to delete the card, that is, update the hand. This was a special case of determining whether an element belongs to a set. The latter occurred further in determining whether a token belongs to the set of accents, whether a token belongs to the set of reserved words of Pascal, and the like. Sooner or later it had to pop up in pattern matching.

[21] Well, . . . it has been essentially in *The TEXbook* 384 all the time. I'm only wondering how much there is still left unnoticed.