Programming bibliographies
Jean-Michel Hufflen

Abstract
We are interested in situations such that using the full expressive power of a programming language is needed when ‘References’ sections are generated for a source text suitable for \( \text{La} \text{T} \text{E} \text{X} \). The data model used by \texttt{BibTeX} is inadequate from this point of view; the \texttt{bblatex} package is based on a more efficient data model, but workarounds may be needed in some circumstances.

1 Introduction
Early computer programs were very different from those running nowadays. Sometimes they were written using programming languages with syntactical features now viewed as strange; they were running on computers whose performance was not comparable with today’s. Besides, these early programs, in the 1950s, were only handled by people specialised in computer science: on the one hand, some rigid syntax was required for inputs, on the other hand, outputs were provided using raw forms, and graphical interfaces were nonexistent.

Things have evolved rapidly for years now, and nowadays many people are able to use programs and operating systems even if they do not have any knowledge about programming. A good example is given by interactive word processors (WYSIWYG\(^1\)), such as Microsoft Word. Secretaries are able to use them, as end users simply typing input texts. Any end user of Word is able to customise it, mainly by means of graphical menus. So nowadays a rich collection of programs—including Word—can be used as any object of everyday life, like a washing machine or a dishwasher.

What is the point of non-interactive typesetting systems (WYSIWYM\(2\)), such as \texttt{La} \text{T} \text{E} \text{X} \text{\textappliance}, according to this point of view? In fact, if an end user only deals with default constructs or predefined document classes, \texttt{La} \text{T} \text{E} \text{X} can be viewed as a kind of black box simply accepting input source texts and producing output texts. Some customisation and extension can be reached by means of \texttt{packages}, introduced by \texttt{La} \text{T} \text{E} \text{X} \text{\textappliance} \texttt{et} \texttt{e} [10]. Stronger customisation is allowed by means of \texttt{commands}, written using \texttt{La} \text{T} \text{E} \text{X}'s programming language. Theoretically, this language has the same expressive power as a Turing machine’s language, so any function can be programmed using it.\(^3\)

In practice, \texttt{La} \text{T} \text{E} \text{X}'s language—which is based on \texttt{macros}—has been mainly designed to handle \texttt{text} \texttt{fragments}. As a kind of counter-example, even though we can implement a sort procedure using this language [9], that is a worthwhile exercise for its own sake, rather than an actual advantage for using \texttt{La} \text{T} \text{E} \text{X}. In fact, some ‘pre-computations’—before typesetting a fragment—such as capitalising some words, or putting them using lower or upper-case characters, can be easily expressed using \texttt{La} \text{T} \text{E} \text{X}'s language, but more advanced features related to programming features are difficult to handle.

This point is one of the reasons why \texttt{LuaTeX} has been developed [2], allowing tasks more related to programming to be delegated to a more modern programming language. Since the initial versions of \texttt{LuaTeX}, \texttt{La} \text{T} \text{E} \text{X} is fully able to typeset the result of a computation performed by a program, for example, sorting an array before displaying its successive elements. Typesetting the results built by a spreadsheet program is another example.

In this article, we propose to apply such a view to \texttt{bibliography processors}, that is, generating ‘References’ sections for \texttt{La} \text{T} \text{E} \text{X} documents. In other words, many end users should be able to use such a program without any knowledge about programming, but some specific applications may need such knowledge—provided that the format used by this processor is open—and it is important for such a bibliography processor to allow the direct programming of some specific functions.

In Section 2, we recall the tasks a bibliography processor should perform and we make precise our terminology. Section 3 briefly reports the main bibliography processors’ current state. Then Section 4 goes thoroughly into some specific points. After some discussion, Section 5 concludes about our approach.

2 Tasks of a bibliography processor
Let us consider a source text (\texttt{.tex} file) including some bibliographical citations by means of \texttt{La} \text{T} \text{E} \text{X}'s \texttt{\cite} command, whose argument is a so-called \texttt{citation key}.

(i) A bibliography processor can use citation keys to \texttt{search} bibliography databases (\texttt{.bib} files) for corresponding \texttt{entries}.

\(^1\) What You See is What You Get.
\(^2\) What You See is What You Mean.
\(^3\) There are many online documents about this subject, more or less easy to read. A didactic written document is [6], showing how to implement a kind of \( \lambda \)-calculus in \texttt{La} \text{T} \text{E} \text{X}.
(ii) These entries should be sorted, unless the document’s bibliography is unsorted, that is, the order of the bibliography’s items is the order of first citations throughout the document’s body.

(iii) Finally, each bibliographical entry should be arranged into a bibliographical reference that future readers can consult, most often at the document’s end, in which case they are stored in a .\bibliography file. Citation keys used throughout the document are replaced during this step: a reference may be identified by a number in plain bibliography styles, by an alphanumeric label in alpha styles. More advanced bibliography styles, such as author-date or short-title, have been successfully implemented within \LaTeX, as reported in [11, Ch. 12].

3 Bibliography processors

For a long time, \BIBTEX [13] was the only bibliography processor usually associated with \LaTeX. It uses information stored in auxiliary (.aux) files and is able to perform the steps (i), (ii) and (iii). \BIBTEX is still used, at least by some conference submission tools, although it is an old program, which does not address modern requirements such as multilingual encodings. \BIBTEX’s bibliography styles have been implemented using the .bst language [12], sometimes complemented by a \LaTeX2e package within some advanced styles such as natbib or jurabib [11, Ch. 12]. This language for bibliography styles — based on handling a stack — is old-fashioned, more suitable for small changes than programming a new bibliography style. However, this language expresses general algorithms. That is, an end user developing a new style using this language has available the possible sequence of operations a computer can perform.

Nevertheless, the same observation can be made as for using \TeX’s language: this bst language is difficult to handle practically.

During the last decade, a new modus operandi has been put into action by the \biblatex package [8]: formatting bibliographical references — that is, most of step (iii) — is wholly controlled by \TeX macros and deferred to \LaTeX’s next pass. Steps (i) and (ii) can be delegated to \BIBTEX, but \biber [7] is preferred, because more features are available. The \biber program does not use .aux files, but control files (.bcf files), built when the \biblatex package is loaded with the option backend=\biber. In addition, this \biblatex/\biber duo has introduced many new fields, many new bibliography types and styles; it seems widespread in the humanities.

The bib module of \ConTeXt — another format built on top of \TeX [1] — works analogously, in the sense that many tasks are deferred to \ConTeXt’s next pass; modern versions of this module are mostly implemented using the Lua programming language.

As another possible bibliography processor, we have personally put into action the implementation of a successor to \BIBTEX: \MIBIBTEX [3], written in the Scheme programming language. We think that a functional programming language is very suitable for this kind of task, because such languages allow the use of functions as parameters, e.g. in:

\begin{verbatim}
    (lambda (f2) (f2 1 2))
\end{verbatim}

To add (resp. multiply) 1 and 2, just apply this expression to + (resp. *). A bibliography style can be compared to this expression, because the resources are known — they are the successive bibliographical entries — whereas the way to arrange these data together is to be determined w.r.t. the bibliography style chosen. \MIBIBTEX has been developed according to such an approach. It has been used rather occasionally, but as far as we know, using it has always proven successful. In particular, it has been used to populate the French official site for open archives from a collection of .\bib files, as reported in [4]. A new version has been announced in [5]; its new features include a better interface with Scheme definitions. This point seems to us to be important and we are going to explain why.

4 Some operations

4.1 Sorting

Let us consider again the tasks enumerated in §2. Step (ii) — sorting the extracted bibliographical entries — would be very difficult to program in \TeX’s macro language, as mentioned in the introduction. Although the \biblatex package tends to use \LaTeX to perform as many operations as possible, this sort step is still performed by the bibliography processor associated, \BIBTEX or \biber. The former provides a ‘basic’ lexicographical sort procedure, in practice only suitable for the English language (without manual adjustments). The \biblatex package allows some customisation of the sort procedure performed by \biber, by means of the sorting option and mnemonics denoting sort keys — e.g., nyt is a sorting scheme for name, year, and title [8, §3.1.2.1].

Neither of these two processors can perform a numeric sort: with these two bibliography processors, the year information is correctly processed because years are supposed to use the same number of digits. Let us go thoroughly into chronological
order: Bib\TeX{} does not sort w.r.t. the month information; biber could do that, since the month information handled by biblatex consists of numbers,\(^5\) but in practice no predefined sorting scheme does so.

The biblatex/biber duo does offer more ways to sort bibliographical entries in comparison with ‘old’ Bib\TeX{}. Moreover, its sorting schemes allow people unfamiliar with computer science to specify precise sort keys, but:

- a descending sort can be applied only to years within predefined schemes;\(^6\)
- the number of person names — for authors or editors — considered for sorting — given by the \texttt{maxnames} constant \cite[§3.1.2.1]{biblatex} — is limited;\(^7\) the sort procedure does not deal with unbounded number of person names, even if the maximum number considered can be changed by end users;
- some bibliographical fields mainly used as sort keys — e.g., \texttt{AUTHOR}, \texttt{TITLE}, \texttt{YEAR} — may be substituted by sort‐suitable fields during this procedure — \texttt{SORTNAME}, \texttt{SORTTITLE}, \texttt{SORTYEAR}: excess markup for corresponding information is avoided, but there is some risk of information redundancy;
- some exotic sorts can be handled by redefining the beginning and end of the sort procedure, by means of the bibliographic fields \texttt{PRESORT} and \texttt{SORTKEY} \cite[§3.6]{biblatex}; more difficult cases can be processed by defining new sorting schemes by means of the \texttt{\textbackslash DeclareSortingTemplate} command \cite[§4.5.6]{biblatex} — to be put in a source \texttt{.tex} file — but this last command defines immediate sort keys by means of additional fields rather than new sort procedures.

In our personal opinion, the biblatex package’s conventions allow people unfamiliar with programming to deal with a rich collection of possible sorts, but if you need to add a new sort procedure, you do not have the full expressive power of a programming language.\(^8\) As an ambitious example, we personally were in charge of the publication list of our laboratory some years ago: we had to sort this list first by research teams, second by categories,\(^9\) third by decreasing years, fourth by authors’ names (increasing), fifth by decreasing months. That would have been possible with biblatex/biber but quite difficult and/or requiring some information redundancy.

### 4.2 Labelling references

The automatic generation of non-ambiguous bibliographical keys for \textit{references} — the keys that will appear within the resulting document — is now satisfactory. Let us remark that if end users would like to use their own keys, they should use a \textit{short-title} system.\(^10\) If we consider \textit{alpha} styles, we would like to point out that such a style does not belong to the author-date system. Bib\TeX{} and biblatex add a letter if several references share the same author and year, that is:

\begin{itemize}
\end{itemize}

According to another approach, we can label a first or unique reference with a name’s abbreviation and a year. If need be, a possible second reference with the same information is given an additional letter, that is:

\begin{itemize}
\end{itemize}

The first way is often used in modern documents, but the second can be observed. Our opinion is that this problem can be solved with access to the function generating successive labels within the bibliography processor.

### 5 Conclusion

We think that M\LaTeX{}’s new version will be finished at this year’s end. We expect to provide an open system, comparable with the Emacs\(^11\) editor. That is, usable as it is, but also \textit{customisable} and \textit{extensible}. As mentioned in the introduction, \LaTeX{} has very rarely been put into action in practice.
has these qualities, and the success of Lua\TeX{} shows that the addition of a modern programming language’s power has opened new windows.

Even if not all of a bibliography processor’s customers are computer scientists, tasks performed by such a tool belong to programming. When the Unix operating system emerged, the idea was that each person using it would be close to someone who precisely knew how Unix worked. That is, a kind of synergy. We think that a comparable synergy should be reached for an open bibliography processor, which implies that the structures it uses are open. This was already the case for MiBiB\TeX{}’s last version. Likewise, most of the new fields and types introduced by biblatex will be recognised and handled by MiBiB\TeX{}’s last version, so we can claim that we are not in opposition to biblatex. We aim to bring another view of what a bibliography processor should be.

Acknowledgements

It is a great pleasure to thank the first version’s proofreaders, Barbara Beeton and Karl Berry.

References


○ Jean-Michel Hufflen
FEMTO-ST (UMR CNRS 6174) &
University of Bourgogne Franche-Comté
16, route de Gray
25030 BESANÇON CEDEX
FRANCE
jmhuffle (at) femto-st dot fr
members.femto-st.fr/Hufflen-Jean-Michel/en

Programming bibliographies