MlBibTEX & ConTEXt: Face-to-Face

Jean-Michel Hufflen

We summarize what the MlBibTEX bibliography processor can provide to the ConTEXt typesetting system, even if MlBibTEX has been initially designed to work with LATEX. We review the successive steps of tasks performed by MlBibTEX, and show how some significant differences between LATEX and ConTEXt can be managed. In addition we show how some of these tasks may be interfaced by means of external functions called by ConTEXt. We also mention some original features of MlBibTEX, including those introduced by the new version, still in development. Last but not least, MlBibTEX can allow ConTEXt users to take advantage of constructs introduced by the `biblatex` package.

Introduction

The BibTEX word processor [1] was initially designed to work in conjunction with the LATEX word processor [2]. Later, an attempt to use BibTEX for generating 'References' sections suitable for ConTEXt documents was Taco Hoekwater's `bib` module [3]. However, this BibTEX program is ageing: in particular, its bibliography styles (.bst files) are written in an old-fashioned language using postfixed notations and based on handling a stack [4]. Besides, new requirements have appeared over the past decade — e.g., multilinguism, character encodings — beyond the capabilities of this venerable program. Some possible replacements have already been proposed for LATEX documents' bibliographies: an emerging alternative is the `biblatex` package [5] in tandem with the biber bibliography processor [6], the latter being written using Perl[1]. Let us notice that this bibliography processor is unusable within ConTEXt because it only generates bibliographies for LATEX's `biblatex` package. Another proposal is MlBibTEX[2]: first it was designed to focus on multilingual features[3], then it has evolved towards a 'better BibTEX' in the sense that a developer of bibliography styles is given more services closer to 'actual' programming. For example, BibTEX cannot perform numerical sorts, it only provides lexicographic ones[4], whereas MlBibTEX can deal with both, according to the type of processed data. Practising MlBibTEX's implementation language — Scheme — is not needed for end-users, but allows advanced developers to put ambitious features into action. In addition, MlBibTEX is able to generate bibliographies according to formats other than LATEX's, in particular, XML[5] dialects [8].
The present article aims to summarize what our MlBibTEX program — as an *adaptable modern* bibliography processor — has already provided to the ConTEXt typesetting

1 **P**ractical **E**xtraction and **R**eport **L**anguage.

2 **M**ulti**L**ingual BibTEX.

3 MlBibTEX's history is summarised in [7].

4 The same for biber.

5 e**X**tensible **M**arkup **L**anguage.

system and can provide to it in the future. A short description of the `mlbibcontext` program — derived from MLBIBTEX's kernel[6] — has already been given in [9]. Here we go thoroughly into more technical details; in particular, we incorporate some material shown at previous ConTEXt meetings, these talks have not been complemented yet by written articles:

- *Bibliography Tools and ConTEXt/LuaTEX* (Bassenge, September 2011),
- *All Roads Lead to* `mlbibcontext` (Brejlov, September 2013),
- *Calling MLBIBTEX's Functions* (Nasbinals, September 2015).

The first two talks referred to MLBIBTEX's first version distributed publicly (1.3). Since Spring 2015, we started a new version (1.4) — introduced in [7] — and some new features have been presented at the last ConTEXt meeting, in September 2015; in particular, this new version is Unicode-compliant. What is shown in the present article is based on the new version, presently in test. Concerning ConTEXt's version, we use MkIV, allowing operations more related to 'actual' programming to be written using the Lua language [10].
In the first section, we explain why a more strict analysis of string values associated with BIBTEX fields, within bibliography database (.bib) files, is needed in order to derive bibliographies for ConTEXt documents. In § 1.1, we will refer to some notions and commands related to LATEX, but only some basic knowledge of this word processor is needed. After a short introduction to ConTEXt's `bib` module, we explain how MLBIBTEX addresses this problem and enumerate the basic cases we can process. Then Section 2. shows how the successive tasks MLBIBTEX performs are launched and organised. We will see how intermediate results may be managed, possibly by means of methods written using the Lua programming language and interfacing Scheme functions. Some points given in this section are very technical, in particular, the use of Scheme definitions, but they can easily be skipped if you are not interested in implementation issues. Section 3. summarises the features of MLBIBTEX, some being to be developed for ConTEXt. Section 4. explains how MLBIBTEX's executable programs are updated w.r.t. the new version 1.4. Finally, the bibliography of the present article has been generated by our executable program `mlbibcontext`.

1. Bibliographies for ConTEXt

1.1 Using .bib files

As mentioned before, BIBTEX difficultly meets some new requirements: these points are explained in [11]. Hereafter, we only focus on using BIBTEX to generate bibliographies suitable for ConTEXt documents rather than LATEX ones. Let us look at Figure 1. We recognise the .bib format and its *entry types*, such as `@BOOK`. Each bibliographical entry is specified by values associated with *fields*, some being *required* (e.g., `TITLE`), some

[6] Throughout this article, the 'MLBIBTEX' logo is used for a large set including this program's kernel and all the derived executable programs, the `typewriter` font for an executable program belonging to this set. So, '`mlbibtex`' (resp. '`mlbibcontext`') is for the executable program derived from MLBIBTEX and able to generate bibliographies for LATEX (resp. ConTEXt) source files.

optional (e.g., SERIES); the fields unrecognised by a bibliography style are *ignored*. In addition Figure 1's examples use some syntactic extensions provided by MLBIBTEX:

- the LANGUAGE field is for the language of the current entry[7], the conventions are given in [12];
- the first entry's AUTHOR field uses *keywords* — 'abbr =>', 'first =>', ... — for a person name's recognised parts[8] and a new connector — 'with', analogous to 'and' — for *collaborators* rather than co-authors: see [13] for more details; let us mention that *mixed* notation — using both standard notation for names and keywords — is allowed: let us consider Figure 1, we use the 'classical' BIBTEX notation for the first and last names of the first entry's first author, and the 'abbr =>' gives a nonstandard abbreviation of the first name[9];
- the second entry uses *multilingual annotations* [14] within the ADDRESS field; let us assume that this entry is cited throughout a document written in English or French, and information should be put using the document's language as far as possible, so 'Munich' will be put if the ADDRESS field is processed by the bibliography style used.

By the way, let us notice that the MONTH information is put within a bibliographical reference, within .bbl files, if the bibliography style handles this information, but is not used by BIBTEX to sort entries.
If such entries are processed by 'old' BIBTEX, these syntactic extensions cause warnings or result in strange-looking references. In fact, the two ways to increase BIBTEX entries' expressive power slowly are the addition of new fields[10] and the markup of values associated with fields by means of LATEX commands. There is no problem if source texts for LATEX are derived, provided that these commands are defined. As a consequence, we may need a more precise analysis of such values if other formats — e.g., simple text, HTML[11], or XML — are targeted. Since ConTEXt and LATEX are both built out of TEX, defining some LATEX commands in ConTEXt solves some cases, but not all. Let us consider Figure 1's examples again. The first entry's NOTE field includes a markup to emphasise the 'Oregon' word, but the LATEX \emph command is unknown within

[7] W.r.t. MLBIBTEX's terminology, .bib files contain bibliographical *entries*, whereas bibliographical *references* — which are inserted into a source file for a typesetting system, such as LATEX or ConTEXt — result from a bibliography processor's task.

[8] The '=>' sign belongs to such a keyword; more precisely, such a keyword consists of a key identifier, followed by '=>''. *Stricto sensu*, these keywords are not needed in this example, a workaround would allow the first name to be abbreviated correctly; as shown in [13], such workarounds related to person names and based on dummy LATEX or ConTEXt commands may be quite simple or more complicated.

[9] By default, BIBTEX and MLBIBTEX abbreviate a first name by retaining only its first letter, followed by a period. In some cases, this *modus operandi* is incorrect, but in fact, typography books do not agree about abbreviating first names. Besides, such abbreviations are language-dependent, or even context-dependent. As an example of this last feature, 'Clive Eric Cussler' is usually abbreviated into 'Cl. Cussler' whereas the well-known abbreviated form for 'Clive Staples Lewis' is 'C. S. Lewis'. At least, our 'abbr =>' keyword allows us to process such cases.

[10] So does the biblatex package: many new fields are added to basic ones. Let us recall that fields unrecognised by bibliography styles are ignored by BIBTEX.

[11] **H**yper**T**ext **M**arkup **L**anguage.

```
@BOOK{cussler-du-brul2010,
 AUTHOR = {Clive Eric Cussler,
           abbr => Cl. with
           first => Jack B.,
           last => Du Brul},
 TITLE = {The Silent Sea},
 SERIES = {\emph{Oregon} Files
          Adventures},
 PUBLISHER = {Penguin Books},
 YEAR = 2010,
 MONTH = mar,
 LANGUAGE = english}

@BOOK{wienfort2008,
 AUTHOR = {Monika Wienfort},
 TITLE = {Geschichte Preu{\ss}ens},
 SERIES = {Wissen},
 VOLUME = 2456,
 PUBLISHER = {Verlag C.~H. Beck},
 ADDRESS = {[M\"{u}nchen] ! german
            [Munich] ! english
            [Munich] ! french},
 YEAR = 2008,
 MONTH = aug,
 LANGUAGE = german}
```

Figure 1: Example of an enriched .bib file.

ConTeXt. In this case, the best solution could be to use the basic construct ‘`{\em …}`’ of LaTeX’s first version, known by both LaTeX and ConTeXt. However, the emph command, introduced by LaTeX 2$_\varepsilon$, is more recommended for LaTeX users, and we can find *many* .bib files where this emph command is used[12]. In addition, other cases are more subtle: if some words are to be put using both italicised characters and bold face, there is no common construct: ‘`\textit{\textbf{…}}`’ or ‘`\textbf{\textit{…}}`] in LaTeX vs ‘`{\bi …}`’ in ConTeXt. Finally, the same name may denote different commands in LaTeX and ConTeXt: as an example, the second entry’s title uses the German ‘ß’ character, accessed by the LaTeX command `\ss`, whereas this command causes a switch towards a sans-serif font in ConTeXt. The right ConTeXt command to get ‘ß’ is \SS; in LaTeX, \SS causes the case of ‘ß’ to be raised and gets the ‘SS’ sequence[13]. It can be noticed that this problem should disappear with the use of Unicode-compliant .bib files; however

[12] … and, more generally, a *huge* number of .bib files including LaTeX commands on the Web.

[13] See [2] for more details about the commands \ss and \SS in LaTeX.

```
\usemodule[bib]  % No longer needed,
                 % this module is
                 % now preloaded.

\setupbibtex[database=example]
\setuppublications
  [criterium=cite,numbering=yes]

\starttext

Some words about \cite[cussler-du-brul2010].

\placepublications
\stoptext
```

Figure 2: Using ConTEXt's `bib` module.

the 'ß' letter is quite frequent in German, and this `\ss` command would be still used in .bib files already developed.

1.2 Using the bib module

For a long time, the only way to use a bibliography processor for ConTEXt documents' bibliographies was the `bib` module, as mentioned in the introduction. A very simple example is given in Figure 2, the example.bib file used in the `\setupbibtex` command's `database` option includes entries pictured in Figure 1. In addition to standard fields of BIBTEX's entries, the bibliography style associated with this module (the cont-no.bst file) can process more entry types and recognises more fields — e.g., ABSTRACT and ANNOTATE — some pairs being quite redundant — e.g., KEYWORD and KEYWORDS, NOTE and NOTES. This module's behaviour is close to the `biblatex` package's in the sense that a *structure* is passed to LATEX — bibliographical references are just marked up with ConTEXt commands — when .bib files are read; formatting these references is deferred until the `\placepublications` command is used to put the document's bibliography. A difference, in comparison with both standard styles and `biblatex`: *all* the entries of .bib files are included into the .bbl file of references[14], only the `\setuppublications` command decides to select all or part of these references, according to the `criterium` option's value (`all` or `cite`). About this last point, our `mlbibcontext` program behaves like BIBTEX in the sense that only cited entries are extracted from .bib files. Concerning fields, some of those introduced by the `bib` module are recognised, they are listed in Table 1. In § 3.1, we will see that the 'recognised'' word means something more precise for MLBIBTEX. The non-standard entry types @PATENT and @PERIODICAL are processed by `mlbibcontext`, too.

[14] Let us remark that if an entry inside a `.bib` file is ill-formed, the whole process crashes. BIBTEX and MLBIBTEX just skip entries not retained, so some syntactic errors in such entries may be irrelevant.

ABSTRACT	DAYFILED	ISBN[15]	MONTHFILED
ANNOTE	DOI[16]	ISSN[17]	SIZE
ASSIGNEE	EPRINT	KEYWORDS	URL
DAY	NAMES	LASTCHECKED	YEARFILED

Table 1: Additional fields of the `bib` module recognised by the `mlbibcontext` program.

As an additional bibliography module of ConTEXt, `bibltx` can be loaded in order to define some LATEX command often used throughout .bib files. However, this is only some partial solution, and the modules `bib` and `bibltx` just aim to take advantage of .bib files, when there was a unique parser for such databases: BIBTEX's. In particular, these two modules and the associated .bst file cannot reach features beyond BIBTEX's ability.

1.3 MLBIBTEX's internal format

BIBTEX allows additional definitions of LATEX commands used throughout a .bib file to be exported, by means of a @PREAMBLE directive, all the directives of the .bib files used are concatenated and put at the beginning of the .bbl file containing corresponding references:

@PREAMBLE{{`\def\`*anewcommand*{…}…}}

Like BIBTEX, MLBIBTEX reads the @PREAMBLE directive when it has to generate outputs for LATEX. An analogous directive, @CONTEXTPREAMBLE, being the same syntax, has been introduced for ConTEXt's definitions:

@CONTEXTPREAMBLE{{`\def\LaTeXe`{…}…}}

In reality, this @CONTEXTPREAMBLE directive is of interest in such a case — that is, a 'specialised' command of LATEX[18] — but useless in practice for predefined font switch commands such as `\emph`, `\textit`, *etc*. When MLBIBTEX parses all or any part of a .bib file, the result — structured as far as possible — may be viewed as an XML tree. For example, processing Figure 1 results in the two XML trees pictured in Figure 3. Basic LATEX commands such as `\emph`, `\textit`, or `\textbf` are implemented by *elements*, we can observe that about the first entry's SERIES field. A more nested example can be given by parsing the value `\textit{\textbf{\LaTeX}}`, which would result[19] in:

```
<emph emf="no" iff="yes" bff="yes">
  <LaTeX-command command="\LaTeX"
                 verbatim="LaTeX"/>
</emph>
```

15 **I**nternational **S**tandard **B**ook **N**umber.

16 **D**igital **O**bject **I**dentifier.

17 **I**nternational **S**tandard **S**erial **N**umber.

18 The `\LaTeXe` command produces the 'LATEX 2$_\varepsilon$' logo.

19 There are many attributes of our emph element, but we do not make precise those that are bound to default values. W.r.t. MLBIBTEX's terminology, an attribute whose values can be yes or no is so-called a *flag* and its name is suffixed by 'f'. All the attributes of our emph element follow this convention.

`\emph`	`\textbf`	`\textrm`	`\texttt`
`\LaTeX`	`\textit`	`\textsc`	
`\TeX`	`\textnormal`	`\textsf`	

Table 2: Font switching LATEX commands recognised by MLBIBTEX's parser.

`\#`	#	`\ss`	ß
`\%`	%	`\textbar`	|
`\&`	&	`\textexclamdown`	¡
`\$`	$	`\textquestiondown`	¿
`\copyright`	©	`\textregistered`	®
`\P`	¶	`` !` ``	¡
`\pounds`	£	`` ?` ``	¿
`\S`	§		

Table 3: LATEX's commands replaced by single characters.

We can also see how a LATEX command known by MLBIBTEX's parser is processed: the `verbatim` attribute is to be used for formats other than TEX-related ones, that is, simple texts, or HTML documents, or XML ones. Commands unknown by MLBIBTEX are left verbatim within the result. The math environments '`$...$`' and '`\[...\]`' are implemented by the `LaTeX-math-mode` element and differentiated by the `displayf` attribute. Let us go back to the previous example, MLBIBTEX is able to generate these two possible outputs, according to the target typesetting system:

```
   LATEX  ⟸  \textit{\textbf{\LaTeX }}
ConTEXt  ⟸  {\bi \LaTeX }
```

The LATEX commands resulting in XML elements when a text is processed by MLBIBTEX's parser are listed in Table 2. Likewise, commands for accents and diacritical marks are replaced by their results if it belongs to the Unicode character set, so '`Preu{\ss}ens`' is replaced by '`Preußens`' in Figure 3. Table 3 lists LATEX commands MLBIBTEX's parser replaces by their effect. Since most of these commands are unknown for ConTEXt end users, we make precise corresponding characters. Besides, we can see that our parser not only gets values associated with an entry's successive fields, but also structures them as far as possible. An accurate example is given by person names, for fields such as `AUTHOR` or `EDITOR`.

1.4 A bridge towards the biblatex package

MLBIBTEX is also able to recognise some fields introduced by the `biblatex` package. If this package is used, the fields `YEAR` and `MONTH` are superseded by the `DATE` field. A *simple date* has the format, closer to the ISO[20] format:

```
simple-date ::= yyyy[-mm[-dd]]
```

[20] **I**nternational **S**tandardisation **O**rganisation.

```
<book id="cussler-du-brul2010" language="english">
  <author>
    <name>
      <personname>
        <first abbrev="Cl.">Clive Eric</first>
        <last>Cussler</last>
      </personname>
    </name>
    <with/>
    <name>
      <personname>
        <first>Jack B.</first>
        <last>Du Brul</last>
      </personname>
    </name>
  </author>
  <title>The Silent Sea</title>
  <publisher>Penguin Books</publisher>
  <series><emph>Oregon</emph>
             Files Adventures</series>
  <year>2010</year>
  <month><mar/></month>
</book>
```

Figure 3: XML trees resulting from Figure 1's examples.

where `y`, `m`, `d` are digits and square brackets are put for optional parts[21]. The value associated with this DATE field may be a simple date, or a date range (`<simple-date>/<simple-date>`); a simple date ending with a '/' character (`<simple-date>/`) means an open-ended date range. If ranges are used within .bib files whilst MLBIBTEX applies a 'classical' bibliography style, only the first date is considered. Other additional fields used by `biblatex` are recognised by our `mlbibcontext` program: some are known by ConTEXt's `bib` module and have already been listed in Table 1, let us add the GENDER field, already introduced by the `jurabib` package.

To end up with this § 1., the `mlbibcontext` program allows ConTEXt users to deal with .bib files, and most new fields introduced by recent evolution of bibliography processing. The only limitation is that LATEX commands should be used within .bib files, if additional markup is needed.

[21] In fact, MLBIBTEX deals with an extension of this syntax where negative years are allowed. More details are given in § 3.1.

```
<book id=”wienfort2008” language=”german”>
  <author>
    <name>
      <personname>
        <first>Monika</first>
        <last>Wienfort</last>
      </personname>
    </name>
  </author>
  <title>Geschichte Preußens</title>
  <publisher>
      Verlag C. H. Beck
  </publisher>
  <volume>2456</volume>
  <series>Wissen</series>
  <address>
    <group language=”german”>
      München
              </group>
    <group language=”english”>
      Munich
    </group>
    <group language=”french”>
      Munich
    </group>
  </address>
  <year>2008</year>
  <month><aug/></month>
</book>
```

Figure 4: XML trees resulting from Figure 1’s examples.

2. How MLBIBTEX works

Broadly speaking, the successive tasks performed by a bibliography processor are chained as follows:

- decide which bibliography style is to be applied, either because it is explicitly specified in the document’s source file, or because a general structure is passed to the typesetting system, as done by the `bib` module (*cf.* § 1.2) and in LATEX with the packages `jurabib` [2] or `biblatex`;
- collect *citation keys*, or decide that every entry must be retained;
- parse .bib files;

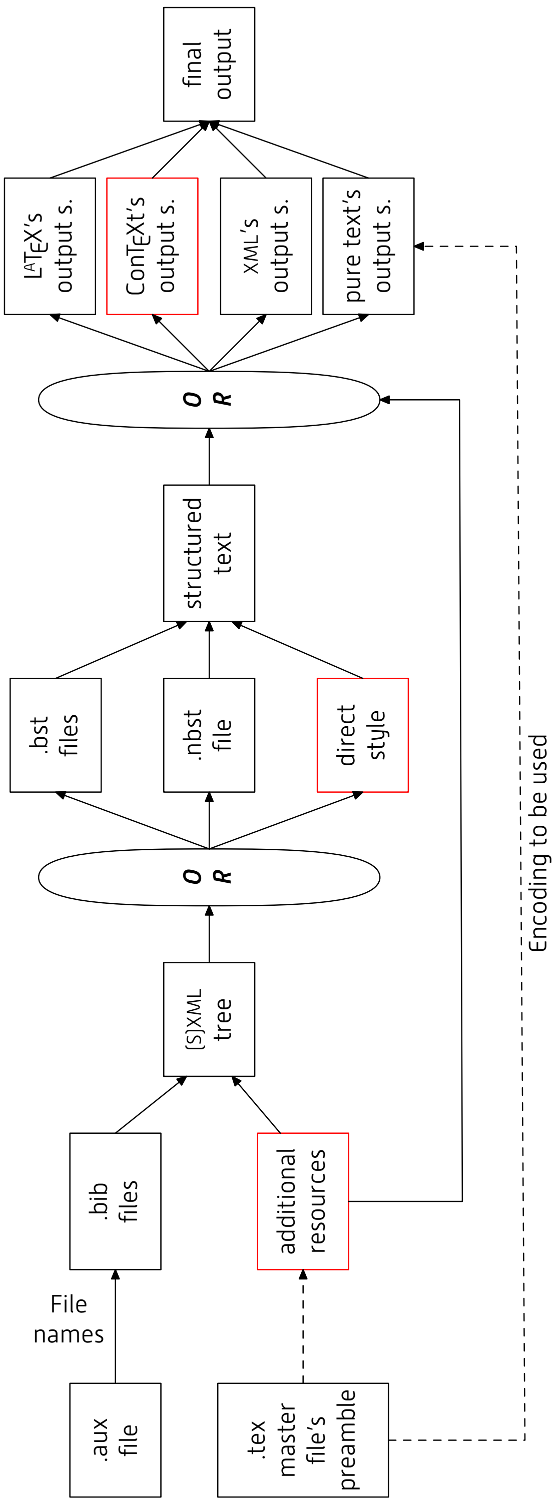

Figure 5: MLBIBTEX's flow.

- sort entries, if need be;
- arrange each entry to the corresponding reference.

In comparison with BIBTEX's process — mainly controlled by .bst files — MLBIBTEX's is more complicated since many additional aspects are managed. In this section, we detail this *modus operandi* from a general point of view, because we would like to emphasise that our support for ConTEXt is easily integrated into this general framework. In this section's end, we will get back to the features that allow MLBIBTEX to work for generating bibliographies for ConTEXt documents. Since ConTEXt developers might be interested in performing separate parts of MLBIBTEX's behaviour, we sometimes make precise the Scheme functions to be used. As mentioned above, they could be invoked by means of external calls performed by Lua functions. Let us go back to this complete *modus operandi*, pictured in Figure 5. As mentioned in § 1.3, parsing entries results in a tree that can be viewed as an XML tree. In fact, we use the SXML[22] format: roughly speaking, XML elements are implemented by Scheme linear lists, textual data by strings, and attributes of an elements by association lists. If we glance at Figure 5, we get such a tree and apply a bibliography style. Since the result is not a string, but a list of strings and elements to be concatenated — examples of such elements (`emph`, `LaTeX-command`, `LaTeX-math-mode`) are given in § 1.3 — we view this result as a kind of *structured text*[23]. The last step may be viewed as *serialisation* process according to output conventions, the final output being written onto an output port.

2.1 Getting an (s)xml tree

Like BIBTEX, MLBIBTEX reads an .aux file in order to get *citation keys*, and .bib *file names*. Let us recall that ConTEXt only builds .aux files when the `bib` module is used, since only BIBTEX deals with such files when ConTEXt is in action. MLBIBTEX can perform these two actions directly, by means of the following Scheme expressions — `filename-list` is a list[24] of .bib file names, `key` a citation key, and `key-list` a list of citation keys; file names and citation keys are given as strings — :

- `((bibtexkey-alist-pv 'add-keys) key-list)` causes citation keys to be arranged according to the reverse order of the first occurrences of each key, without duplicates; in addition, this expression creates *placeholders* for these entries[25]; in addition, notice that:
  - `((bibtexkey-alist-pv 'add-key) key)` can be used to insert keys one by one;
  - if you would like each item of .bib files to be retained[26], just run:
    `((bibtexkey-alist-pv 'extend))`

  these last three expressions return `#t`, the 'true' value in Scheme;

[22] **S**cheme implementation of XML.

[23] This notion may be related to *mixed* contents for XML elements.

[24] That is, a linear list of Scheme: '(… …)'.

[25] This is the basic form. If *namespaces* are used, in order to solve *name conflicts* among different items belonging to different .bib files [15], another expression must be used.

[26] As done in LATEX by the `\nocite{*}` command.

- `((bibliographyfile-list-pv 'adjoin-all) filename-list)` memoizes the pathnames of `.bib` files, provided that no pathname is repeated; this expression must be performed once[27]; the result is `#t` if this operation succeeds, `#f` — the 'false' value in Scheme — otherwise.

When MLBIBTEX is used 'classically', such an .aux file also gives the bibliography style to be applied. Once information is extracted from this .aux file, .bib files are parsed in turn, and resources associated with citation keys are updated. Let us give some additional details about this process. The list of .bib file names is the result of:

```
((bibliographyfile-list-pv 'get))
```

Let `filename` be a pathname, given as a string, and `filename-list` a list of strings, parsing a .bib file is launched by:

```
(s-parse-bib-file filename)
```

In fact, such an expression is mainly used for debug purposes. To look for all the members of `filename-list` using the kpathsea library [16], and parse them in turn, run:

```
(s-parse-kpbib-filename-list filename-list)
```

These last two expressions return `#t` (resp. `#f`) in case of success (resp. failure). When all the .bib files are parsed, the following expression checks if every citation key has been associated with a resource:

```
(get-sxml-mlbiblio-tree)
```

and results in `#f` if no entry has been found, otherwise the result is the SXML tree for all the entries built, *unsorted*, that is, according to the order of first citations throughout the document.

The box labelled by 'Additional resources' in Figure 5 encompasses the information not included into `.aux` files. An example of such an additional resource is given by *preambles* — @PREAMBLE (resp. @CONTEXTPREAMBLE) for bibliographies for LATEX (resp. ConTEXt) documents — as mentioned in § 1.3. Other information is exploited at subsequent steps and may induce the parsing of a LATEX document's preamble. For example, the babel package allows LATEX to be conformant to typographic rules of many languages other than English [2]. But when this package is loaded, end-users must put all the languages they may use, so some languages are available throughout a document, some not[28]. MLBIBTEX looks for this information within a .tex source's preamble, and also gets the main language[29], whereas this information has to be provided when the `mlbibcontext` program is called. Likewise, the encoding to be used for a .bbl file — UTF-8 for ConTEXt's recent versions — is given by the `inputenc` package's option, this information being included into a LATEX document's preamble.

[27] This expression is suitable when .bib file names are specified within a unique order, by the `*bibliography` command in LATEX or the `\setupbibtex` command in ConTEXt. If .bib file names may be entered one by one, as done by the `biblatex` package's `*addbibresource` command, another function is to be used.

[28] This feature does not exist within ConTEXt: all the languages, denoted by ISO codes, are available by means of the '`\language[...]`' construct.

[29] When the `babel` package is used, the main language is this package's last option.

```
(define next-value
  (let ((counter 0))  ;  May be viewed as a private
                      ;  attribute.
    (lambda ()        ;  Method incrementing the counter.
      (let ((result counter))
        (set! counter (+ counter 1))
        result))))
```

Figure 6: Lexical closure in Scheme.

2.2 Applying a bibliography style

W.r.t. MLBIBTEX's terminology, a *bibliography style* is a *function* applied to an SXML tree of bibliographical entries and resulting in a *structured text*, this notion being introduced at this section's beginning. Let us recall that MLBIBTEX is written using a functional programming language, so such a function can be the result of some *computation*. For example, when MLBIBTEX works in compatibility mode, a .bst file is 'compiled' into Scheme expressions, and a Scheme function launches the process. Other bibliography styles taking advantage of MLBIBTEX's multilingual features are written in `nbst`[30], a variant of XSLT[31], also interpreted by Scheme functions. In both cases, these programs build source files for LATEX and implement 'particular' styles, that is, `alpha`, `plain`, *etc*. For sake of efficiency, a different technique is used when resulting .bbl files are marked up with LATEX or ConTEXt commands: a *direct style*, wholly written in Scheme, is applied. Let us remark that applying small changes into a .bst or .nbst styles is easy — which is an advantage of these languages — but such a feature is irrelevant about a direct style since customisation is performed by redefining commands used for markup. MLBIBTEX's direct styles include `mlbiblatex`, for .bbl files suitable for the `biblatex` package and `mlbibcontext`, for bibliographies of ConTEXt documents.

An important point: if you try to process bibliographical entries in turn, each being processed independently of others, by means of external calls, such a process may result in strange-looking references. In fact, when an entry is processed, information about the following entries may be *updated*. In other words, the following entries, not processed yet, should never be dropped out. This feature is related to *lexical closures* in Scheme. Let us consider Figure 6: the first call to the `next-value` function returns `0`, then each call returns the next integer. More precisely, the function returns the current value of the local variable `counter` and updates this variable for the next call. The `counter` variable's value is bound to the definition of the `next-value` function, so a change of this variable impacts the function's next calls.

The same technique is used for bibliographical entries when the `mlbibcontext` program is applied: each entry $[E_0, E_1, \ldots, E_n]$ has its own additional environment $[c_0, c_1, \ldots, c_n]$, as pictured in Figure 7. Entries *before* the current one are already processed and are no longer searched, no longer updated. On the contrary, the

30 **N**ew **B**ibliography **ST**yles.

31 e**X**tensible **S**tylesheet **L**anguage **T**ransformations, the language of transformations used for XML texts.

enviroments associated with entries *after* the current one may be searched and updated.
More precisely, if you are familiar with CPS[32], here is the call of the function processing an entry when the `mlbibcontext` program is applied:

```
(mlbibcontext-process-entry-cps
   entry-sxml-subtree current-s
   current-s-length current-index output-p
   k3)
```

where `entry-sxml-subtree` is the current entry's SXML tree, `current-s` is the longest string used to label a reference and `current-s-length` its length, `index` is the current entry's index in the complete list, `output-p` is the output port where results will be finally written. At the end, the `k3` continuation — a three-argument function — will be applied to the longest string and its length, possibly updated after processing the current entry. The third value is a *thunk*[33] memoizing the result of processing the current entry. This `k3` continuation expresses how processing next entries is launched after updating them. When all the bibliographical entries are processed, successive thunks are applied, in turn.

2.3 Getting a final text

As mentioned above, the last step starts from a *structured text*, that is, a list whose members are strings or SXML elements. These elements are flattened into strings, and all the strings are concatenated. Our Scheme function performing this task is:

```
(o-process-wrt-context-mode str-text)
```

where `str-text` is a structured text, the result is a string.

2.4 Dealing with other formats

Let us recall that our internal format may be viewed as an XML document. Since ConTEXt can deal with such syntax, saving an SXML tree of bibliographical entries into a file using XML syntax may be of interest:

```
(sxmlh-get-xml-syntax  sxml-tree filename
                       encoding)
```

where `sxml-tree` is an SXML tree[34]. The last two arguments are optional: if `filename` is omitted or bound to the `screen` symbol, the result is just displayed, if `encoding` is omitted, the output encoding is `Latin-1`. This function returns `#t`.
If bibliographical entries are given using XML syntax, according to our organisation, use the following function to get the corresponding SXML tree:

```
(sxmlh-parse-xml-file filename)
```

[32] **C**ontinuation-**P**assing **S**tyle. The last argument of a function using this technique is a continuation, it expresses that this function's *future* is the application of this continuation.

[33] W.r.t. Scheme's terminology, a *thunk* is a zero-argument function.

[34] Our `sxmlh-get-xml-syntax` function adds blank nodes in order for the result to be displayed nicely, especially if it is an XML tree of bibliographical entries. Another function should be used if you would like XML syntax, but as it is, without additional blank nodes.

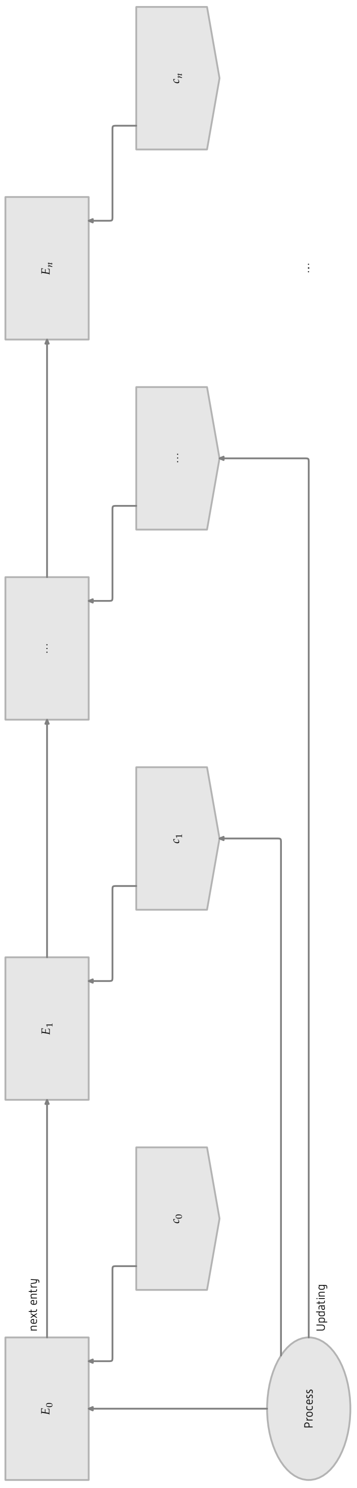

Figure 7: How to chain entries' processes.

or the `#f` value if the contents of the `filename` file is not well-formed. As other possible imports, we plan to implement a parser for bibliographies expressed using the Refer format. Last we also mention that MLBIBTEX includes a converter from JSON[35] texts into trees, although it seems to us that JSON is not really used for bibliographical entries.

3. Features of interest

In this section, we show some MLBIBTEX features of whose interest is not limited to ConTEXt. Some may require additional development of this typesetting system, most of them have not been implemented by BIBTEX or biber.

3.1 Type-checking

In general, when BIBTEX end-users try MLBIBTEX, they are very surprised because the latter is less permissive than the former about values associated with fields throughout .bib files. A very simple example is given by possible values of the `YEAR` field: by default[36] it *must* be a number, different from zero[37], written without the '+' sign, negative numbers are allowed, too. This point may be viewed as a drawback, since some .bib files that can be read by BIBTEX without problem are unexploitable by MLBIBTEX. But the question could be: 'BIBTEX is not too permissive, is it?' If end-users put something other than number as a `YEAR` field, BIBTEX could sort entries because it only performs lexicographic sorts, the `YEAR` field's value being a substring of the sort key built. So `3` is greater than `2016` if such an order is used. Such a comparison does not occur in practice and could be easily solved by changing '`3`' to '`0003`'. However, we think that is bad technique: numerical sorts should be performed wherever they are accurate, in particular for years and months. As a consequence, we have to check that corresponding data are well-formed. Besides, this notion of data type can be noticed within the `biblatex` package's documentation. In addition, we personally experienced `.bib` files where mistakes within `AUTHOR` fields were discovered several years (!) after establishing the entry. Likewise, some mispelled optional fields such as `EDITORS` were unremarked for a long time. On the contrary, MLBIBTEX performs type-checking when it reads .bib files. If some fields are unknown, they are put at the end of an entry's internal form — so no information is lost — and a message warns users. This notion of unknown field depends on the executable program you are using: if it is `mlbibtex`, an additional field of the `bib` module is supposed to be unknown, if `mlbibcontext` is called, such a field is known.
We mention that customising the type-checking function associated with a field is quite easy. To be honest, such an operation should be done by an experienced Scheme programmer, but is not really difficult.

3.2 Directives

MLBIBTEX allows *directives* at the beginning of a `.bib` file. Presently, there are two directives, values associated with such a directive are bounded on the right by the end-of-line character:

[35] **J**ava**S**cript **O**bject **N**otation.

[36] We are going back on this point in § 3.3.

[37] Let us recall that this was *no* year zero, the year after 1 BC (**B**efore **C**hristi) was 1 AD (***A****nno* ***D****omini*).

```
%encoding  =  ...
%prefix    =  ...
```

An example of the `%encoding` directive has been given in Figure 1. This directive may be useful for a human agent, and if the operating system difficultly determines the encoding used. Let us make precise that even if several .bib files are to be processed, each can specify its own encoding[38].

The `%prefix` directive aims to solve *name conflicts* when there are namesake entries within different `.bib` files. If all the entries of .bib files have different citation keys, that is, if all the citation keys are unambiguous, this `%prefix` directive can be ignored. Adding a prefix to an ambiguous key requires the `\cite[…]` command's syntax to be extended. We did that in a package for LATEX [15].

3.3 Inexact information

MLBIBTEX can also deal with *inexact* information, what is suitable for *ancient* works. Let us consider the following work:

Σοφοκλῆς: *Ἀντιγόνη*. 441 BC, approximately.

that is, *Antigone*, by Sophocles (495 BC–406 BC). Of course, this reference does not denote a bibliographical item in the sense that it is not a published book. But a written document may cite several modern translations of this work, so putting the original title separately and accessing it by means of cross references may be interesting. This tragedy premiere's was presumably held in 441 BC, but this date is uncertain: some historians believe that it took place before. Of course, the problem is to rank such an entry when a bibliography is sorted.

Since the interest of such entries is purely historical, we chose to allow such *inexact* information only if the `-inexact` option is enabled. So, such an inexact year may be denoted by '`ca-441`'. Some digits at the end of an inexact year may also be replaced by question marks — e.g., '`-4??`'. In this last case, an accurate bibliography style should put down '5th century BC', so additional development may be required. We can also express that an author is *unknown* by a pair of question marks — e.g., '`?? and Alan Braslau`' — such a pair just before a name means that the identity is merely surmised — e.g., '`?? Hans Hagen`'. The order relations used by MLBIBTEX to sort years and authors' names have been enlarged in order to process such inexact information [17].

3.4 Interface with Scheme

In MLBIBTEX's first publicly released version, it was possible to add new definitions in Scheme, or redefine some functions by more accurate versions, but in practice, this ability was only used by people able to recompile the whole program. In next versions, programs derived from MLBIBTEX will look for *initialisation files* in users' home directories. More precisely, for UNIX-based systems:

`mlbibtex`	$\Longleftarrow$	˜/.mlbibtex
`mlbibcontext`	$\Longleftarrow$	˜/.mlbibcontext

These Scheme source files could add new definitions or redefine some default conventions. For example, the default input encoding for .bib files is `Latin 1`,unless an

[38] However, the first release of MLBIBTEX's new version will provide only byte-based encodings, that is, `Latin-1`, `Latin-2`, UTF-8, but not UTF-16, which will be available later.

`%encoding` directive specifies another. If you would like this two default encoding to be UTF-8, just put the following Scheme expression in your initialisation file:

```
((encodings-pv 'set-default-for-4-files)
 'utf-8)
```

This feature can also be used to introduce new sort procedures, as described in [18] and bind them to natural languages. For example, MLBIBTEX's library provides two lexicographic orders for the Dutch language: `<dutch-basic?`, by default, and `<dutch-ijs?`. To associate the latter with this natural language, run:

```
(c-language->order-relation  "dutch"
                             <dutch-ijs?)
```

To end up with this interface, let us mention that MLBIBTEX's library includes functions generating unambiguous labels when `alpha` styles are used. For example:

`(generate-newly` s_1 s_2 `y)`

associates the s_1 string with the SXML tree y if s_1 is unambiguous, and uses s_2 to enrich s_1 otherwise. In any case, the string associated with y is returned. Let us consider that only the y_1 sxml tree is the only entry whose author is Taco Hoekwater, whereas the author of the SXML trees y_2 and y_3 is Willi Egger. The year of these three entries is 2015. The results of three successive calls of this `generate-newly` is given in Table 4. The three other sessions are analogous, but show the second argument's interest.

4. Executable programs

When MLBIBTEX is installed, Scheme definitions are *compiled* and executable files are built. Among them, we have already mentioned the programs `mlbibtex` and `mlbiblatex`. Hereafter we give more details about the programs `mlbibcontext` and `mlbibtex2xml`. Let us mention that all these programs can be invoked with the option `-h` or `--help`, in which case a short description of arguments and options is displayed. Hereafter we use square brackets for optional parts and ellipses for repetitions. The `mlbibcontext` program is to be invoked as follows:

```
mlbibcontext [-inexact] job-name[.aux] \
  key-expr lg-code
```

where the `-inexact` option allows the use of inexact information (*cf.* § 3.3), all the other arguments are strings:

- `job-name` is a pathname for an .aux file, as in BIBTEX, the .aux suffix being implicit;
- `key-expr` gives successive sort keys, according to the pattern `(<l>[!][\[<value>\]])`…:

`(generate-newly "Hoekwater 2015" "a" y1);`
`     ⟹"Hoekwater 2015"`
`(generate-newly "Egger 2015" "a" y2);`
`     ⟹"Egger 2015"`
`(generate-newly "Egger 2015" "a" y3);`
`     ⟹"Egger 2015a"`
	`a' (resp. `A') means `numbering with lower-case (resp. upper-case) letters.

`(generate-newly "Hoekwater 2015" "+a" y1);`
`     ⟹"Hoekwater 2015"`
`(generate-newly "Egger 2015" "+a" y2);`
`     ⟹"Egger 2015a"`
`(generate-newly "Egger 2015" "+a" y3);`
`     ⟹"Egger 2015b"`
	`+' at the beginning of the control argument causes the first argument not to be used without suffix if this string is shared.

`(generate-newly "Egger 2015" "-1" y2);`
`     ⟹"Egger 2015"`
`(generate-newly "Egger 2015" "-1" y3);`
`     ⟹"Egger 2015-1"`
`(generate-newly "Hoekwater 2015" "-1" y1);`
`     ⟹"Hoekwater 2015"`
	`1' means `numbering with Arabic digits'. Let us remark that the `+' character is not put at the control argument's beginning.

`(generate-newly "Hoekwater 2015" "+-i" y1);`
`     ⟹"Hoekwater 2015"`
`(generate-newly "Egger 2015" "+-i" y2);`
`     ⟹"Egger 2015-i"`
`(generate-newly "Egger 2015" "+-i" y3);`
`     ⟹"Egger 2015-ii"`
	`i' (resp. 'I') means `numbering with lower-case (resp. upper-case) Roman numerals'. All the other characters are put into the suffix.

Table 4: Four examples of sessions using our `generate-newly` function.

  - `<l>` is a letter among '`m`' for '**M**onth' , '`n`' for '**N**ame' (person name as an author or editor), '`t`' for '**T**itle', '`y`' for '**Y**ear', all the other signs are ignored;
  - if '`!`' is present, the corresponding sort is performed using descending order;
  - a value surrounded by square brackets ('`\[<value>\]`') is a default value to be used whenever an optional sort key does not apply.

  For example, '`y!m![0]`' denotes a sort by reverse chronological order, items without month information being ranked after item with expressed month for the same year[39]; there is no default order relation[40]: the list of bibliographical items is left unsorted unless a sort is specified[41];

- `lg-code` is the ISO code of the document's main language.

Previous versions of this program included a fourth argument for the output encoding, no longer allowed since ConTEXt's recent versions deal with UTF-8.

[39] Let us recall that the `@MONTH` field is optional.

[40] The default order relation used by both BIBTEX and biber would be specified by '`ynt`'.

[41] Dealing with more elaborate sort procedures is possible, but needs to use Scheme functions.

The `mlbibtex2xml` program allows .bib files to be converted into XML files and can be run as follows:

```
mlbibtex2xml [-inexact] [<dest>] \
  f0[.bib] f1[.bib] …
```

where the `-inexact` option allows the use of inexact information, `f`$_0$`[.bib]`, `f`$_1$`[.bib]`, … are .bib files, the .bib suffix being implicit. The <dest> information is organised as follows:

```
<dest> ::
  (-screen | -o output) [-encoding encoding]
```

If the `-screen` option is used, the result is displayed at the screen, otherwise it is written into a file. If the `-o` option is used, `output` gives the output file name, otherwise, this name defaults to `f`$_0$`-mlbiblio.xml`, even if several .bib files are processed. The output encoding defaults to `Latin-1`.

5. Discussion and conclusion

If we examine the evolution of the typesetting systems built out of TEX, we may notice that use of the Lua programming language allows tasks related to actual programming to be put into action by means of an 'actual' programming language, whereas engines based on TEX remain wonderful tools for typesetting texts. In other words, programs such as LuaLATEX or ConTEXt format texts and *delegate* some tasks to Lua functions. Even if programming a sort procedure using TEX's language is possible, implementing the same operation is obviously easier using Lua. A comparison between BIBTEX's *modus operandi* and MLBIBTEX's shows the same trend: some operations more related to 'actual' programming — e.g., multi-criteria sorting algorithms — can be more easily implemented in MLBIBTEX by means of Scheme definitions, in comparison with the expressive power of the language of BIBTEX's .bst files.

Within this framework, which advantages are provided by a *functional programming* language in general and by Scheme in particular? In such a language, we can easily write an expression such as:

```
(lambda (f2) (f2 2 1))
```

which may be viewed as follows: a procedure — denoted by `f2` — will be applied to the numbers 1 and 2, but presently, we do not know *which* procedure. That is, *data* are known, but the *way* to process them is not. If we decide to perform an addition (resp. subtraction), just present + (resp. −). There is something equivalent when bibliographical references are built from entries: *resources* are known, they are successive bibliographical items; the operations performed by a bibliography style depend on typographical rules, they may also depend on natural languages, e.g., when a month name is to be displayed. In any case, a bibliography style may be abstracted by a *function*, even if this function is itself built from many parameters. We have followed such a guide-line when MLBIBTEX's direct styles were developed.

A debatable point is given by the syntactic extensions to .bib files' format. The main criticism is that end-users cannot be compatible with the original format, they cannot revert to 'old' BIBTEX, either, if using these extensions[42]. That may be needed when users put the final version of an article, processed automatically by LATEX and BIBTEX on journals' Web sites. However, we think that the .bib format is too restrictive. As mentioned in § 1.1, the only way to express some cases is the use of dummy LATEX commands. Even if we can understand that at a time when source LATEX files were BIBTEX's only target, it seems to us that putting:

```
AUTHOR = {{\relax Cl}{ive Eric} Cussler}
```

in order for this name to be correctly abbreviated by 'Cl. Cussler' is nothing less than a *dirty trick*! A compatible solution might be the addition of new comma-separated subfields, something like:

```
Cussler,, Clive Eric, Cl.
```

— the *Junior* part being empty — but unfortunately such syntax is rejected by BIBTEX. Broadly speaking, our extensions are debatable, but we think that keeping this .bib format as it is will leave many problems unsolved. Besides, many graphical interfaces now exist in order to fill in the information of a BIBTEX entry. End-users could interactively enter the components of a person name, the tool being in charge of assembling these components according to a new syntax with more expressive power. On another point, it seems to us to be regretable that each new tool has developed its own extensions. For example, the biblatex package uses a PAGETOTAL field for the total number of the pages of a book, whereas this information is often known as a TOTALPAGES field, in particular by the jurabib package. Many similar examples exist. Maybe we can consider that we are living an *experimentation* period, which will result in new bibliography processors and new formats for bibliographical entries. Within this framework, developing a program that would allow ConTEXt to take advantage of BIBTEX's bibliography database files is an interesting challenge for us.

As last words, let us say that MLBIBTEX has its advantages and drawbacks, but it is ready to cooperate with ConTEXt.

Acknowledgements

My first thanks are for Alan Braslau, who was waiting for this article for a long time and who helped me a lot, when I was just a beginner for some advanced features of ConTEXt. I am also grateful to Hans Hagen and Willi Egger for nice assistance.

References

[1] O. Patashnik, *BIBTEXIng* (1988).

[2] F. Mittelbach, M. Goossens, J. Braams, D. Carlisle, C. A. Rowley, C. Detig, and J. Schrod, *The LATEX Companion*, 2 ed. (Addison-Wesley Publishing Company, 2004).

[3] CONTEXTGARDEN, *Bibliographies in MkII* (2012).

[4] O. Patashnik, *Designing BIBTEX Styles* (1988).

[42] The biblatex package has the same drawback: if the DATE field (*cf.* § 1.4) is used rather than the standard fields YEAR and MONTH, it will be unrecognised by BIBTEX's standard bibliography styles.

[5] P. Lehman, P. Kime, A. Boruvka, and J. Wright, *The biblatex Package. Programmable Bibliographies and Citations. Version 2.9a* (2014).

[6] P. Kime and F. Charette, *Biber. A Backend Bibliography Processor for biblatex. Version biber 1.9 (biblatex 2.9)* (2014).

[7] J.-M. Hufflen, From MLBIBTEX 1.3 to 1.4, In T. Przechlewski, K. Berry, B. Jackowski, and L. B. Ludwichowski (Eds.) Various Faces of Typography. Proc. BachoTEX 2015 conference (2015).

[8] J.-M. Hufflen, Using MLBIBTEX to Populate Open Archives, In T. Przechlewski, K. Berry, G. Gic-Grusza, E. Kolsar, and L. B. Ludwichowski (Eds.) Typographers and Programmers: Mutual Inspirations. Proc. BachoTEX 2010 Conference (2010).

[9] J.-M. Hufflen, Demonstration of the `mlbibcontext` Program, In Proc. 6th ConTEXt Meeting & EuroTEX 2012 (2012).

[10] R. Ierusalimschy, *Programming in Lua*, 2 ed. (Lua.org, 2006).

[11] J.-M. Hufflen, A Comparative Study of Methods for Bibliographies, TUG*boat* **32** (2011). (Proc. TUG 2011 conference, Trivandrum, India)

[12] J.-M. Hufflen, Managing Languages within MLBIBTEX, TUG*boat* **30** (2009).

[13] J.-M. Hufflen, Names in BIBTEX and MLBIBTEX, TUG*boat* **27** (2006). (TUG 2006 proceedings, Marrakesh, Morocco)

[14] J.-M. Hufflen, MLBIBTEX's Version 1.3, TUG*boat* **24** (2003).

[15] J.-M. Hufflen, Managing Name Conflicts and Aliasing with MLBIBTEX, In T. Przechlewski, K. Berry, B. Jackowski, and L. B. Ludwichowski (Eds.) What Can Typography Gain from Electronic Media? Proc. BachoTEX 2014 conference (2014).

[16] TUG Working Group on a TEX Directory Structure, *A Directory Structure for TEX Files. Version 0.9995* (TUG Working Group on a TEX Directory Structure, 1998). (CTAN: tex/archive/tds/standard/tds-0.9995/tds.dvi)

[17] J.-M. Hufflen, Dealing with Ancient Works in Bibliographies, *ArsTEXnica* **18** (2014). (In Proc. GUIT meeting 2014)

[18] J.-M. Hufflen, MLBIBTEX And Its New Extensions, In Proc. 6th ConTEXt Meeting & EuroTEX 2012 (2012).