Tame the BeaST

The B to X of BibTEX

Nicolas Markey
markey@lsv.ens-cachan.fr

Version 1.3 – October 16, 2005

This 45-page tutorial presents and explains, as clearly and exhaustively as possible, what BibTEX can do. Indeed, BibTEX manuals, essentially two documents by its author [Pat88a, Pat88b] and chapters in some LATEX books [Lam97, GMS93, MGB+04, ...], are often short and incomplete.
The capital letters "BST" in the title represent the standard extension of BibTEX style files. "B to X" means that I tried to be as complete as possible. Don't hesitate to e-mail me you TEXnical as well as (mis)spelling remarks.

Contents

Part 1

Basic bibliography with LaTeX

Table of Contents

BibTeX is often seen as something magical, as well as LaTeX bibliography-related commands in general. Many users simply copy and paste classical examples without trying to understand how it works. But in fact, a bibliography in LaTeX is nothing but a *list* of *references* mentioned in a document. If we had to do it "by hand", without knowing anything about the `thebibliography` environment, it could look like this:

```
This is the main matter of the document, mentioning
[\ref{doc1}] and [\ref{doc2}], for instance.
\section*{References}
\begin{enumerate}
  \renewcommand\labelenumi{[\theenumi]}  %% numbers are surronded with brackets
  \item \label{doc1} Michel Goossens, Franck Mittelbach and Alexander
        Samarin, \emph{The \LaTeX{} Companion}, Addison Wesley, 1993.
  \item \label{doc2} Leslie Lamport, \emph{\LaTeX: A Document Preparation
        System}, Addison Wesley, 1997.
\end{enumerate}
```

which, when compiled, gives

This is the main matter of the document, mentioning [1] and [2], for instance.

References

[1] Michel Goossens, Franck Mittelbach and Alexander Samarin, *The LaTeX Companion*, Addison Wesley, 1993.

[2] Leslie Lamport, *LaTeX: A Document Preparation System*, Addison Wesley, 1997.

This is mostly what the `thebibliography` environment does (it starts a `list` environment, similar to `enumerate`), `\bibitem` corresponding to the `\item` commands. One major difference is that `\bibitem` allows for more general cross-references than `\item` and `\label` (for example, one can cite [GMS93]).

This is what this part deals with: "Writing a bibliography *without* BibTeX". This is not the main goal of this manual, but it is a cornerstone for understanding the sequel.

1 The `thebibliography` environment

The `thebibliography` environment is not defined in LaTeX itself (neither is it in TeX[1], of course). It has to be defined by the class file used in the document (*e.g.* article.cls or book.cls). As mentioned earlier, it is a `list`-like environment inside a new `\section` (or `\chapter`, depending on the document class[2]). This list has to be set up carefully, however, in order to avoid indentation problems. For instance, if we put "alphanumerical" labels in the previous example, we would get:

References

[GMS93] Michel Goossens, Franck Mittelbach, and Alexander Samarin, *The LaTeX Companion*, Addison Wesley, 1993.

[Lam97] Leslie Lamport, *LaTeX: A Document Preparation System*, Addison Wesley, 1997.

In order to avoid this problem, `thebibliography` has a mandatory argument, which should be the largest label occurring in the list. This will allow to set up margins properly.

Let's now have a look at the precise definition of the `thebibliography` environment (as defined in the article.cls class, for instance):

```
\newenvironment{thebibliography}[1]
     {\section*{\refname
        \@mkboth{\MakeUppercase\refname}{\MakeUppercase\refname}}%
```

As mentioned earlier, `thebibliography` starts a new section[3]. The environment also sets the page headers[4].

```
      \list{\@biblabel{\@arabic\c@enumiv}}%
           {\settowidth\labelwidth{\@biblabel{#1}}%
```

This is not surprising either: We start a new `list` environment[5]. It takes two mandatory arguments:

- the first one (`\@biblabel{...}`) defines the format of the default command for generating labels. Here it is defined with the `enumiv` counter, using `\@biblabel`[6]. Thus we have [1], [2], ... Since `\bibitem` is a sort of `\item`, it may have an optional argument for modifying this default label.

[1] BibTeX may be used with plain TeX, you'll simply have to input the btxmac.tex package

[2] This is the reason why `thebibliography` has to be defined in the class files. The other bibliographic commands are defined in the standard LaTeX format.

[3] It is in fact a `\section*`, so that it won't appear in the table of contents. In order to cleanly insert the bibliography in your table of contents, use the tocbibind.sty package. Other methods you can think of will probably lead to wrong page numbers.

[4] The standard apalike.sty package hard-codes the headers to "REFERENCES" or "BIBLIOGRAPHY", depending on the class file. But another package (with the same name, unfortunately) correctly sets the headers to `\refname` or `\bibname`. Simply check in the sources if you have problems for redefining the headers with that package.

[5] The `\list` command is equivalent to a `\begin{list}`, and has to be followed with a `\endlist`.

[6] `\@biblabel` is defined in LaTeX. It outputs its argument surrounded with brackets. The precise definition is: `\def\@biblabel#1{[#1]}`.

- The second argument(lines 5 above to 11 below) is a set of commands that are run at the beginning of the environment. They set the values of different lengths and parameters of the `list` environment. This is where we need the longest label (which the mandatory argument of the `thebibliography` environment should be set to), in order to correctly indent the whole list. People often write `\begin{thebibliography}{99}`, but this is correct only if there are between 10 and 99 cited references (assuming all digits have the same length, which is the case with the `cmr` fonts).

The rest of the definition of `thebibliography` contains some borderline definitions for the list environment and the use of the `enumiv` counter:

```
6              \leftmargin\labelwidth
7              \advance\leftmargin\labelsep
8              \@openbib@code
9              \usecounter{enumiv}%
10             \let\p@enumiv\@empty
11             \renewcommand\theenumiv{\@arabic\c@enumiv}}%
```

`\@openbib@code`, which is empty by default, allows to modify some parameters if necessary. Option `openbib` of classical style files uses this command for resetting some parameters. Line 9 to 11 set the list counter.

Last, within the reference list, spacing rules as well as some special penalties are used:

```
12       \sloppy
13       \clubpenalty4000
14       \@clubpenalty \clubpenalty
15       \widowpenalty4000%
16       \sfcode`\.\@m}
```

That's it for the initialization of this environment. Ending the `thebibliography` is much easier: we just echo a warning if no reference has been included, and we close the `list` environment:

```
17      {\def\@noitemerr
18        {\@latex@warning{Empty `thebibliography' environment}}%
19       \endlist}
```

2 The `\bibitem` command

Inside the `list` environment described above, we have to insert `\items`, as is usual. It will be a special `\item`, though, in order to have a correct rendering of each bibliographical item. The adequate command is named `\bibitem`, and has two roles: Writing the new entry in the list, and defining the cross-reference to be used when citing this entry, which defaults to `\@biblabel{\@arabic\c@enumiv}`. The result is [1] for instance, but of course can be modified into [GMS94], say, with an optional argument, exactly in the same way as for an `\item`. Here is the precise definition of `\bibitem`:

```
\def\bibitem{\@ifnextchar[\@lbibitem\@bibitem}
```

`\bibitem` calls `\@lbibitem` if there is an optional argument, and `\@bibitem` otherwise. Those auxiliary commands are defined as follows:

```
\def\@lbibitem[#1]#2{\item[\@biblabel{#1}\hfill]\if@filesw
      {\let\protect\noexpand
       \immediate
       \write\@auxout{\string\bibcite{#2}{#1}}}\fi\ignorespaces}
```

Let's take an example in order to see how it works: Assume we wrote `\bibitem[GMS94]{companion}`. The command first creates an item having the same optional argument, which will be surrounded with brackets by `\@biblabel` and flushed to the left by `\hfill`. It then write a `\bibcite` command, with two arguments, into the `.aux` file[7]. `\bibcite` is simply defined as follows:

```
\def\bibcite{\@newl@bel b}
```

The `\@newl@bel` command requires three arguments, `#1`, `#2` and `#3`, and defines a command named `#1@#2` (with of course `#1` and `#2` being replaced by their values) whose value is the third argument `#3`.

This behavior is the same as when defining a cross reference (with `\label`) in the document: when the `.aux` file is read by LaTeX (namely at the `\begin{document}` and `\end{document}`), those `\@newl@bel` commands are executed, and a `\b@companion` command is defined, containing, in our case, `GMS93`.

When there is no optional argument, it is quite similar:

```
\def\@bibitem#1{\item\if@filesw \immediate\write\@auxout
        {\string\bibcite{#1}{\the\value{\@listctr}}}\fi\ignorespaces}
```

The new `\item` is created, and the `\bibcite` command is output in the `.aux` file. The only new thing to know here is that `\@listctr` is the list counter, and points to `enumiv` as requested by the `\usecounter` command in the definition of `thebibliography`. Everything after the `\bibitem` (and its argument(s)) is output in the document, within the recently created item of the list, until the next `\bibitem` or the end of the `thebibliography` environment[8].

To conclude, here is a small example of a bibliography, having two entries, as it could be defined in a document:

```
\begin{thebibliography}{GMS93}   %% GMS93 is the longest label.
\bibitem[GMS93]{companion} Michel Goossens, Franck Mittelbach and Alexander
     Samarin, \emph{The \LaTeX{} Companion}, Addison Wesley, 1993.
\bibitem[Lam97]{lamport} Leslie Lamport, \emph{\LaTeX: A Document Preparation
     System}, Addison Wesley, 1997.
\end{thebibliography}
```

And here is the result:

References

[GMS93] Michel Goossens, Franck Mittelbach and Alexander Samarin, *The LaTeX Companion*, Addison Wesley, 1993.

[Lam97] Leslie Lamport, *LaTeX: A Document Preparation System*, Addison Wesley, 1997.

3 The `\cite` command

When considering a bibliography as a list of cross-references, `\cite` is the equivalent for `\ref`. It has one mandatory argument, which is the internal label to be cited. It also has an optional argument, that can be used to add some comments to the reference. For instance, a good reference concerning BibTeX is [GMS93, Chap. 13], which is obtained by entering `\cite[Chap.~13]{companion}`.

Here is how it is defined[9]:

[7] More precisely, the file pointed to by the `\@auxout` command, but this generally is the `.aux` file.

[8] Some packages redefine `\bibitem` and could not meet this last rule. See section 4.4 on this topic.

[9] Details about `\DeclareRobustCommand` are given at page 7. If you don't know what it is, you can see it as a simple `\newcommand`.

```
\DeclareRobustCommand\cite{%
  \@ifnextchar [{\@tempswatrue\@citex}{\@tempswafalse\@citex[]}}
```

If there is an optional argument, the boolean variable `\@tempswa` is set to true (we need to remember that an optional argument was provided), and `\@citex` is called. Otherwise, `\@tempswa` is false, and `\@citex` is called with an empty optional argument.

Before explaining `\@citex`, we first have a quick look at `\@cite`, which will be used by `\@citex`. This will help understand how `\@tempswa` is used:

```
\def\@cite#1#2{[{#1\if@tempswa , #2\fi}]}
```

This is the command used for outputting the reference in the document. The second argument is used only if `\@tempswa` has been set to true. Together with the first argument, they are put into brackets, and output in the document.

Now `\@citex` will be the "bridge" between `\cite` and `\@cite`:

```
\def\@citex[#1]#2{%
  \let\@citea\@empty
  \@cite{\@for\@citeb:=#2\do
```

This calls `\@cite`. Its first argument will be computed by the `\@for` command, in case several references are cited at one time.

```
    {\@citea\def\@citea{,\penalty\@m\ }%
```

Starting from the second run through the `\@for`-loop, we add a comma, and a penalty for a line break not to occur between references. The default is to never have a line break inside a set of references.

```
     \edef\@citeb{\expandafter\@firstofone\@citeb\@empty}%
```

This redefines `\@citeb`, the variable used in the loop. The `\@for` command successively sets `\@citeb` to all the values that have been cited, and `\@citeb` is redefined here in order to remove extra spaces. This is somewhat tricky, but it works.

```
     \if@filesw\immediate\write\@auxout{\string\citation{\@citeb}}\fi
```

This writes a `\citation` command in the `.aux` file, indicating that `\@citeb` has been cited in the document. This is not useful here, but will be crucial for BibTEX to generate the bibliography (see section 5).

```
     \@ifundefined{b@\@citeb}{\mbox{\reset@font\bfseries ?}%
       \G@refundefinedtrue
       \@latex@warning
         {Citation `\@citeb' on page \thepage \space undefined}}%
```

This handles cases where the requested reference does not exist yet. In that case, the reference is replaced by a boldface question mark. A warning is also echoed in the `.log` file.

```
       {\@cite@ofmt{\csname b@\@citeb\endcsname}}}}{#1}}
```

If the reference exists, it is written here, using the `\b@...` command created when reading the `.aux` file (cf. page 5). `\@cite@ofmt` is equivalent to `\hbox` [10]. The loop is executed for all the requested references, and the whole result is then passed to `\@cite`, together with the optional second argument, which is `#1` here.

This may look intricate, but it is really easy to use: You simply enter `\cite{companion, lamport}` in order to get [GMS93, Lam97], with the bibliographic items shown at the end of the previous section.

[10] The command `\@citex` is defined with `\hbox` in old (before 2003) versions of LaTeX.

4 Some more little tricks

4.1 What is `\DeclareRobustCommand`?

A command having an optional argument, such as `\cite`, is said to be *fragile*: Generally speaking, they cannot be directly used in the argument of other commands (for instance, `\cite` in the argument of a `\section`). These problems can be overcome by preceding the fragile command by a `\protect`, but this is annoying. The other solution is to declare the command as robust, by defining it with `\DeclareRobustCommand` instead of `\newcommand`.

4.2 Changing the name of the bibliography

This is obvious from the definition of the `thebibliography` environment: We simply have to redefine `\refname`, which defaults to *References*. However, this only works with the report.cls class. The book.cls and article.cls classes use `\bibname` instead, which defaults to *Bibliography*.

For instance, when using report.cls, you'll write:

```
\renewcommand{\refname}{Some references}
```

while with book.cls or article.cls, it should be:

```
\renewcommand{\bibname}{Some references}
```

As mentioned earlier, apalike.sty does not use `\refname` and hard-codes the reference name in the page headers.

4.3 Adding text before the first reference

Putting text just after the beginning of the `thebibliography` environment raises an error, since the `list` environment demands an `\item` command. Thus we will put a real `\item`, then add some negative horizontal space back to the left margin, and write our text within a `minipage` environment (in order to avoid indentation due to the list):

```
\begin{thebibliography}{GMS93}
\item[]
\hskip-\leftmargin
\begin{minipage}{\textwidth}
Here are some useful references about \LaTeX. They are
available in every worthy bookshop. Many other good documentations
might be found on the web (the FAQ of \textsf{comp.text.tex} for
instance).
\end{minipage}
\bigskip
\bibitem[GMS93]{companion} Michel Goossens, Franck Mittelbach and
     Alexander
     Samarin, \emph{The \LaTeX{} Companion}, Addison Wesley, 1993.
\bibitem[Lam97]{lamport} Leslie Lamport, \emph{\LaTeX: A Document Preparation
     System}, Addison Wesley, 1997.
\end{thebibliography}
```

This code gives:

References

Here are some useful references about LaTeX. They are available in every worthy bookshop. Many other good documentations might be found on the web (the FAQ of comp.text.tex for instance).

[GMS93] Michel Goossens, Franck Mittelbach and Alexander Samarin, *The LaTeX Companion*, Addison Wesley, 1993.

[Lam97] Leslie Lamport, *LaTeX: A Document Preparation System*, Addison Wesley, 1997.

4.4 Redefining `\bibitem`

Some style files need redefining the `\bibitem` command, or in fact `\@bibitem` and `\@lbibitem`, so that an entry has to end with a `\par` command (or an empty line). backref.sty is an example of such a style file. Some other will turn the optional argument of `\bibitem` into a mandatory one. apalike.sty does so.

I've nothing to add about that, but it is useful to know this to avoid spending too much time on debugging...

4.5 Turning brackets into parentheses

As we saw earlier, `\biblabel` is in charge of adding brackets around reference labels, in the reference list. It is easy to redefine it in order to get parentheses:

```
\makeatletter % @ is now a letter
\def\bibleftdelim{(}
\def\bibrightdelim{)}
\def\@biblabel#1{\bibleftdelim #1\bibrightdelim}
\makeatother % @ is a symbol
```

This does the trick, and it is now easy to change parentheses into anything else, by redefining `\bibleftdelim` and `\bibrightdelim`.

However, this won't change the behavior of `\@cite`, which will still write brackets around cited reference labels. Thus we also have to redefine `\@cite`:

```
\makeatletter
\def\@cite#1#2{\bibleftdelim{#1\if@tempswatrue , #2\fi}\bibrightdelim}
\makeatother
```

4.6 ... and more

Several packages have been written for modifying the look of bibliographies and citations. For instance, the package cite.sty allows to modify the result of the `\cite` command: turning brackets into parentheses, but also sorting and compressing a set of numeric references. The package overcite.sty allows, moreover, to get references displayed as superscript.

The package splitbib.sty changes the output of the list of references: it allows to split the bibliography into several categories, and/or reordering that list. See the documentation [Mar05] for more details.

4.7 Can the symbol $ be used in an internal key?

Probably not, but I don't know exactly which character may or may not be used. Obviously, any letter and digit can be used, and my opinion is that it is quite enough. On the other hand, commas, curly brackets and backslashes are clearly forbidden. For the other ones, I don't know, just give it a try and you'll know.

Conclusion

Well... It could be over right now, since we know how to write a bibliography. However, typesetting all references by hand is long and annoying. Moreover, when writing several articles on related areas, we often cite the same sets of references, but possibly with different styles. It would be interesting to have a *database* containing a large set of references, some of which would be picked up, formatted and typeset by LaTeX. This does exist, and is described in the sequel.

Part 2
How to use BibTeX?

Table of Contents

5 How does it work?

As mentioned earlier, BibTeX can be seen as a general database manager: It extracts items from a database, sorts them, and exports the result, generally as a LaTeXable `thebibliography` environment.

This description is a somewhat optimistic view, however, and reality is a little more complex: In order to tell BibTeX which entries have to be extracted, you have to first run LaTeX on your document. And once BibTeX has finished its job, you have to run LaTeX anew in order to take the resulting bibliography into account. Here are some more precisions:

- At the very first stage, you run LaTeX on your document. Since this is the first time you compile it, no bibliographic information is included, and references are left empty. Nevertheless, each time LaTeX encounters a bibliographic reference in the document, it writes the key of the reference in the `.aux` file[11]. During the compilation, LaTeX also indicates in the `.aux` file which databases have to be used, and which bibliography style has to be applied for typesetting the bibliography;

- BibTeX can now be executed: It takes the `.aux` file as argument, which contains all the relevant informations for extracting the bibliography: The style (`.bst` file) to be applied to the bibliography, the database (`.bib` file) to be used, and the entries to be extracted from the database. With this stuff, BibTeX will then extract the `\cited` references and write the bibliography in a `.bbl` file. Logs of this operation are written in a `.blg` file;

- The next step is to rerun LaTeX. The `.bbl` file will then be included, and its `\bibitem` commands executed. This writes the necessary bibliographic cross references in the `.aux` file. However, this does not define the cross references for the current compilation, and bibliographic references still won't be defined;

[11] This is the role of the `\citation` command issued by the `\cite` command (cf. page 6).

- We then run LaTeX a third time: When reading the `.aux` file at the beginning of the compilation, LaTeX will store the references of the bibliographic citations, and those references will be correct in the document.

It follows that, in the best case, we need to compile the file three times, and to run BibTeX once. There are some cases where this is still not sufficient: If, for instance, a bibliographic entry cites another one, another run of both BibTeX and LaTeX is needed. And so on. Finally, here is the global pattern to be applied:

$$\text{LaTeX (BibTeX LaTeX)}^{+}\ \text{LaTeX}.$$

Everything else works as in part 1, since BibTeX generally creates the complete `thebibliography` environments with all the `\bibitem`s of the references that are cited in your document. There are two new LaTeX commands, however: Those defining the style file to consider and bibliographic database(s):

- `\bibliographystyle` is the command to be used to declare the bibliography style to be used by BibTeX. Here is how it is defined:

```
\def\bibliographystyle#1{%
  \ifx\@begindocumenthook\@undefined\else
    \expandafter\AtBeginDocument
  \fi
    {\if@filesw
        \immediate\write\@auxout{\string\bibstyle{#1}}%
     \fi}}
```

  This simply writes a `\bibstyle` command in the `.aux` file, whose argument is the name of the style. `\bibstyle` itself is a command that has one argument, but does nothing. Indeed, the name of the bibliography style is only needed by BibTeX, and LaTeX doesn't care about it.

- `\bibliography` is the command for defining the bibliographic database to be used. Contrary to the previous command, the argument of `\bibliography` might be a comma-separated list of bibliographic databases. Note that "comma-separated" is strict here: No space and no line break are allowed. Apart from this, the behavior of this command is similar to the behavior of the previous one: It writes its argument in the `.aux` file, as the argument of a command named `\bibdata` (which is then read by BibTeX, but does nothing in LaTeX). Last, this `\bibliography` command includes the `.bbl` file, which writes the bibliography. Here is the precise definition:

```
\def\bibliography#1{%
  \if@filesw
    \immediate\write\@auxout{\string\bibdata{#1}}%
  \fi
  \@input@{\jobname.bbl}}
```

  You've probably already understood that `\jobname` returns the name of the file being compiled.

Last precision: The `.aux` file also contains the list of keys of the entries to be extracted. This is achieved by the `\citation` commands echoed by `\cite` in the `.aux` file. `\citation` does the same thing as `\bibstyle` and `\bibdata`: Nothing.

6 Some bibliography styles...

There exist quite a bunch of bibliography styles, since each publisher has his own needs and preferences. I'll present some generic styles here, together with their specificities.

6.1 Classical bibliography styles

The following four styles were originally written by Oren PATASHNIK, whom I forgot to introduce but is in fact the author of BibTEX. The styles are named plain.bst, alpha.bst, unsrt.bst and abbrv.bst. If you want to use the plain.bst style, you'll write `\bibliographystyle{plain}` (anywhere) in your document.

Globally speaking, the bibliography style has to manage everything. "Everything" here can be decomposed into three points: Defining the available entry types, with their relevant fields[12]; Extracting and sorting bibliographic items; And typesetting the bibliography.

So the first role of the style file is to define entry types, such as **@book**, **@article** or **@inproceedings** in classical styles, and for each of them, the relevant fields that will be either mandatory or optional or just ignored[13]. The table below describes the role of the fields that are used in classical style files. Descriptions are short, but more details can be easily found in any LATEX book.

address	Generally the city or complete address of the publisher.
author	For author names. The input format is quite special, since BibTEX has to be able to distinguish between the first and last names. Section 11 and 18 are completely dedicated to this topic.
booktitle	For the title of a book one part of which is cited.
chapter	The number of the chapter (or any part) of a book being cited. If not a chapter, the `type` field might be used for precising the type of sectioning.
crossref	This one is quite peculiar. It's used to cross-reference within the bibliography. For instance, you might cite a document, and a part of it. In that case, the second one can reference the first one, or at least inherit some of its fields from the first one. This deserves some more comments, see section 12.
edition	The edition number. Or in fact its ordinal, for instance `edition = "First"`. This might raise problems when trying to export a bibliography into another language.
editor	The name of the editor(s) of the entry. The format is the same as for authors.
howpublished	Only used in rare cases where the document being cited is not a classical type such as a **@book**, an **@article** or an **@inproceedings** publication.
institution	For a technical report, the name of the institution that published it.
journal	The name of the journal in which the cited article has been published.
key	Used for defining the label, in case it cannot be computed by BibTEX. It does not force the label, but defines the label when BibTEX needs one but can't compute it.
month	Well... The month during which the document has been published. This also raises the problem of the translation of the bibliography: It's better having a numerical value, or an abbreviation, instead of the complete name of the month. Having the number would also allow BibTEX to sort the entries more precisely (even though, as far as I know, no bibliography style does this at the present time).
note	For any additional data you would want to add. Since classical styles were written in 1985, they don't have a `url` field, and `note` is often used for this purpose, together with the url.sty package.

[12] I insist that entry types and field names depend on the bibliography style, and are not fixed by BibTEX.

[13] The following rule applies: A field that is neither mandatory nor optional, is ignored. Thus you can add any comment or personal field in your bibliography, even if they're not in the list below. Some other fields might of course be used by other, non classical styles.

number	A number... Not whichever, but the number of a report. For volume numbers, a special `volume` field exists.
organization	The organizing institution of a conference.
pages	The relevant pages of the document. Useful for the reader when you cite a huge book; Note that such a precision could be added through the optional argument of `\cite` (see page 5), in which case it would appear in the document but not in the bibliography.
publisher	The institution that published the document.
school	For theses, the name of the school the thesis has been prepared in.
series	The name of a collection of series or books.
title	The title of the document being cited. There are some rules to be observed when entering this field, see section 10.
type	The type. Which type? It depends... The type of publication, if needed. For thesis, for instance, in order to distinguish between a masters thesis and a PhD. Or the type of section being cited (see `chapter` above).
volume	The volume number in a series or collection of books.
year	The publication year.

The table below describes the different entry types. Once again, you can find more details in any LaTeX documentation, so I won't go deep into the details.

Entry type	Mandatory fields	Optional fields
@article : An article published in a journal.	`author`, `title`, `year`, `journal`.	`volume`, `number`, `pages`, `month`, `note`.
@book : Well... A book.	`author` or `editor`, `title`, `publisher`, `year`.	`volume` or `number`, `series`, `address`, `edition`, `month`, `note`.
@booklet : A *small* book, that has no `publisher` field.	`title`.	`author`, `howpublished`, `address`, `address`, `month`, `year`, `note`.
@conference : Article that appeared in the proceedings of a conference, a meeting...	`author`, `title`, `booktitle`, `year`.	`editor`, `volume` or `number`, `series`, `pages`, `address`, `month`, `organization`, `publisher`, `note`.
@inbook : Part (generally a chapter) of a book.	`author` or `editor`, `title`, `chapter` or `pages`.	`volume`, `number`, `series`, `type`, `address`, `edition`, `month`, `note`.
@incollection : Part of a book having its own title.	`author`, `title`, `booktitle`, `publisher`, `year`.	`editor`, `volume` or `number`, `series`, `type`, `chapter`, `pages`, `address`, `edition`, `month`, `note`.

Entry type	Mandatory fields	Optional fields
@inproceedings	Same as **@conference**.	
@manual : A little manual, such as this one for instance.	`title`.	`author`, `organization`, `year`, `address`, `edition`, `month`, `note`.
@mastersthesis : Masters thesis, or something equivalent.	`author`, `title`, `school`, `year`.	`type`, `address`, `month`, `note`.
@misc : When nothing else fits...	at least one of the optional fields.	`author`, `title`, `howpublished`, `year`, `month`, `note`.
@phdthesis : PhD dissertation, or similar.	`author`, `title`, `school`, `year`.	`type`, `address`, `month`, `note`.
@proceedings : Conference proceedings.	`title`, `year`.	`editor`, `volume` or `number`, `series`, `address`, `month`, `organization`, `publisher`, `note`.
@techreport : Technical report, published by a laboratory, a research center, ...	`author`, `title`, `institution`, `year`.	`type`, `address`, `number`, `month`, `note`.
@unpublished : A document that has not been published. Very close to **@misc**, but `author` and `title` are needed here.	`author`, `title`, `note`.	`month`, `year`.

Concerning the order and typesetting of all this stuff, it's similar in all the classical style files. Of course it depends on the entry type, but it generally begins with the author names and the title. Then the references of the journal or of the proceedings... The best thing to do if you need more details is to give it a try, or have a look directly in the style files (but read Part 4 before if you don't know how BibTEX style files are built).

Up to now, there are no differences between the four standard style files. The only possible difference now can only be related to labelling and typesetting method to be used.

- the plain.bst style sorts the entries according to the name of their authors (using the alphabetical order[14], of course), and, for papers by the same author(s), the year they have been published (the older first). The last criterion in case of equality is the title, being a little bit modified. If two references can't be distinguished with the above, the first one being cited in the document appears first. Labels are numbers, starting with 1.
- the alpha.bst style file is named *alpha* because it uses alphanumerical labels: Those labels are computed by BibTEX using the first three letters of the author name (or initials of the author names if multiple authors), followed by the last two digits of the publication year. Sorting the entries is done according to the label first, and then to the same criteria as for plain.bst, in case several publication have the same label[15];

[14]The standard alphabetical order with 26 letters. Unfortunately, some language have a different alphabet, for instance Swedish, in which "å" and "ö" are considered as letters and placed after "z".

[15]You certainly don't want that several publications have the same labels. Thus computing the labels is done in two phases: First computing the label with the standard method, and then adding a supplementary letter ("a", "b", ...) for multiple labels. And sorting is done just between those two phases...

- you probably understood what unsrt.bst does: It does not sort its references, which appears in the order they are cited in the document. Everything else is done as in plain.bst;
- abbrv.bst abbreviates first names of the authors and the names of predefined journal and month names. I forgot to mention that bibliography styles historically predefine some shorthands for computer science journal names (Oren Patashnik is a computer scientist...). Those shorthands are abbreviated journal names in this style file. These are the only difference between abbrv.bst and plain.bst.

That's all for classical styles. Those styles suffer from several problems, for instance not having a `url` field, or not being multilingual, or sorting in a weird way... Moreover, publishers often impose precise typographic rules for bibliographies. This entails that many other styles have been proposed. Let's have a look at some of them.

6.2 Some other bibliography styles

6.2.1 The apalike.bst style

apalike.bst was (also) written by Oren Patashnik. It uses a special construction for labels, generally called *author-year*. I think the best way to understand is with some examples:

On page 384 of (Goossens et al., 1993), you'll find a complete example of what apalike.bst does.

References

Goossens, M., Mittelbach, F., and Samarin, A. (1993). *the LaTeX Companion*. Addison-Wesley.

Lamport, L. (1997). *LaTeX: A Document Preparation System*. Addison-Wesley.

Don't forget to include the apalike.sty package when using the apalike.bst style: Indeed, if you remember how `\@biblabel` and `\cite` are defined, you should have seen why... Moreover, labels created by apalike.bst might be long, and you probably will accept that LaTeX hyphenates them if necessary, which the default `\cite` won't do (cf. section 3).

Also: There are some other *author-year* style files, named authordate1.bst, authordate2.bst, authordate3.bst, authordate4.bst, and that slightly differ from apalike.bst. They must be used together with the authordate1-4.sty package.

Very last important thing: apalike.sty redefined `\bibitem` so that the optional argument becomes mandatory (but must still be within square brackets). But you probably don't care since, of course, the bibliography style file tells BibTeX to always output it.

6.2.2 The natbib.sty package

The natbib.sty package, written by Patrick W. Daly, goes a bit further: It mainly redefines `\cite` so that you can get *author-year* or numerical labels, in a very elegant way.

Classical bibliography styles have been ported to natbib.sty, except alpha.bst since it already was an *author-year* style. The names of the ports are plainnat.bst, abbrvnat.bst et unsrtnat.bst. Moreover, those styles have a `url` field for adding reference to articles on the web. Also note that natbib.sty can be used with apalike.bst or authordate1.bst to authordate4.bst.

Last interesting remark: There is a very clean documentation [Dal99c] for natbib.sty, it's really worth reading.

6.2.3 The jurabib.sty package

The package jurabib.sty, by Jens Berger, is another package for adapting the output to the typographic rules used in legal studies. It is associated to a bibliography style jurabib.bst, and rests on a very special format for the optional argument of `\bibitem`. It also redefines `\cite`, and defines many flavors of that command. See [Ber02] for detailed comments on jurabib.sty.

6.2.4 custom-bib

Since there are many possible criteria for defining a bibliographic style, Patrick W. Daly decided to write a little piece of software for automatically generating customized bibliography styles. It asks you about 20 questions and produces a pretty ready-to-use bibliography style file.

As usual with Patrick Daly, the documentation [Dal99b] is excellent, and I won't go any further in the details. I just mention how to create your `.bst`-file: Simply type `latex makebst.tex`, and answer the questions. All along the execution, the style file is being created. It's really easy and intuitive.

7 Questions and answers

This section ends the LaTeX part of this manual with some frequently asked questions. We'll then tackle the BibTeX part, which was intended to be the central topic...

7.1 How to get several bibliographies?

I did not insist on that, but it is of course recommended that you only have one occurrence of each `\bibliography` and `\bibliographystyle` commands. LaTeX won't mind if it is not the case, but BibTeX won't appreciate, since it can't choose which style or bibliographic file to use.

Thus the only possible thing is to write several `.aux` files, each of them containing a complete set of data for one bibliography. This is precisely what multibib.sty, chapterbib.sty and bibunit.sty do.

- multibib.sty, written by Thorsten Hansen, allows for precising in which bibliography a document lies. This is achieved by using different `\cite` commands, one for each bibliography. For this manual, I could have used multibib.sty to define special `\citelatex` and `\citebibtex` commands defining two separate bibliographies[16]. Of course, this requires that you run BibTeX against each of the `.aux` files created. For more details, refer to the documentation [Han00b], which (also) is really worth reading. The doc also mentions the main issue with such a method: The same label could be used in several bibliographies for different references. And you can't easily avoid this problem...

- The chapterbib.sty package, mainly written by Donald Arseneau, provides a way to get one bibliography for each chapter or part of a (long) document (short documents won't need several bibliographies). Indeed, a long document will be made of several files `\include`d by a main file. chapterbib.sty creates one `.aux` file for each of those included files, each of which being intended to contain the necessary `\bibstyle`, `\bibdata` and `\citation` commands. Documentation for this package may be found at the end of the package file itself.

- The bibunit.sty package, also written by Thorsten Hansen, is pretty similar to chapterbib.sty: it allows to create one bibliography for each "unit", a unit being any part of the document beginning with `\begin{bibunit}` and ending with `\end{bibunit}`.

  Inside such a unit, all occurrences of `\cite` refer to the bibliography of the current unit. [Han00a] gives many details of how it works.

There are some other packages for creating several bibliographies: camel.sty, bibtopic.sty... I can't detail everything since it is not the aim of this manual.

[16] For those who know the multind.sty package for getting multiple indexes, the idea is similar.

7.2 How to have references closer to where they are cited?

There are two solutions to this problem, depending on what you precisely want: The first solution consists in putting references in footnotes. This might be practical for the reader, since he has the complete references without always going back and forth to the end of the book. There is a package just designed to this purpose: footbib.sty. It's quite well documented [Dom97].

The second solution is to write the complete references in the text. Instead of citing [GMS93], you cite Michel Goossens, Franck Mittelbach et Alexander Samarin, *The LaTeX Companion*, Addison Wesley, 1993. This may enhance your document, but can sometimes become annoying for the reader. However, just have a look a the package bibentry.sty, and its documentation [Dal99a], if you want some more details.

Note that both of these packages might conflict with some other ones, such as hyperref.sty. This is due to the fact that each of them wants to impose its own definition of the `thebibliography` environment or of the `\bibitem` command. The last one being included will win, but the other ones will eventually complain. There is no simple solution to avoid this, the only way, if any, is to merge all the definitions by hand.

7.3 How to add entries in the reference list without citing them in the document?

This is achieved by the `\nocite` command. It works exactly like `\cite`, but writes nothing in the document. It just includes the `\citation` command in the `.aux` file.

A variant of this command is `\nocite{*}`: it amounts to `\nocite`-ing the whole bibliography at one time. Those references are included in the same order as they appear in the `.bib` file, except for those having been cited earlier. Note that `\cite{*}` is also correct, but I'm not sure it has any interest...

Well... That's all for this part. We now forget LaTeX, and go closer to the core of the problem. The next section explains how to create a clean `.bib` file.

Part 3
The .bib file

Table of Contents

The .bib file is the database. Its contents heavily depends on the style being applied to it, even though bibliography styles are generally "compatible" with each w.r.t. the database. I will only describe here the case of standard styles[17]. But remember that BibTEX can do many other things, we'll see some examples in section 5.

8 Structure of the .bib file

We start with an example:

```
@book{companion,
  author       = "Goossens, Michel and Mittelbach, Franck and Samarin, Alexander",
  title        = "The {{\LaTeX}} {C}ompanion",
  publisher    = "Addison-Wesley",
  year         = 1993,
}
```

We just consider the structure of this entry for the moment, not its contents. The general form is the following[18]:

[17] Thus using entry types and fields as described at pages 12 to 13.
[18] The outermost braces can be replaced by parentheses: `@entry_type(internal_key, ...)`.

```
@entry_type{internal_key,
  field_1       = "value_1",
  field_2       = "value_2",
  ...
  field_n       = "value_n"
}
```

Some basic remarks:

- New entries always start with `@`. Anything outside the "argument" of a "command" starting with an `@` is considered as a comment. This gives an easy way to comment a given entry: just remove the initial `@`. As usual when a language allows comments: Don't hesitate to use them, so that you have a clean, ordered, and easy-to-maintain database. Conversely, anything starting with an `@` is considered as being a new entry[19].
- BibTeX does not distinguish between normal and capital letters in entry and field names. BibTeX will complain if two entries have the same internal key, even if they aren't capitalized in the same way. For instance, you cannot have two entries named `Example` and `example`.

  In the same way, if you cite both `example` and `Example`, BibTeX will complain. Indeed, it would have to include the same entry twice, which probably is not what you want;
- Spaces and line breaks are not important, except for readability. On the contrary, commas are compulsory between any two fields;
- Values (*i.e.* right hand sides of each assignment) can be either between curly braces or between double quotes. The main difference is that you can write double quotes in the first case, and not in the second case. For citing *Comments on "Filenames and Fonts"* by Franck Mittelbach, you can use one of the following solutions:

  ```
  title         = "Comments on {"}Filenames and Fonts{"}",
  title         = {Comments on "Filenames and Fonts"},
  ```

  Curly braces have to match, since they will appear in the output to be compiled by LaTeX. A problem occurs if you need to write a (left-, say) brace in an entry. You could of course write `\{`, but the entry will have to also include the corresponding right brace. To include a left brace without its corresponding right brace, you'll have use a LaTeX function having no brace in its name. `\leftbrace` is the right choice here. Another solution is to add an extra `\bgroup` in the entry, so that both LaTeX and BibTeX will find the correct number of "braces".
- For numerical values, curly braces and double quotes can be omitted.

As I already mentioned, you can define fields even if they aren't used by the style being applied. For instance, the following can be used for the LaTeX Companion:

```
@book{companion,
  author        = "Goossens, Michel and Mittelbach, Franck and Samarin, Alexander",
  title         = "The {{\LaTeX}} {C}ompanion",
  booktitle     = "The {{\LaTeX}} {C}ompanion",
  publisher     = "Addison-Wesley",
  year          = 1993,
  month         = "December",
  ISBN          = "0-201-54199-8",
  library       = "Yes",
}
```

[19] There is a special entry type named **@comment**. The main use of such an entry type is to comment a large part of the bibliography easily, since anything outside an entry is already a comment, and commenting out one entry may be achieved by just removing its initial `@`.

It gives complementary information about the book[20], for instance the fact that it is available in your local library. You really should not hesitate to use auxiliary, personal fields, giving them explicit names in order to be sure that no bibliography style will use them incidentally[21].

9 The @string and @preamble entries

These are not really entry types: **@string** entries can be used in order to define abbreviations. For instance, we've cited two books published by Addison-Wesley. It might be useful to define a shortcut for this publisher. Thus we write:

```
@string{AW      = "Addison-Wesley"}

@book{companion,
  author        = "Goossens, Michel and Mittelbach, Franck and Samarin, Alexander",
  title         = "The {{\LaTeX}} {C}ompanion",
  booktitle     = "The {{\LaTeX}} {C}ompanion",
  publisher     = AW,
  year          = 1993,
  month         = "December",
  ISBN          = "0-201-54199-8",
  library       = "Yes",
}
```

This does not only spare some time, but most importantly, it ensures that you won't misspell it, and helps you maintain an homogeneous database. I find this really interesting for author names, so that you're sure to always write them correctly (or always incorrectly, but it would be easy to detect and correct a global mistake anyway).

As regards **@preamble**, it may be used for inserting commands or text in the file created by BibTEX. Anything declared in a **@preamble** command will be concatenated and put in a variable named `preamble$`, for being used in the bibliography style and, generally, inserted at the beginning of the `.bbl` file, just before the `thebibliography` environment. This is useful for defining new commands used in the bibliography. Here is a small example:

```
@preamble{ "\makeatletter" }
@preamble{ "\@ifundefined{url}{\def\url#1{\texttt{#1}}}{}" }
@preamble{ "\makeatother" }
```

This way, you may safely use the `\url` command in your entries. If it is not defined at the beginning of the bibliography, the default command defined in the **@preamble** will be used.

Please note that you should never define style settings in the **@preamble** of a bibliography database, since it would be applied to any bibliography built from this database.

10 The `title` field

Let's see how to fill the "title" field. We start by studying how I entered the title field for the *LaTeX Companion* :

```
  title         = "The {{\LaTeX}} {C}ompanion"
```

[20] Filling both `title` and `booktitle` for a book may be of real interest, as we will see in the section dedicated to cross references (see section 12 for some more details).

[21] You'll ask "Is it really useful?". First of all, it is not that long to add those informations each time you add a new entry, and it is much longer to add a field to several entries. Moreover, we'll see later how to design bibliography styles, and you'll be able to write styles that take those new fields into account.

We'll need several definitions before going further. The *brace depth* of an item is the number of braces surrounding it. This is not a very formal definition, but for instance, in the title above, `\LaTeX` has brace depth 2, the `C` has brace depth 1, and everything else has depth 0[22]. A *special character* is a part of a field starting with a left brace being at brace depth 0 immediately followed with a backslash, and ending with the corresponding right brace. For instance, in the above example, there is no special character, since `\LaTeX` is at depth 2. It should be noticed that anything in a special character is considered as being at brace depth 0, even if it is placed between another pair of braces.

That's it for the definitions. Generally speaking, several modifications can be applied to the title by the bibliography style:

- first of all, the title might be used for sorting. When sorting, BibTEX computes a string, named `sort.key$`, for each entry. The `sort.key$` string is an (often long) string defining the order in which entries will be sorted. To avoid any ambiguity, `sort.key$` should only contain alphanumeric characters. Classical non-alphanumeric characters[23], except special characters, will be removed by a BibTEX function named `purify$`. For special characters, `purify$` removes spaces and LATEX commands (strings beginning with a backslash), even those placed between brace pairs. Everything else is left unmodified. For instance, `t\^ete`, `t{\^e}te` and `t{\^{e}}te` are transformed into `tete`, while `tête` gives `tête`; `Bib{\TeX}` gives `Bib` and `Bib\TeX` becomes `BibTeX`. There are thirteen LATEX commands that won't follow the above rules: `\OE`, `\ae`, `\AE`, `\aa`, `\AA`, `\o`, `\O`, `\l`, `\L`, `\ss`. Those commands correspond to ı, ȷ, œ, Œ, æ, Æ, å, Å, ø, Ø, ł, Ł, ß, and `purify$` transforms them (if they are in a special character, in `i`, `j`, `oe`, `OE`, `ae`, `AE`, `aa`, `AA`, `o`, `O`, `l`, `L`, `ss`, respectively.

- the second transformation applied to a title is to be turned to lower case (except the first character). The function named `change.case$` does this job. But it only applies to letters that are a brace depth 0, except within a special character. In a special character, brace depth is always 0, and letters are switched to lower case, except LATEX commands, that are left unmodified.

Both transformations might be applied to the `title` field by standard styles, and you must ensure that your title will be treated correctly in all cases.

Let's try to apply it now, for instance to *the LATEX Companion*. Several solutions might be tried:

- `title = "The \LaTeX Companion"`: This won't work, since turning it to lower case will produce `The \latex companion`, and LATEX won't accept this...

- `title = "The {\LaTeX} {C}ompanion"` : This ensures that switching to lower case will be correct. However, applying `purify$` gives `The  Companion`. Thus sorting could be wrong;

- `title = "The {\csname LaTeX\endcsname} {C}ompanion"`: This won't work since `LaTeX` will be turned to `latex`;

- `title = "The { \LaTeX} {C}ompanion"` : In this case, `{ \LaTeX}` is *not* a special character, but a set of letters at depth 1. It won't be modified by `change.case$`. However, `purify$` will leave both spaces, and produce `The  LaTeX Companion`, which could result in wrong sorting;

- `title = "The{ \LaTeX} {C}ompanion"`: This solution also works, but is not as elegant as the next one;

- `title = "The {{\LaTeX}} {C}ompanion"` : This is the solution I used. It solves the problems mentioned above.

For encoding an accent in a title, say *É* (in upper case) as in the French word *École*, we'll write `{\'{E}}cole`, `{\'E}cole` or `{{\'E}}cole`, depending on whether we want it to be turned to lower case (the first two solutions) or not (the last one). `purify$` will give the same result in the three

[22] Of course, surrounding the field with braces instead of quotes does not modify the brace depth.

[23] Except hyphens and tildes that are replaced by spaces. Spaces are preserved. The precise behavior of `purify$` is explain on page 32.

cases. However, it should be noticed that the third one is not a special character. If you ask BibTEX to extract the first character of each string using `text.prefix$`, you'll get `{\'{E}}` in the first case, `{\'E}` in the second case and `{{\}}` in the third case.

That's all for subtleties of titles. Let's have a look at author names, which is even more tricky.

11 The `author` field

We still begin with the entry for the *LATEX Companion*:

```
author  = "Goossens, Michel and Mittelbach, Franck and Samarin, Alexander"
```

The first point to notice is that two authors are separated with the keyword `and`. The format of the names is the second important point: The last name first, then the first name, with a separating comma. In fact, BibTEX understands other formats.

Before going further, we remark an important point: BibTEX will have to *guess* which part is the first name and which part is the last name. It also has to distinguish a possible "von" part (as in John von Neumann) and a possible "Jr" part.

The following explanation is somewhat technical. The first name will be called `First`, the last name is denoted by `Last`, the "von" with `von` and the "Jr" part, `Jr`.

So, BibTEX must be able to distinguish between the different parts of the `author` field. To that aim, BibTEX recognizes three possible formats:

- `First von Last`;
- `von Last, First`;
- `von Last, Jr, First`.

The format to be considered is obtained by counting the number of commas in the name. Here are the characteristics of these formats:

- `First von Last`: Suppose you entered `Jean de La Fontaine`. There is no comma, hence the format is `First von Last`. The `Last` name cannot be empty, unless the whole field is. It should then contain at least `Fontaine`. BibTEX then looks at the first character[24] of each remaining word. If some of them are lower cases alphabetic letters, anything between the first and the last ones (beginning with lower cases) is considered as being in the `von`. Anything before the `von` is in the `First`, anything after is in the `Last`. If no first letter is in lower case, then everything (except the part already put in the `Last`) is put in the `First`.

  `Jean de La Fontaine` will then give `La Fontaine` as the `Last`, `Jean` for the `First` and `de` for the `von`.

  Here is what it gives for several other combinations. This is for you to check if you understood:

Name	`First`	`von`	`Last`
`jean de la fontaine`		`jean de la`	`fontaine`
`Jean de la fontaine`	`Jean`	`de la`	`fontaine`
`Jean {de} la fontaine`	`Jean de`	`la`	`fontaine`

[24] In the sequel, the first character means "the first non-brace character that at brace depth 0, if any, characters of a special character being at depth 0, even if there are 15 braces around." If there is no character at depth 0, then the item will go with its neighbour, first and foremost with the `First`, then with the `Last`. It will be in the `von` if, and only if, it is surrounded with two `von` items. Moreover, two words in the same group (in LATEX sense) will go to the same place. Last, for a LATEX command outside a special character, the backslash is removed and BibTEX considers the remaining word. If you did not understand, please take a while for reading this note anew, since it will be used in the sequel.

Name	First	von	Last
`jean {de} {la} fontaine`		`jean`	`de la fontaine`
`Jean {de} {la} fontaine`	`Jean de la`		`fontaine`
`Jean De La Fontaine`	`Jean De La`		`Fontaine`
`jean De la Fontaine`		`jean De la`	`Fontaine`
`Jean de La Fontaine`	`Jean`	`de`	`La Fontaine`

The last line is the only one (in this table) that is correct. Of course, some of you will have a counter example where the `von` has to begin with an upper case letter. We'll see this at section 13.

- `von Last, First`: The idea is similar, but identifying the `First` is easier: It's everything after the comma. Before the comma, the last word is put in the `Last` (even if it starts with a lower case). If any other word begins with a lower case, anything from the first word to the last one starting with a lower case is in the `von`, and what remains is in the `Last`. Once again, an example should make everything clear:

Name	First	von	Last
`jean de la fontaine,`[25]		`jean de la`	`fontaine`
`de la fontaine, Jean`	`Jean`	`de la`	`fontaine`
`De La Fontaine, Jean`	`Jean`		`De La Fontaine`
`De la Fontaine, Jean`	`Jean`	`De la`	`fontaine`
`de La Fontaine, Jean`	`Jean`	`de`	`La Fontaine`

- `von Last, Jr, First`: Well... It's still the same, except that anything between the commas is put in the `Jr`.

Names are separated by spaces above, but it may occur that two first names are separated by a hyphen, as in "Jean-François" for instance. BibTEX splits that string, and if both parts are in the `First`, the abbreviated surnames is "J.-F." as (generally) wanted. A tilde is also seen as a string separator. This all boils down to the fact that, if you enter `Jean-baptiste Poquelin`, the string `baptiste` will be in the `von` part of the name, since it erroneously begins with a lower case letter.

I think it's a good place for coming back to abbreviations: You probably agree that names are something not that easy to enter, and are error-prone. I personally advise defining an abbreviation for each author. You'll then concatenate them with ` ``and'' ` using `#`.

For instance, I always include a `.bib` file containing the following lines:

```
@string{goossens        = "Goossens, Michel"}
@string{mittelbach      = "Mittelbach, Franck"}
@string{samarin         = "Samarin, Alexander"}
```

another one containing:

[25] This case raises an error message from BibTEX, complaining that a name ends with a comma. It is a common error to separate names with commas instead of "`and`".

```
@string{AW              = "Addison-Wesley"}
```

and my main bibliographic file contains[26]:

```
@book{companion,
  author        = goossens #" and "# mittelbach #" and "# samarin,
  title         = "The {{\LaTeX}} {C}ompanion",
  booktitle     = "The {{\LaTeX}} {C}ompanion",
  year          = 1993,
  publisher     = AW,
  month         = "December",
  ISBN          = "0-201-54199-8",
  library       = "Yes",
}
```

This makes adding an entry much easier when you're used to such an encoding. Moreover, you can easily recover self-contained entries by using `bibexport.sh` tool (see section 23).

12 Cross-references (`crossref`)

As mentioned earlier (can't remember where... Oh, yes, on page 12), BibTEX allows for cross-referencing. This is very useful, for instance, when citing a part of a book, or an article in conference proceedings. For instance, for citing chapter 13 of the LATEX Companion:

```
@incollection{companion-bib,
  crossref      = "companion",
  title         = "Bibliography Generation",
  chapter       = 13,
  pages         = "371-420",
}
```

This shows why having defined the `booktitle` field of the `companion` entry is useful: It is not used in the **@book** entry, but it is inherited in the above **@incollection** entry. We could of course have added it by hand, but we should have added it in each chapter we cite.

The other possible interesting feature is that, when cross-referencing several times an entry that is not cited by itself, BibTEX can "factor" it, *i.e.* add it in the list of references and explicitly `\cite` it in each entry. On the other hand, if the cross reference appears only once, it inherits from the fields of the reference, which is not included in the bibliography. Here is an example of both behaviours:

References

[1] Michel Goossens, Franck Mittelbach, and Alexander Samarin. Bibliography generation. In *The LATEX Companion* [4], chapter 13, pages 371–420.

[2] Michel Goossens, Franck Mittelbach, and Alexander Samarin. Higher mathematics. In *The LATEX Companion* [4], chapter 8, pages 215–258.

[3] Michel Goossens, Franck Mittelbach, and Alexander Samarin. Index generation. In *The LATEX Companion* [4], chapter 12, pages 345–370.

[4] Michel Goossens, Franck Mittelbach, and Alexander Samarin. *The LATEX Companion*. Addison-Wesley, December 1993.

[26] This is not quite true, since `September` still depends on the language, and I prefer using `9` and letting the style file translate that into the corresponding month in the correct language.

or

References

[1] Michel Goossens, Franck Mittelbach, and Alexander Samarin. Bibliography generation. In *The LaTeX Companion*, chapter 13, pages 371–420. Addison-Wesley, December 1993.

[2] Michel Goossens, Franck Mittelbach, and Alexander Samarin. Higher mathematics. In *The LaTeX Companion*, chapter 8, pages 215–258. Addison-Wesley, December 1993.

[3] Michel Goossens, Franck Mittelbach, and Alexander Samarin. Index generation. In *The LaTeX Companion*, chapter 12, pages 345–370. Addison-Wesley, December 1993.

In order to have one presentation or the other, we can tell BibTeX the number of cross-references needed for an entry to be explicitly `\cite`'d. We use the `-min-crossrefs` command line argument of BibTeX for that. For the first example above, I used `bibtex biblio`, while the second example was obtained with `bibtex -min-crossrefs=5 biblio`.

One other important remark is that cross-referenced entries must be defined *after* entries containing the corresponding `crossref` field. And you can't embed cross-references, that is, you cannot `crossref` an entry that already contains a `crossref`.

Last, the `crossref` field has a particular behaviour: it always exists, whatever the bibliography style. If a `crossref`'ed entry is included (either by a `\cite` command or by BibTeX if there are sufficiently many `crossrefs`), then the entries cross-referencing it have their original `crossref` field. Otherwise, that field is empty.

13 Quick tricks

13.1 How to get *Christopher* abbreviated as *Ch.*?

First names abbreviation is done when extracting names from the `author` field. If the function is asked to abbreviate the first name (or any other part of it), it will return the first character of each "word" in that part. Thus *Christopher* gets abbreviated into *C.* Special characters provide a solution against this problem: If you enter `{\relax Ch}ristopher`, the abbreviated version would be `{\relax Ch}.`, which gives *Ch.*, while the long version is `{\relax Ch}ristopher`, *i.e. Christopher*.

13.2 How to get caps in `von`?

Note: This part is somewhat technical. You should probably read and understand how BibTeX extracts names, which is explained at pages 33 and 34.

It may occur that the `von` part of a name begins with a capital letter. For some reason, the standard example is *Maria De La Cruz*.

The basic solution is to write `"{\uppercase{d}e La} Cruz, Maria"`. When analyzing this name, BibTeX will place `Cruz` in the `Last` part, then `Maria` as the `First` name. Then `{\uppercase{d}e La}` is a special character, whose first letter is `d`, and is thus placed in the `von` part.

In that case, however, if you use an "alphanumeric" style such as `alpha.bst`, BibTeX will use the first character of the `von` part in the label[27]. But the first character here is `{\uppercase{d}e La}`, and the label would be `{\uppercase{d}e La}C`, and you'll get `[De LaC]`. You'd probably prefer `[DLC]` or `[Cru]`. The second solution would then be

```
author = {\uppercase{d}}e {\uppercase{l}}a Cruz, Maria.
```

The label would be `{\uppercase{d}}{\uppercase{l}}C`, which is correct. Another (easier) proposal would be

[27] It could be argued `von` parts should not be used when computing the label, but classical style file do use it.

```
author = {D}e {L}a Cruz, Maria.
```

This also solves the problem, because BibTEX only considers characters at level 0 when determining which part a word belongs to, but takes all letters into account when extracting the first letter.

13.3 How to get lowercase letters in the `Last`?

This is precisely the reverse problem, but the solution will be different. Assume you cite a paper by the famous Spanish scientist *Juan de la Cierva y Codorníu*. The basic ideas are

```
author = de la Cierva {\lowercase{Y}} Codorn{\'\i}u, Juan
```

or

```
author = de la Cierva {y} Codorn{\'\i}u, Juan
```

However, these solutions yields labels such as `CYC` or `CyC`, where we would prefer `CC`.

Several solutions are possible:

```
author = de la Cierva{ }y Cordon{\'\i}u, Juan
```

or

```
author = de la {Cierva y} Cordon{\'\i}u, Juan
```

Both solutions work: In the first case, BibTEX won't see the space, and considers that the `y` belongs to the previous word. In the second case, `Cierva y` is at brace-level 1, and thus goes into the `Last` part, which has priority over the `von` part.

13.4 How to remove space between `von` and `Last`?

Here the example will be *Jean d'Ormesson*. The best way to encode this appears to be

```
author = "d'\relax Ormesson, Jean"
```

Indeed, the `\relax` commands will gobble spaces until the next non-space character.

13.5 How to get *et al.* in the author list?

Some special bibliography styles will automatically replace long lists of authors with the name of the first author, followed by *et al.* However, standard BibTEX styles won't. You can however get the same result by using the special name `others`. For instance, if you enter

```
author = "Dupont, Jean and others"
```

you get "Jean Dupont *et al.*" in the resulting bibliography.

13.6 The `key` field

It may happen that no author is given for a document. In that case, bibliography styles such as alpha.bst, when computing the "label" of such an entry, will use the `key` field (the first three letters for alpha.bst, but the entire field for apalike.bst, for instance). When no `key` is given, the first three letters of the internal citation key are used.

Is it ok with everyone? Yes ?! Right, we can go to the next, most exciting part of this doc: How to create or modify a bibliography style...

Part 4

Bibliography style (`.bst`) files

Table of Contents

14 What's that?

The `.bst` file is responsible for building the reference list. It is crucial to distinguish between the role of BibTEX and the role of LATEX, and, inside the role of BibTEX, between what is done by the style file and what should be done in the `.bib` file.

Roughly speaking, BibTEX reads three files, in the following order: The `.aux` file first, where it finds the name of the style file to be used, the name of the database(s), and the list of cited entries. The `.bst` file is read next, and the `.bib` file last.

The `.bst` file tell BibTEX what to do with each cited entry. Namely:

- Which entry-type are defined;
- Which fields are mandatory, or just allowed, depending on the entry;
- And, mainly, how to handle all that stuff in order to produce a clean, LATEX-able bibliography.

In view of this, BibTEX uses a language with two types of instructions: The so-called *commands*, first, and the *internal functions*. The following section deals with commands, and the other ones with internal functions.

15 The structure of a `.bst` file

Here is, roughly speaking, the structure of a `.bst` file:

```
ENTRY
  {  ...  }
  {  ...  }
  {  ...  }

INTEGERS {  ...  }
STRINGS {  ...  }

MACRO {  ...  }{  ...  }
FUNCTION {  ...  }{  ...  }

READ
EXECUTE {  ...  }
ITERATE {  ...  }
SORT
ITERATE {  ...  }
REVERSE {  ...  }
EXECUTE {  ...  }
```

This example shows all the commands understood by BibTEX. Here is their meaning:

`ENTRY` : This command defines the list of all possible fields. More precisely, it takes three arguments (between braces), defining the list of possible "external" (string) entries[28], the list of "internal" integer variables, and a list of "internal" string variables. Those variables are used for internal computations of BibTEX, for instance when building the label with the alpha.bst style.

Inside braces, variable names are separated with spaces. I shall also mention that BibTEX does not distinguish between upper- and lowercase in the names of functions and variables. I will generally write command names in uppercase, and everything else in lowercase.

For instance, plain.bst starts with:

```
ENTRY
  { address
    author
    booktitle
    ...
    volume
    year
  }
  {}
  { label }
```

This defines the classical fields, plus an internal string variable for the label. In fact, plain.bst uses numbers as labels, but remember that the length of the longest label has to be evaluated and given as argument to the `thebibliography` environment. This is the only reason why this `label` variable is defined.

Note that `ENTRY` must appear once and only once in each style file.

[28] Except the `crossref` field, which is added automatically by BibTEX.

`INTEGERS` : This command declares integer variables[29]. The argument of this command is a space-separated list of names.

`STRINGS` : This commands defines string variables in the same way as `INTEGERS` defines integer variables. Contrary to integer variables, string variables are very "expensive", and BibTEX limits the number of such variables to twenty. You should really spare string variables if you plan to develop a large style file.

`MACRO` : This command defines abbreviations, in the same way as **@string** does[30]. If an abbreviation is defined in both the `.bst` and in the `.bib` file, the definition in the `.bib` file is used. Also note that you can define **@string** definitions involving `MACRO` definitions. But this is probably not a good advice...

As regards the syntax, `MACRO` requires two arguments, the first one being the name of the abbreviation to be defined, the second one being the definition itself. It must be a string surrounded with double-quotes. As an example, standard style files define the following:

```
MACRO {jan} {"January"}
MACRO {feb} {"February"}
MACRO {mar} {"March"}
...
```

`FUNCTION` : The most useful command: It allows to define macro functions that will be executed later on. The first argument is the name of the function, the second one is its definition. No example for the moment since there will be numerous ones in the following.

`READ` : As for `ENTRY`, this command must occur exactly once in any `.bst` style file. This is not surprising since it tells BibTEX to read the `.bib` file: Reading it several time would not change anything, and not reading it would seriously limit the interest of the bibliography style. So, that command extracts from the `.bib`-file the entries that are cited in the `.aux` file. Commands `ENTRY` and `MACRO` should appear before `READ`, while `ITERATE` and `REVERSE` may only appear after.

`EXECUTE` : This executes the function whose name is given as argument. Note that the function must have been defined earlier. The argument could also be a sequence of BibTEX internal functions. For some more comments, see the description of `ITERATE` below.

`ITERATE` : This commands also execute the function given in the argument, but contrary to `EXECUTE`, the function is executed as many times as the number of entries imported by `READ`. The function may use the fields of each entry. `EXECUTE` only executes its argument once, and it can not use any field of any entry.

`SORT` : This command sorts the entries according to the variable `sort.key$`. That special variable is a string variable, implicitly declared for each entry, and that can be set by `ITERATE`ing a function. The entries are then sorted alphabetically according to that string. Of course, some styles won't sort the entries, in which case they will appear in the same order as in the document.

`REVERSE` : Like `ITERATE`, but from the last entry to the first one.

That's all for the list of commands. You probably can imagine how this will be set up. It remains to define the relevant `FUNCTION`s... This is the subject of the next two sections: We will first see the notation used by BibTEX, and then how to define functions.

[29] Those variables are not linked with any entry, contrary to variables defined in the second argument of `ENTRY`.
[30] Note that **@string** has nothing to do with `STRINGS`...

16 Reverse Polish Notation

BibTeX uses the so-called Reverse Polish Notation. This is the main reason why people say that BibTeX language is hard to understand. It is a stack-based language: You put arguments on a stack (think of a stack of plates), and each function takes as many arguments as necessary on the top of the stack, and replace them with its result. Addition, for instance, will take the first two elements on the stack, add them, and put the result on the top of the stack. Another example[31]: `1 3 5 + 2 3 * -` is executed as follows:

- `1` is put on the stack. Assuming the stack was initially empty, it now contains `1`;
- `3` is then put on the stack, which now contains `1` and `3` (`3` being on the top of the stack);
- `5` is added on the stack, which now contains `1`, `3` and `5`;
- `+` is a binary operator: it reads (and removes) the topmost two elements on the stack, which are `5` and `3`, adds them, and put the resulting value, `8`, on the top of the stack. The stack is now `1`, `8`, with `8` being on the top;
- `2` is put on the top of the stack;
- `3` is put on the stack. It now contains `1`, `8`, `2` and `3`.
- `*` is applied to the topmost two elements, which are removed. The resulting value[32] $3 \times 2 = 6$ is put on top of the stack, which is now made of `1`, `8` and `6`.
- `-` is now applied to `6` and `8`. As defined in BibTeX [33], the first value, `6`, is subtracted to the second one, `8`. Finally, the stack contains `1` and `2`.

If you did not understand this example, you'd better re-read it, or try to find more informations somewhere. This is really the core of BibTeX language, and you'll probably won't understand anything below if you did not understand this example.

But for most of you, it should be ok, and we can tackle the most exciting part.

17 Internal functions

The table below describes all internal functions. For each of them, I give

- on the left, the items it requires to be present on the top of the stack, the rightmost item being the uppermost on the stack;
- on the right, what the function puts on the top of the stack.

I will use the following conventions: $\mathcal{I}$ represents an integer, $\mathcal{S}$ is a string, $\mathcal{F}$ is a function, $\mathcal{N}$ is a variable name[34], $\mathcal{C}$ is the name of a field declared with `ENTRY`[35], and $\mathcal{E}$ is an item that can be either a string or an integer..

$\mathcal{I}_1\ \mathcal{I}_2$	`+`	$(\mathcal{I}_1{+}\mathcal{I}2)$	integer addition[36];

[31] This part is crucial, that's why I insist on it. But if you really understood how it works, you can skip the explanation.

[32] We assume that $*$ is the multiplication in that example, but it is not the case in BibTeX. In fact, there is no multiplication operator in BibTeX, as we will see later, and $*$ is the concatenation on strings.

[33] This operator is not commutative, and it could have been defined in the other way.

[34] A variable name is a name that has been declared with `STRINGS` or `INTEGERS`, or with `ENTRY`. Moreover, it must be preceded with a single quote, so that BibTeX understands that you mean the name of the variable and not its value. For instance, `'label`.

[35] It could also be `crossref`, which is implicitly declared by BibTeX.

[36] Integers must be preceded with a `#`. For instance, if you want to compute $2 + 5$, you'll enter `#2 #5 +`. Negative numbers are entered with `#-3`, for instance.

$\mathcal{I}_1\ \mathcal{I}_2$	`-`	$(\mathcal{I}_1 - \mathcal{I}_2)$	integer subtraction;
$\mathcal{I}_1\ \mathcal{I}_2$	`>`	$\mathcal{I}$	returns 1 if $\mathcal{I}_1$ is strictly greater than $\mathcal{I}_2$, and 0 otherwise[37];
$\mathcal{I}_1\ \mathcal{I}_2$	`<`	$\mathcal{I}$	returns 1 if $\mathcal{I}_2$ is strictly greater than $\mathcal{I}_1$, and 0 otherwise;
$\mathcal{I}_1\ \mathcal{I}_2$	`=`	$\mathcal{I}$	returns 1 if both integers are equal, and 0 otherwise;
$\mathcal{S}_1\ \mathcal{S}_2$	`=`	$\mathcal{I}$	returns 1 if both strings are equal, and 0 otherwise;
$\mathcal{S}_1\ \mathcal{S}_2$	`*`	$(\mathcal{S}_1\mathcal{S}_2)$	concatenates both strings[38];
$\mathcal{E}\ \mathcal{N}$	`:=`		assignment operation. Provided that variable $\mathcal{N}$ has the same type as item $\mathcal{E}$, it is set to that value;
$\mathcal{S}$	`add.period$`	$\mathcal{S}$	adds a period at the end of string $\mathcal{S}$, except if $\mathcal{S}$ already ends with a period, or an exclamation or question mark (after removing right braces);
	`call.type$`		executes the function whose name is the type of the current entry. For instance, for an entry **@book**, it runs the function `book`. Of course, this command cannot be executed through an `EXECUTE` command, it can only be called with `ITERATE` or `REVERSE`. This explains how to add new entry-types: you simply have to define a function whose name is the entry-type. If an entry-type is used and the corresponding function has not been defined before the `READ` command, BibTEX will complain;
$\mathcal{S}$ `"t"`	`change.case$`	$\mathcal{S}$	turns $\mathcal{S}$ to lower case, except for the first letter and for letters that appear at brace-depth strictly positive. Remember that special characters are at depth 0;
$\mathcal{S}$ `"l"`	`change.case$`	$\mathcal{S}$	turns $\mathcal{S}$ to lower case, except parts that are at strictly positive brace-depth;
$\mathcal{S}$ `"u"`	`change.case$`	$\mathcal{S}$	turns $\mathcal{S}$ to upper case, except...
$\mathcal{S}$	`chr.to.int$`	$\mathcal{I}$	if $\mathcal{S}$ contains only one character (in the classical sense, *i.e.* not considering special characters), returns its ASCII code;
	`cite$`	$\mathcal{S}$	puts on the stack the internal key of the current entry. Only meant to be used with `ITERATE` or `REVERSE`, of course;
$\mathcal{E}$	`duplicate$`	$\mathcal{E}\ \mathcal{E}$	adds a copy of the first item on the top of the stack;
$\mathcal{E}$	`empty$`	$\mathcal{I}$	adds 1 on the top of the stack if $\mathcal{E}$ is an empty string or an empty (but defined) field, and 0 otherwise;
$\mathcal{S}_1\ \mathcal{I}\ \mathcal{S}_2$	`format.name$`	$\mathcal{S}$	extract the $\mathcal{I}$-th name of string $\mathcal{S}_1$ (in which names are separated by `and`), and formats it according to specification given by string $\mathcal{S}_2$. I can't explain all that here, see the next section for more details;
	`global.max$`	$\mathcal{I}$	put on the stack the maximal length allowed for strings. This is useful to ensure that a string obtained by concatenation is not too long, but the value is generally quite high (5000 characters);
$\mathcal{I}\ \mathcal{F}_1\ \mathcal{F}_2$	`if$`		execute $\mathcal{F}_1$ if $\mathcal{I}$ is strictly positive, and $\mathcal{F}_2$ otherwise;

[37] There is no boolean type in BibTEX, because integers are sufficient. Negative numbers (including 0) mean `false`, and strictly positive mean `true`.

[38] As mentioned earlier, there is not multiplication in BibTEX[39].

[39] We will define one later, since it may be useful.

$\mathcal{I}$	`int.to.chr$`	$\mathcal{S}$	turns $\mathcal{I}$ to its corresponding character in the ASCII table. $\mathcal{I}$ must be between 0 and 127;
$\mathcal{I}$	`int.to.str$`	$\mathcal{S}$	turns an integer to the equivalent string[40];
$\mathcal{C}$	`missing$`	$\mathcal{I}$	returns 1 if field $\mathcal{C}$ has been defined in the current entry, and 0 otherwise;
	`newline$`		flushes the output buffer to the `.bbl` file, and starts a new line;
$\mathcal{S}$	`num.names$`	$\mathcal{I}$	returns the number of names in string $\mathcal{S}$. This is done by counting the number of occurrences of `and`, surrounded with spaces and at level 0, and adding 1;
$\mathcal{E}$	`pop$`		removes the topmost item on the stack;
	`preamble$`	$\mathcal{S}$	puts on the stack a string made of the concatenation of all `preamble` declarations found in the `.bib` file;
$\mathcal{S}$	`purify$`	$\mathcal{S}$	removes all non-alphanumeric characters of string $\mathcal{S}$. More precisely, this functions preserves letters and numbers and spaces, replaces each tabulation, hyphen and tilde with a space, and removes other standard characters, namely characters not listed above and appearing in the table given in appendix C of the TEXbook. Special characters are an exception: LATEX commands are removed, as well as spaces, tabulations, hyphens and tildes. Only numbers and letters appearing outside LATEX command names are preserved. Practically, this commands is used for cleaning strings before comparing them, when sorting entries. This is why it is important to keep only "known" characters. I insists on the fact that "`\'e`" and "`é`" are different, from this point of view, since the first one will be transformed into an `e`, while the second one will be unchanged, but, when sorting, will be put after the standard 26 letters.
	`quote$`	$\mathcal{S}$	adds the 1-character string `"` on the stack;
	`skip$`		does nothing;
	`sort.key$`	$\mathcal{S}$	this function is in fact a string variable, implicitly declared for each entry, and that is used by `SORT`. It must therefore be defined accordingly before sorting;
... $\mathcal{E}$ $\mathcal{E}$	`stack$`		clears the stack, and prints its contents on standard output (not in the output file);
$\mathcal{S}$ $\mathcal{I}_1$ $\mathcal{I}_2$	`substring$`	$\mathcal{S}$	extracts from $\mathcal{S}$ the string of length $\mathcal{I}_2$ starting at position $\mathcal{I}_1$. By convention, the first character is at position 1. Note that special characters and braces are not considered here, so that the substring of `{\LaTeX}` of length 3 starting at position 2 is `\LaT`;
$\mathcal{E}_1$ $\mathcal{E}_2$	`swap$`	$\mathcal{E}_2$ $\mathcal{E}_1$	swaps both topmost items on the stack;
$\mathcal{S}$	`text.length$`	$\mathcal{I}$	returns the number of characters of $\mathcal{S}$, special characters being considered as only one character, and braces being omitted;

[40]Scientists would remark that I did not define the equivalence relation I deal with. I think an example will be sufficient: if $\mathcal{I}$ is 147, the equivalent string is `"147"`.

$\mathcal{S}\ \mathcal{I}$	`text.prefix$`	$\mathcal{S}$	returns the first $\mathcal{I}$ characters of $\mathcal{S}$, still considering special characters as only one character, and omitting braces. Braces are output, however. For instance, the prefix of length 3 of `{{\LaTeX}}1234` is `{{\LaT}}`, while it is `{\LaTeX}12` for `{\LaTeX}1234`;
$\mathcal{E}$	`top$`		echoes the first item of the stack on standard output (not in the output file) and removes it from the stack;
	`type$`	$\mathcal{S}$	adds on the stack the type of the current entry;
$\mathcal{S}$	`warning$`		writes $\mathcal{S}$ in a warning message on the standard output;
$\mathcal{F}_1\ \mathcal{F}_2$	`while$`		successively executes $\mathcal{F}_1$ and $\mathcal{F}_2$ while $\mathcal{F}_1$ returns a strictly positive value. $\mathcal{F}_1$ has to return an integer;
$\mathcal{S}$	`width$`	$\mathcal{I}$	returns the length of string $\mathcal{S}$, in hundredths of a point, when written using font *cmr*10 of June 1987. You probably don't mind the details... This function is used for comparing widths of labels, the longest label being passed as the argument of the `thebibliography` environment;
$\mathcal{S}$	`write$`		write $\mathcal{S}$ in the output `.bbl` file.

We're done. This should be sufficient for defining plenty of functions for managing entries, sorting them, label them and typeset them. Before using them, I promised to explain how `format.name$` works.

18 The `format.name$` function

This function is quite complicated. The syntax is as follows:

$\mathcal{S}_1\ \mathcal{I}\ \mathcal{S}_2$ `format.name$` $\mathcal{S}$

The first argument $\mathcal{S}_1$ is generally the content of fields `author` or `editor`. It is a list of names separated with `and`. If names are in one of the three formats recognized by BibTEX (see section 11), BibTEX will be able to detect the last name, first name, “von” and “complement”. String $\mathcal{S}_2$ defines the output format. Here is an example:

```
"{ff }{vv }{ll}{, jj}"
```

The format of such a specification is defined as follows:

- globally, it is a list of strings, each one being surrounded with braces;
- `ff` represents the `First` part, *i.e.* the first name; `ll` is the `Last` part, *i.e.* the last name, and `vv` and `jj` are the `von` and `jr` part of the name;
- `format.name$` replaces, in each string of list $\mathcal{S}2$, strings `ff`, `ll`, `vv` and `jj`, with their respective values. If a value is empty, the complete string is removed. Otherwise, the other characters in the string are output. In the example above, for instance, if the `jr` part is empty, the comma won't be output, and if the `First` part is not empty, it will be followed with a space.

This should be almost clear, now. I just mention, but you've probably already understood, that the integer $\mathcal{I}$ indicates that we are working on the $\mathcal{I}$-th name of $\mathcal{S}_1$.

Last, there exists abbreviated forms for name formats. They are used to extract the first letter of first names, for instance. It suffices to replace `{ff }` with `{f }`. For multiple names in one part (several first names, for instance), the first letter of each name is written, followed with a period.

Different names must be separated with a space, a tabulation, a tilde or an hyphen. Tabulations are turned to spaces, tildes and hyphens are preserved. Note that there is no period at the end, and it must be specified by hand by writing `{f. }`. For instance

```
"Goossens, Michel and Mittelbach, Franck and Samarin, Alexander"
#2 "{f. }{vv }{ll}{, jj}" format.name$
```

returns *F. Mittelbach*. Let's have a look at several other interesting examples:

```
"Charles Jean Gustave Nicolas de La Vall{\'e}e Poussin"
#1 "{vv }{ll}{, f}" format.name$
```

yields *de La Vallée Poussin, C.J.G.N* without the final period, since we did not ask for it.

```
"Doppler, {\relax Ch}ristian Andreas"
#1 "{f. }{vv }{ll}" format.name$
```

gives *Ch. A. Doppler*. Last,

```
"Jean-Baptiste Poquelin" #1 "{f. }{vv }{ll}" format.name$
```

outputs *J.-B. Poquelin*.

You probably remarked that one problem is still unsolved: For creating labels from author names, we will of course use `format.name$`, and we'd like to get `LVP`, say, for a book by La Vallée Poussin. The first idea, of course, is:

```
"Charles Jean Gustave Nicolas de La Vall{\'e}e Poussin"
#1 "{l}" format.name$
```

which unfortunately gives *L. V. P*, which would look awful as a label. But in fact, the period added between names is a default value, but can be explicitly given. In our case, we would like to replace the period with empty strings:

```
"Charles Jean Gustave Nicolas de La Vall{\'e}e Poussin"
#1 "{l{}}" format.name$
```

which gives the expected *LVP*. Braces following `l` mean that nothing should be put between names or letters. It is also possible to add some text only once, before or after the names. Spaces can be entered directly, while text and commands should be put between braces, so that BibTEX distinguished them with `ff`, `ll`, ... specifications. An example should make this clear:

```
"Charles Jean Gustave Nicolas de La Vall{\'e}e Poussin"
#1 "{{\scshape\bgroup}ff{ }{\egroup}" format.name$
```

gives `{\scshape\bgroup}Charles Jean Gustave Nicolas{\egroup}`, and thus « CHARLES JEAN GUSTAVE NICOLAS ». Note that we had to trick BibTEX and LATEX regarding the `\bgroup`: braces, which are necessary in BibTEX name specifications, are copied verbatim in the output...

Well... Once again, I could stop here, since you know everything, at least from the theoretical point of view. But this would not be a good idea. So the next sections are devoted to some examples and tricks that could be useful.

19 Some practical tricks

In fact, the main trick to know is how to build a minimal file, in order to test BibTEX's behavior, for instance for testing the encoding of a name. This basic style file will simply output its result on the standard output. Assume you want to see how BibTEX cuts the following name:

```
de la Cierva y Codorn{\'\i}u, Juan
```

The minimal `.bst` file must contain `ENTRY` and `READ` commands, even if we won't use any entry. It must also compute the desired result. Our style file, `min.bst`, will be:

```
ENTRY {}{}{}

FUNCTION {test}
{"de la Cierva y Codorn{\'\i}u, Juan"
#1 "{ff - }{vv - }{ll}" format.name$ top$}

READ

EXECUTE{test}
```

This will split the name and write the different parts, separated with hyphens.

The `.aux` file now: it will simply define the style file to be used:

```
\bibstyle{min}
```

This file, called `min.aux`, contains neither the `\bibdata` precising the `.bib` file to use, nor the `\citation`s declaring the entries to be extracted. BibTEX will, of course, complain about that, but will continue the execution anyway.

The result is as follows:

```
% bibtex min
This is BibTeX, Version 0.99c (Web2C 7.4.5)
The top-level auxiliary file: min.aux
The style file: min.bst
I found no \citation commands---while reading file min.aux
I found no \bibdata command---while reading file min.aux
Warning--I didn't find any fields--line 1 of file min.bst
Juan - de~la Cierva~y - Codorn{\'\i}u
(There were 2 error messages)
```

We get the error messages, but also the result, showing that our encoding is not correct (since the `von` part should only contain `de la`). As regards unbreakable spaces: BibTEX automatically adds unbreakable spaces instead of the first and last spaces.

This provides you with a way to test all your future BibTEX functions, which can be quite useful since BibTEX error messages are generally difficult to understand. You can also try to test if `purify$` behaves as I said previously.

```
FUNCTION {not}
{   { #0 }
    { #1 }
  if$
}

FUNCTION {and}
{   'skip$
    { pop$ #0 }
  if$
}

FUNCTION {or}
{   { pop$ #1 }
    'skip$
  if$
}
```

Figure 1: Basic boolean functions

20 Some small simple functions

20.1 Boolean functions

As you know, BibTEX does not know about booleans. In particular, there is no negation, conjunction or disjunction, while it would be quite useful. Those operations are defined on figure 1.

Let me just explain the first two: For negation, you assume there is a boolean (*i.e.* an integer) on the top of the stack. Function `if$` will test that boolean. If it evaluates to `true`, the function returns `#0` (*i.e.* `false`), and otherwise it returns `#1` (*i.e.* `true`). This is precisely what we wanted.

For the conjunction: we have two booleans on the stack. Function `if$` tests the first one. If it is `true`, then we do nothing (thus leaving the second boolean on the top of the stack, it's value being the result of the conjunction). If it is `false`, the conjunction is false: We remove the second argument and put `#0` on the top of the stack.

20.2 Multiplication

Figure 2 shows how to compute the multiplication of two integers. I won't enter into the details, comments should be sufficient. In the same way, you can try to define integer division, gcd, primality testing, ... But I'm not sure if it is useful. Multiplication can be, as you'll see below.

20.3 Converting a string to an integer

There is no direct way for converting a string (made of digits) to the corresponding integer. We will define a function achieving this, and returning an error message whenever the string is not a number. This is done recursively, as shown on figure 3.

There are several interesting points: first, it requires boolean functions, and multiplications. Those functions must thus be added before the above definitions. Moreover, function `chr.to.value` emphasizes an important rule of thumb: A function must always behave in the same way as regards the stack: Same type and numbers of inputs, same type and number of outputs. For instance, here, if the

```
INTEGERS { a b }

FUNCTION {mult}
{
 'a :=                        %% we store the first value
 'b :=                        %% we store the second value

 b #0 <                       %% We remember the sign of b, and
    {#-1 #0 b - 'b :=}        %% then consider its absolute value.
    {#1}                      %%
 if$                          %%

 #0                           %% Put 0 on the stack.
 {b #0 >}                     %% While b is strictly positive,
 {                            %% we add a to the value on the stack
   a +                        %% and decrement b.
   b #1 - 'b :=               %%
 }                            %%
 while$                       %%

 swap$                        %% Last, we take the opposite
   'skip$                     %% if b was negative.
   {#0 swap$ -}               %%
 if$                          %%
}
```

Figure 2: The "multiplication" function

character is not an integer, we complain, but we return an integer, as if everything was ok. This rule is very important unless you want to spend some time in debugging.

20.4 Counting the number of characters in a string

As we saw on page 32, `text.length$` counts the number of characters in a string, but assuming BibTEX's meaning of "character". In fact, the only function that assumes the standard definition of a character is `substring$`. We will use it repeatedly for counting the number of characters of a string, by computing the successive prefixes until they equal the original string.

As an exercise, the reader can try to implement another algorithm: removing one after the other the first characters of the string until the resulting string is empty.

20.5 A "search-and-replace" function

With the function above, we can now design an algorithm for searching and replacing the occurrences of a string in a given text. Once again, we cannot use `text.length$` and `text.prefix$` since they assume a special definition of a character. We only use the function `string.length` and `substring$`. The idea is to successively extract the first n characters of the text, compare them with the pattern, and replace it or drop the first character, depending on the result of the comparison.

This function replaces all the occurrences of the pattern, but it can easily be adapted for replacing only the first occurrence. As an exercise, the reader can try to implement the same algorithm using

```
FUNCTION {chr.to.value}       %% The ASCII code of a character
{
  chr.to.int$ #48 -           %% ASCII value of "0" -> 48
  duplicate$ duplicate$       %%                "1" -> 49
  #0 < swap$ #9 > or          %%                   ...
  {                           %%                "9" -> 57
      #48 + int.to.chr$
      " is not a number..." *
      warning$                %% Return 0 if it is not a number
      pop$ #0                 %%
    }
  {}
  if$
}

FUNCTION {str.to.int.aux}     %% The auxiliary function
{
  {duplicate$ empty$ not}     %% While the string is not empty
    {                         %% consider its first char
      swap$ #10 mult 'a :=    %% and ``add'' it at the end of
      duplicate$ #1 #1 substring$   %% the result.
      chr.to.value a +
      swap$
      #2 global.max$ substring$
    }
  while$
  pop$
}

FUNCTION {str.to.int}
{                             %% Handling negative values
  duplicate$ #1 #1 substring$ "-" =
    {#1 swap$ #2 global.max$ substring$}
    {#0 swap$}
  if$
                              %% Initialization, and then
  #0 swap$ str.to.int.aux     %% call to the aux. funtion
  swap$
    {#0 swap$ -}              %% And handle the sign.
    {}
  if$
}
```

Figure 3: Converting a string to an integer

```
INTEGERS{ l }
FUNCTION{ string.length }
{
  #1 'l :=
  {duplicate$ duplicate$ #1 l substring$ = not}
    {l #1 + 'l :=}
  while$
  pop$ l
}
```

Figure 4: A function for counting the *real* number of characters

```
STRINGS{replace find text}
INTEGERS{find_length}
FUNCTION{find.replace}
{ 'replace :=
  'find :=
  'text :=
  find string.length 'find_length :=
  ""
    { text empty$ not }
    { text #1 find_length substring$ find =
        {
          replace *
          text #1 find_length + global.max$ substring$ 'text :=
        }
        { text #1 #1 substring$ *
          text #2 global.max$ substring$ 'text :=
        }
      if$
    }
  while$
}
```

Figure 5: Search-and-replacing with BibTEX

only two string variables[41] (it may be important to spare string variables since their number is very limited).

[41]Hint: try to leave `find` on the stack.

Part 5
Other use of BibTEX

Table of Contents

Here are some general extensions, showing how to use BibTEX for other purposes. I'm developing some of those extensions, and you can get the latest news on my web page:

http://www.lsv.ens-cachan.fr/~markey/bibla.php.

Don't hesitate to give me your comments and ideas.

21 A publication list

This is not really a wide extension of the use of BibTEX, since it still deals with bibliography. In this example, we use classical `.bib` files in order to extract only publications by one single person, indicating the names of the possible other authors. This gives, for instance:

Books by Michel Goossens

[1] *the LATEX Companion*. Addison-Wesley, 1993. Joint work with Frank Mittelbach and Alexander Samarin.

[2] *the LATEX Graphics Companion*. Addison-Wesley, 1999. Joint work with Sebastian Rahtz and Frank Mittelbach.

[3] *the LATEX Web Companion*. Addison-Wesley, 1999. Joint work with Sebastian Rahtz.

There are three important points here:

- In this case, the entries to be cited are extracted by the style file;
- Typesetting of names is a little bit different.

The third difference consists in the way the style is configured: Instead of asking the user to modify the `.bst` file, which is rarely desirable, we add a new entry-type, named **@config**. A **@config** entry will define the name of the author whose publications are to be extracted, as well as some other configuration variables.

22 An addressbook

A list of addresses is a database, that has to be sorted, grouped and typeset. BibTEX should be able to do that...

To that aim, we have to completely redefine the fields, entry-types, and functions. Fields include names, addresses, phone numbers, e-mail addresses, ... Here is an example of the output:

MARKEY Nicolas	LSV – ENS Cachan 61, avenue du Président Wilson 94235 CACHAN Cédex `markey@lsv.ens-cachan.fr`	☎ 🖷	01.47.40.75.32 01.47.40.24.64

mn

As in the previous example, I defined a **@config** entry-type in order to define how letters should be grouped together, ...

23 Exporting, extracting and cleaning bibliographic entries

It sometimes occurs that you have to extract a part of your bibliographic database, for instance for sending it together with a LATEX document. Several tools already exist for doing this, but they generally can't handle crossrefs, macros, string abbreviations, ... If fact, BibTEX can do this on its own: You simply have to ask the bibliography style to format its output as a `.bib` database (The result will be named `.bbl`, since there is no way to tell BibTEX to write somewhere else). This technique has many advantages: It ensures that the fields are treated exactly as BibTEX would treat them with another style file; abbreviations are automatically expanded; It is easy to extract all the references by one author (remember that names could be encoded in many different manners inside a `.bib` database), ... The package `bibexport` implements these techniques: `http://www.ctan.org/tex-archive/biblio/bibtex/contrib/bibexport/`

24 Using several languages in one bibliography

Classical bibliography style only handle one language, generally English. Handling several languages is quite tricky: First, when filling the `.bib` file, you must ensure not to write language-dependant data. Namely, for instance, months should be entered numerically. Moreover, BibTEX adds some words while processing the entries, such as the "and" before the last author name.

There are several solution for handling several languages:

- Designing one style file per language. This is somewhat tiring to do, but not really complicated, and many such translations already exist.

  French styles are available on `http://www.ctan.org/tex-archive/biblio/bibtex/contrib/bib-fr/`. In those styles, I put every translated word at the beginning of the file, so that it is easy to adapt the translation.

- Another solution is to replace language-dependant words with a LATEX command. This is done in babelbib (`http://www.ctan.org/tex-archive/biblio/bibtex/contrib/babelbib/`), developed by Harald Harders. Those style files require that you use the package babel.sty, and allows you to define one language for each entry (and thus to use the right hyphenation patterns).

25 A glossary

A glossary is a dictionary defining technical terms that appear in a document. Glossaries are generally made using the index mechanisms of LATEX. But BibTEX can also do that (and even better than with `\index`, since it could extract definitions from a general dictionary). Moreover, it is possible to add reference to pages where the term is used by using the backref.sty package. See for instance the package gloss.sty, developed by Jose Luis Diaz de Arriba and Javier Bezos.

References

[Ber02] Jens Berger. *The **jurabib** package*, September 2002.

[Dal99a] Patrick W. Daly. *A LaTeX Package to Place Bibliography Entries in Text*, February 1999.

[Dal99b] Patrick W. Daly. *A Master Bibliographic Style File*, May 1999.

[Dal99c] Patrick W. Daly. *Natural Sciences Citations and References*, May 1999.

[Dom97] Éric Dommenjoud. *The **footbib** package*, March 1997.

[GMS93] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley, December 1993.

[Han00a] Thorsten Hansen. *The **bibunits** package*, October 2000.

[Han00b] Thorsten Hansen. *The **multibib** package*, January 2000.

[Lam97] Leslie Lamport. *LaTeX: a Document Preparation System*. Addison-Wesley, March 1997.

[Mar05] Nicolas Markey. *The **splitbib** package*, February 2005.

[MGB+04] Frank Mittelbach, Michel Goossens, Johannes Braams, David Carlisle, and Chris Rowley. *The LaTeX Companion — Second Edition*. Addison-Wesley, April 2004.

[Pat88a] Oren Patashnik. BibTeXing, February 1988.

[Pat88b] Oren Patashnik. *Designing* BibTeX *styles*, February 1988.

LaTeX

BibTeX