For submission to The PracTEX Journal
Article revision 2008/02/07

Using BiBTeX to produce customized layouts

Yogeshwarsing Calleecharan

Email yogeshwarsing@passagen.se

Abstract Normal LaTeX and TeX usage does not require touching existing .bst files nor creating new ones. However, BibTeX offers several interesting commands which can be used to do many things apart bibliography generation. In this article, it is demonstrated how customized layouts for a database can be created without much trouble.

1 Introduction

To many LaTeX users, BibTeX and its commands remain a mystery. Yet as pointed out in [1], LaTeX users can play with BibTeX commands and use it for purposes other than bibliography generation. Since I started using LaTeX in 2005, I realized that handling LaTeX and TeX as programming languages can really make one abandon traditional word processing packages as the sky certainly becomes the limit.

The article [1] is regarded as the most exhaustive source of information on BibTeX. BibTeX is a powerful tool for handling a database and in this article, I have a modest goal: To show how BibTeX can be used to create customized layouts for a database. I am sure that any LaTeX user will be able to create more interesting layouts given here and to adapt them for their own needs.

2 A Simple Example

Programming books often begin their first program with the traditional words `Hello World!` displayed on the screen. We shall remain faithful to this tradition and use BibTeX to this end. By displaying just two words, the codes shown next serve to outline the minimal structure of a `.bst` file which is the key file in creating any desired style or layout.

Normally where a `.bib` file is involved, it is useful to run the minimal commands: LaTeX BibTeX LaTeX LaTeX in succession. I use the TeXMAKER [2] IDE to do the compilation and the IDE built-in `Quick Build` tool permits constructions of macro-like commands that save button presses. Further time can be saved by giving the same filename to the `.tex`, `.bib` and `.bst` files as in `first.tex`, `first.bib` and `first.bst` as this allows running the `Quick Build` tool on any of these files.

Enough of talking now! Displaying the words `Hello World!` has nothing to do with formatting a database of information but it can be useful to illustrate the structure of a `.bst` style file. For a database one would normally start with the `.bib` file and think of the fields required for different entry types. But here for this simple example, we directly create the `.bst` file as one entry is enough to display `Hello World!`. We will need to create three files: `first.bst`, `first.bib` and `first.tex`. Next is shown the `first.bst` file listing:

```
1  ENTRY
2    { toDisplay
3      } {} {}
4  FUNCTION{print}
5  {
6   cite$ pop$
7  }
8  FUNCTION{style1}{
9  print toDisplay write$
10 }
11 FUNCTION {fin}
12 {newline$
13 }
14 READ
15 ITERATE{call.type$}
16 EXECUTE {fin}
```

We shall give a short attempt to dissect the `first.bst` file listing and detailed information of the commands used are available in [1]. We begin on line 1 with the BibTeX `ENTRY` command which is used to create a field named `toDisplay`

which will be defined later in the `first.bib` file. The two pairs of matching braces on line 3 are for internal variables and will not be discussed further. The next important command is `FUNCTION`. All the macro functions in our example are user-defined. Furthermore, a function can also be made to run via the `EXECUTE` command (refer to line 16).

The `FUNCTION` command takes as first argument the name of the function and as second argument its definition which usually contains a useful sequence of instructions. On line 9 we find the internal command `write$` which interprets the sequence of instructions between the function argument name `print` and itself on the same line, and it writes the output in a `.bbl file`, which is generated on running BibTeX. On line 4 is the first function called `print` which will pop out the topmost item from the stack via the `pop$` command. The user can change the argument name `print` on line 4 to anything that he or she likes e.g. `rambo` provided the same argument name is written on line 9 as well. But as in programming, it is always better to work with meaningful names. Finally, the other important function is on line 8 and it takes as first argument `style1` and defines the sequence of instructions on line 9 as second argument. This function also defines an entry type (`style1`) which shall appear in the `first.bib` file as well.

Moving further down in the file listing `first.bst` to line 14 is the `READ` command which is advisable to be placed after all functions have been defined. Occurring just once in a `.bst` file, it processes only from the `first.bib` file the entries listed in the `first.aux` file (generated on LaTeX compilation). Next on line 15 is the `ITERATE` command which should come after the `READ` command. This command is executed for a number of times that equals to the number of entries in the `first.bib` file (subject that an entry is called in the `first.tex` file). In this first example, there is just one entry as will be seen later in the file `first.bib`. The `ITERATE` command has as argument `call.type$` which looks for the name of each entry to be executed in the `first.bib` file and takes this name as argument. And this `call.type$` command works together with the `cite$` on line 6 to treat each entry in the `first.bib` file in turn.

Finally on line 16 is the `EXECUTE` command which is required to generate an output from a `.bst` file. It has been found necessary to add an arbitrary function named `fin` (with `newline$` as argument) so that BibTeX compiles correctly. The `newline$` command begins a new line.

The next step is to write the `.bib` file where one entry is sufficient to display the words `Hello World!`. Thus we will have the entry type `style1` defined in `first.bst`, a key named arbitrarily as `Elem1` and then one field named `toDisplay` taking as argument the words `Hello World!`. The listing of the `first.bib` file is:

```
@style1{Elem1,
  toDisplay = {Hello World!}
  }
```

In last we write the `first.tex` file. Omitting for brevity the compulsory `\documentclass` commands and the `document` environment, the listing of the `.tex` file is:

```
\nocite{Elem1}
\bibliographystyle{first}
\bibliography{first}
```

We should note two things here. Firstly, the `\nocite` command is used instead of the usual `\cite` for bibliography generation. The interested reader can try the latter command to see why `\nocite` is better. Secondly, since we have created a custom `.bst` file whose objective is not principally for bibliography generation, we choose the `bibliographystyle` and the `bibliography` to be the style file `first.bst`.

After running LaTeX for the first time, `.aux` and `.log` files are generated. Then, running BibTeX yields `.bbl` and `.blg` files, and finally a run with LaTeX twice again gives the `.dvi` file with the words `Hello World!` displayed. There is nonetheless one serious drawback to use BibTeX as we have been showing. Most of the time, the user will make a BibTeX related error in the `.bib` and `.bst` files. In this case, BibTeX will only complain of undefined citation(s). During compilation, many files are created and the TeXMAKER [2] IDE offers a tool to clean these compilation-time files except for the `.blg` file. This is a convenient feature that allows to isolate the `.tex`, `.bib` and `.bst` files quickly and hence investigate any mistake(s).

3 A Database Example

Table 1 shows a few entries of a tiny database which contains some information on the first three elements of the Periodic Table. We will see how BibTeX may be used to create a custom layout to present the data in Table 1 in a different way. Here unlike in our first example, we will first identify the fields from Table 1 and thus create the `.bib` file. There are five fields namely: Atomic Number, Element, Symbol, Type and Group. The second line from Table 1 concerning the element Helium for instance is converted to a `bib` entry as shown next.

Table 1. First Three Elements in the Periodic Table

Atomic #	**Element**	**Symbol**	**Type**	**Group**
1	Hydrogen	H	Nonmetal	I
2	Helium	He	Noble gas	VIII
3	Lithium	Li	Alkali metal	I

```
@style2{Elem2,
  atomicNumber = {2},
  name = {Helium},
  symbol = {He},
  type = {noble gas},
  group = {VIII},
  note = {A very light gas}
 }
```

We have taken the liberty to add a field named `note` in the `style2` function and it will become useful in Section 4.1. The next file to write is the `.bst` file and here we will concentrate on writing the `style2` function. The reader is referred to the available file `datab.bst` which contains this function. The `style2` function is given next:

```
1  FUNCTION{style2}{
2  " {\bf " write$ print atomicNumber write$ " }" write$%
3   print "{ \kern-2.6mm}" write$ ". " write$ name write$%
4   ", " write$ symbol write$ newline$ newline$
5  print "\\" write$ " It is classified as " write$ type write$%
6   " and is found in group " write$ group write$%
7   newline$ newline$
8  print "\\" write$ " {\sl " write$ "Note: " write$%
9   newline$ print note write$ " }" write$
10 print "\\" write$
11 print "\\" write$
```

In the `style2` function, the symbol % denotes that the successive line does not start a new line but instead continues just afterwards. In essence there are just five lines (with formatting commands) starting on line numbers 2, 5, 8, 10 and 11 respectively and they act as second argument to the `style2` function. More will be said on the formatting techniques used in Section 3.1. The final step is to write the `.tex` file. Here we have three entries corresponding to the three elements of Table 1 and as in the previous section, the entries can be outputted through `\nocite{Elem1}`, `\nocite{Elem2}` and `\nocite{Elem3}` respectively. The output for instance from a `.dvi` file, if a dvi viewer has been called, is shown in Fig. 1.

1. Hydrogen, H
It is classified as nonmetal and is found in group I
Note:

2. Helium, He
It is classified as noble gas and is found in group VIII
Note: A very light gas

3. Lithium, Li
It is classified as alkali metal and is found in group I
Note: Lithium salts are known to enhance moods

Figure 1. Customized layout 1 obtained with style2.

Two things are noteworthy here. Firstly, one is free to output the information that is desired. This can be achieved in two ways. Commenting the whole of line 8

of the `style2` function in the `.bst` file listing would prevent the `note` field from being outputted. Inserting the `%` symbol (normally after `write$`) in the `style2` function enables one to control which information is to be outputted. This is a huge convenience in getting desired information from a database. That said, the attention of the reader is drawn to the verbatim `style2` function listed where the `%` symbols have only been included so as the listing can fit within the text width of this article.

A second method is to extract only specific entries by choosing which of the three entries from Table 1 is to be outputted. This can be easily be done in two easy ways: Either the entry to be hidden is commented in the `.tex` file as `%\nocite{Elem3}` for the key entry of `Elem3` for instance (defined in the `.bib` file) or the entry in question is modified by removing the `@` symbol appearing in front of the key entry in the `.bib` file. This feature helps the user to pick out which entry that he or she needs. This issue will be re-visited with the final example in Section 4.

3.1 Formatting Techniques

In this subsection, we will focus a bit on the formatting used in the previous `.bst` file listing. Formatting with BibTeX commands bears resemblance to those of LaTeX but there are some new commands like `write$` and `newline$` which have to be inserted at the right places. The `print` function plays a central role here. It has the effect to get something outputted after compilation. For example, the sequence of instructions `print " It is classified as " write$` will output the line `It is classified as` and typing `print type write$` as on line 9 will output the `type` of the element as per Table 1.

If more complex formatting is desired as shown on line 2, the following sequence of instructions

```
" {\bf " write$ print atomicNumber write$ " }" write$
```

will write the `atomicNumber` in bold. Finally, the `newline$` command is a native BibTeX that functions as the `newline \n` command in C programming. However, `newline$` on its own might not suffice and it has been found that two `newline$` commands, one after the other, will force BibTeX to move to a new line. If an

empty line is to be inserted, then the sequence of instructions on line 10 will do the job.

3.2 Taking It A Bit Further

In the same `.bst` file (`datab.bst`), the user can define another function say `style3` that produces a different formatting from that of the `style2` function. Considering more elements in the Periodic Table, a user can choose between `style2` and `style3` the formatting style that he or she would like to apply to an individual element (and its associated information like Atomic Number etc.). Now it might be a good idea also to have the possibility to group the elements pertaining to one formatting style under a heading or title. But what if the information outputted spans more than a page? Then it will be convenient to have the relevant headings at the beginning of each page. This issue will be covered in the next section.

4 An Elaborate Example

This section aims to combine some various features of BibTeX commands so as to give the reader a bigger perspective of what can be done. We will still work with the Periodic Table. Since formatting with BibTeX can be trickier than LaTeX, it should be noted that the examples given only serve to introduce the capabilities of BibTeX and as such these examples only offer basic formatting techniques that do not necessarily comply with typographic conventions.

Essentially, two features will be demonstrated and the codes accompanying the discussions are available. Briefly the two features are as follows: Firstly, it will be shown how to add another function which will give a different formatting style to other entries. In addition the function will be created in a manner so that it will have the possibility to filter out a field (which is `note` in our case) when it is empty. Secondly, a different layout in the form of a table will be created and as in the first case, we shall see how we can take advantage of table properties from a LaTeX package.

4.1 Filtering Out An Irrelevant Field

Referring to Fig. 1 in Section 3, we see that unlike the elements Helium and Lithium, Hydrogen has an `note` field. Say that a user would not like BibTeX to output a blank `note` field as he or she considers it to be a waste of space. Besides, say the user would not like information contained within the field `group` for the element to be displayed. In this case, one has to write another function say `style3`. This new function can be included in the same `.bst` file (`datab.bst`) containing the function `style2` listed in Section 3. Also one might want to include headings in the output file so that the two styles `style2` and `style3` can be distinguished from each other. To our rescue, there is the well-established `longtable` [3] package.

We will start with putting headings in our style files. The `longtable` [3] package offers a convenient feature to have a table spanning the full width of a page via the commands `\setlength\LTleft{0pt}` and `\setlength\LTright{0pt}`. A function arbitrarily named `createsection` containing mostly commands from the `longtable` [3] package permits headings to be created and classification (and demarcation) of entries (refer to the `datab.bib` file attached). One can further make use of commands like `\endfirsthead` and `\endhead` for better structure.

The next thing that we shall investigate is how to display the `note` field only when this field in the `.bib` file is not empty. To this end, we shall make use of the BibTeX commands `if$` and `empty$`. The `if$` internal command, like in programming languages, allows decisions subject to some defined conditions while the `empty$` command elegantly fits our specific need here as in combination with an `if$` command, it allows an output to be written or not. The output from the `style3` function is shown in Fig. 2 and the reader is invited to compare this figure to Fig. 1 from Section 3. The reader is also directed to [1] for deeper insights.

Headings do not accompany Fig. 2 in order to save space but the user is reminded again that the `createsection` function is responsible for the headings and also commands like the key environment `%\nocite{begcla1}` and `%\nocite{endcla1}` are required (refer to the file `datab.tex`).

1. Hydrogen, H
It is classified as nonmetal

2. Helium, He
It is classified as noble gas
Note: A very light gas

3. Lithium, Li
It is classified as alkali metal
Note: Lithium salts are known to enhance moods

Figure 2. Customized layout 2 obtained with style3.

4.2 Yet Another Formatting Style

The objective here is to create a tabular structure like Table 1 but this time displaying only three fields: `Atomic Number`, `Element` and `Symbol`. The function is named `style4` and by making use of the `longtable` [3], running headings can be easily made for example when a table continues to a new page. The function `tablesection` given in the attached `datab.bst` file handles the headings in this case. It contains only some basic formatting and a user can easily bring modifications.

5 Some Useful Information

The user is referred to the `datab.bib` and `datab.tex` files where entry types (`style2, style 3 and style4`) or styles are grouped under argument key environment such as `begcla2` and `endcla2`. The user is invited to mix styles under a given argument key environment.

Nevertheless, BibTeX is stubborn in that it will not allow a key (referring here to elements `Elem1`, `Elem2` etc. as in the `datab.bib` file) to be repeated. A simple workaround would be to give the duplicate key a different name (as under the `tablesection` function in the `datab.bib` file) and to adjust the corresponding entries in the `.tex` file accordingly.

6 Conclusion

By writing this article, I hope that more LaTeX (and TeX) users will be encouraged to consider digging into BibTeX commands. BibTeX is a well-suited tool to handle static database information. The user is expected to realize that almost any information which can be treated as a database can benefit from the formatting techniques shown.

Readers conversant with the Extensible Markup Language (XML) will find that BibTeX perhaps is not as flexible in terms of simplicity and capabilities. But let us remind ourselves of two important things: Firstly, BibTeX and XML were created for different purposes, and secondly BibTeX has been here before XML came to life. Hence, any comparison is unfounded and unfair.

That said, this gap can lessened if BibTeX is combined with JabRef [4] to unveil similar XML capabilities like filtering, sorting and information retrieval [5]. JabRef [4] is a very flexible tool where the user can create custom fields and maybe through custom filters, it can produce better results than given in this article. I will not venture in this alley as tweaking deep with JabRef [4] remains a mystery for me.

Nevertheless, even without JabRef [4], smart constructs can be made with BibTeX as we have seen in Section 4 with commands like `if$` and `empty$`. With the sky being the limit for LaTeX, TeX fans, who knows how BibTeX will spice up our world in the future?

Disclaimer

The codes available with this article are named `datab.tex`, `datab.bib` and `datab.bst` respectively. Compilation is achieved through running LaTeX BibTeX LaTeX LaTeX sequentially. No guarantee is made for the accuracy of the information contained in the database nor the codes provided are warranted to be fit for any particular application.

References

[1] Nicolas Markey. Tame the Beast: The B to X of BibTeX, Version 1.3, October 2005.

[2] TeXMAKER 1.2.1. 2004-2005, http://www.xm1math.net/texmaker/.

[3] David Carlisle. The longtable Package, CTAN: http://www.ctan.org/tex-archive/help/catalogue/entries/longtable.html.

[4] JabRef Reference Manager 2.3.1. 2008, http://jabref.sourceforge.net/.

[5] Michael J. Young. *Formation Á XML*. Microsoft Press, 2000, Translated to French from English: James Guerin.