Ebib: a BIBTEX database manager for Emacs

Joost Kremers
joostkremers@yahoo.com

August 3, 2004

Contents

Ebib is a program with which you can manage a BIBTEX database without having to edit the raw `.bib` file. It runs in Emacs, version 21.1 or higher. Lower versions are not supported.

It should be noted that Ebib is *not* a minor or major mode for editing BIBTEX files. It is a program in itself, which just happens to make use of Emacs as a working environment, in the same way that for example Gnus is.

The advantage of having a BIBTEX database manager inside Emacs is that X is no longer required, as Emacs can run on the console, and also that some integration with Emacs' TEX and LATEX modes becomes possible. For example, one can insert a key from the database into the text one is editing using tab completion. Another advantage of Ebib is that it is completely controlled by one-key commands: no stressful mouse movements are required, as with most other (usually X-based) BIBTEX database managers.

Note for the impatient: pressing 'h' in Ebib will give you an overview of the commands you can use in the buffer you are in. You should be able to get by with just reading the Overview (section 1) and using this help function. Though reading the entire manual is still a good idea, of course.

1 Overview

A BIBTEX database is somewhat of a free-form database. A BIBTEX entry consists of a set of field-value pairs. Furthermore, each entry is known by a unique key. The way that Ebib navigates this database is by having two windows, one that contains a list of all the keys in the database, and one that contains the fields and values of the currently highlighted entry.

When Ebib is invoked (with the command `M-x ebib`), the current windows in Emacs are hidden and the screen is divided into two windows. The top one contains a buffer that is called the *keys buffer*, while the lower window contains the *fields buffer*. When a database is loaded, the keys buffer holds a list of all the keys in the database. You can navigate through these keys with the cursor keys or with 'j' and 'k'. In the fields buffer, the fields of the currently highlighted buffer are show, with their values.

Field types In the fields buffer, the entry type is given at the top, followed by (normally) three groups of fields. The first group are the so-called required fields, the fields that BIBTEX requires to be filled. The second group are the optional fields, which do not have to be filled but which BIBTEX will normally add to the bibliography if they do have a value. The third group are the so-called ignored fields. These fields are usually ignored by BIBTEX (note that BIBTEX will normally ignore *all* fields it does not know), although there are bibliography styles that treat some of these fields as optional rather than as ignored; (i.e., the *harvard* styles do typeset the `url` field, if present.)

As you will probably know, the obligatory and the optional fields can be different for each entry type. The ignored fields, however, are common to all the entry types. You can use these, for example, to add personal comments to the works in your database. (To see how you can define the entry types and the fields for them, see section 5.)

Basic editing When Ebib is first started, both buffers are empty. You can open a database with 'o', or start a new one from scratch simply by adding entries, using the key 'a'. For a full description of all the available key commands in the keys buffer, see section 3. A special key is the key 'e', which allows you to edit the current entry. When you hit 'e' on a key, focus moves to the fields buffer, with the first field (the `type` field) highlighted. You can move up and down the fields with the cursor keys or with 'j' and 'k'. There are commands for cutting, copying and pasting the field contents, and of course one for editing the value of the current field. See section 3 for a full description.

Field values The first field of each entry is special: this is the `type` field, and it holds the type of the entry (i.e., whether it is a `book, article, phdthesis,` etc.)[1] When you create a new entry, it is advisable to edit this field first and set it to the correct value, because the entry type determines the fields that are available. (Note, however, that it is really not a problem to change the type of an entry when some of its fields already have values. If the old entry type had fields that do not exist in the new entry type, their values will be retained and saved, although they are not shown. If you should later change the entry type back, those fields will become visible again.)

All the other fields of an entry have simple text values. You can enter a value for a field, or change the existing value, by hitting 'e'. BIBTEX requires that a field value is surrounded by either braces or quotes, but you do not have to type these, as Ebib will add them automatically when it saves the database.[2] Note that if you want extra parenthesis in or around the field contents (e.g., to keep BIBTEX

[1] Note that in the BibTeX file, this is not really a field at all, but the keyword that follows the at-sign opening a new entry.

[2] More precisely, Ebib stores the field values in memory with braces around them. It removes those braces if the field value is printed on the screen.

from downcasing words in a titel, or to make sure it handles accented letters correctly) you can of course type them. Ebib puts braces around the values that you enter, but does not change them in any other way.

Multiline values Normally, a field value consists of only one line of text. However, BibTeX does allow a field value to have newlines, i.e., a field value can consist of more than one line. This can be useful for example in annotated bibliographies, if you want to add a field for personal comments about the cited works, or if you want to add an abstract to the entry. In Ebib, a field value that contains newlines is called a *multiline* value, and Ebib has a special way of handling them.

When a field holds a multiline value, only its first line will be displayed in the fields buffer. A plus sign '+' will appear in front of the field's value, however, to indicate that the value is longer than what is shown. You can see the whole of the value by highlighting the field and hitting 'l'. The field value will then be displayed in the bottom window (replacing the fields buffer) in what is called the *multiline edit buffer* (see section 3). Here you can see and edit the field's value. If you want to give a field a multiline value that does not have one yet (either because it is empty, or because it contains a single-line value), you can also use 'l'.

To leave the multiline edit buffer and store the text you have entered, hit `C-x C-s` or `C-x b`. To leave the buffer without saving your changes, hit `C-x k`.

Raw values As just mentioned, Ebib will put braces around the field values that you enter. This is not always desirable, however. When a field contains an abbreviation defined with an `@string` command, it must not be enclosed in braces, because then BibTeX would ignore the abbreviation and treat it as normal text. In order to tell Ebib that a field must not be surrounded by braces, you must mark the field as *raw*. This is done with the key 'r'. An asterisk '`*`' will appear before the field's value in the fields buffer. When Ebib saves the database, raw fields will be saved *as is*, without additional braces.

Note that this also makes it possible to enter field values that are composed of concatenations of strings and abbreviations. The BibTeX documentation for example explains that if you have defined:

```
@string{WGA = "World Gnus Almanac"}
```

you can create a BibTeX field like this:

```
title = 1966 # WGA
```

which will produce "1966 World Gnus Almanac". Or you can do:

```
month = "1~" # jan
```

which will produce someting like "1 January", assuming your bibliography style has defined the abbreviation `jan`. All this is possible with Ebib, simply by entering the exact text including quotes or braces around the strings, and marking the relevant field as raw.

Note, by the way, that a field value can be both raw and multiline at the same time. Such a value will be marked with '`*+`' in the field buffer.

`@String` definitions There would be no point in having raw fields if there were no way to create `@string` abbreviations. So obviously, Ebib provides one. When you hit 'S' in the keys buffer, the fields buffer will disappear and be replaced by the *strings buffer*. This buffer shows all the `@string` definitions in the database. You can move through them in the same manner as you navigate the fields of an entry, and you can edit them, delete them and add new ones. BibTeX requires that string values be surrounded by braces or quotes, but just like with the field values, Ebib takes care of this: you do not have to type them yourself.

Note that the value for an `@string` definition cannot be raw, simply because it would not make any sense, and BibTeX would not accept it. An `@string` can be multiline, however. A multiline string is created with 'l', just as a multiline field. There is one limitation, however: if you define a new string, you are always asked to enter a single-line value. You can only make it multiline after it has been defined.

`@Preamble` definition Ebib allows you to add one `@preamble` definition to the database. In principle, BibTeX allows more than one such definition, but, as explained in the BibTeX documentation, one should suffice, because you can use the concatenation character # to include multiple TeX or LaTeX commands. So, you can write this in your `.bib` file:

```
@preamble{ "\newcommand{\noopsort}[1]{} "
         # "\newcommand{\singleletter}[1]{#1} " }
```

With Ebib, of course, you only have to type the text between the braces. Ebib will take care of including the braces of the `@preamble` command, but otherwise it will save the text exactly as you enter it. This means for example that you have to add the quotes around the string or strings in the `@preamble` command yourself.

When Ebib loads a `.bib` file that contains more than one `@preamble` definition, it will concatenate all the strings in them in the manner just described and save them in one `@preamble` definition,

Closing Ebib There are two ways of leaving Ebib. The first is to use the key 'z'. All this does is put Ebib in the background by hiding the Ebib buffers: the database remains loaded, and you can go back to Ebib at any time with the command `M-x ebib`. The second way to leave Ebib is the key 'q'. This quits Ebib completely, and unloads the database. You can of course restart Ebib, but it will restart with an empty database. Note, by the way, that both these commands are only available in the keys buffer.

When Ebib is in the foreground, it requires that both its windows are present, and for proper operation it is also necessary that none of its buffers are closed explicitly by the user. For this reason, the key sequences `C-x b` and `C-x k` are redefined inside the Ebib buffers. In the fields buffer, they are set to nil (i.e., they do nothing), while in the keys buffer, `C-x b` is equivalent to 'z', which lowers emacs, while `C-x k` is equivalent to 'q', which quits Ebib.

This, by the way, will not stop you from killing an Ebib buffer explicitly with `M-x kill-buffer`. It will stop you from *accidentally* killing an Ebib buffer, though.

If you should kill Emacs with `C-x C-c` while the database in Ebib was not saved yet, you will be asked if you want to save it. If you answer 'y', Ebib will save the database (possibly asking for a file name if none was specified yet) and Emacs will close. If you answer 'n', you will be asked if you really want to kill Emacs and abandon the changes you made to the database.

LaTeX integration Leaving Ebib with the command 'z' has an additional advantage, apart from the fact that you do not have to reload your database when you invoke Ebib again. When you are in a non-Ebib buffer and call the command `M-x ebib-insert-bibtex-key`, Emacs will prompt you for a key from the database currently loaded in Ebib. You can use tab-completion at this prompt, to complete a partial key, or to show you all the possible completions. After hitting ENTER, Emacs will put the selected key at the cursor position in the current buffer, surrounded by braces.

The easiest way to use this function is of course to bind it to a key sequence. You can do this with `global-set-key`, but it makes more sense to define a key sequence in your LaTeX mode(s) only. For example, if you use AucTeX, the following command in `~/.emacs` will bind the key sequence `C-c b` to `ebib-insert-bibtex-key` in AucTeX's `LaTeX-mode`:

```
(add-hook 'LaTeX-mode-hook #'(lambda ()
                  (local-set-key "\C-cb" 'ebib-insert-bibtex-key)))
```

Obviously, you can choose any other key or key sequence. Under a standard set-up, however, `C-c b` should still be available.[3]

2 Installation

To install Ebib, so that it will be loaded automatically when Emacs is started, simply copy the file `ebib.el` to somewhere in your load path and add the following line to your `~/.emacs`:[4]

```
(autoload 'ebib "ebib" "Ebib, a BibTeX database manager." t)
```

Alternatively, you can put `ebib.el` anywhere you like and load the file explicitly, e.g.:

```
(load "~/programs/emacs/ebib")
```

When Ebib is loaded, you can run it with `M-x ebib`. This command is also used to return to Ebib when you have put the program in the background. You can, of course, bind this command to a key sequence by putting something like the following in your `~/.emacs`:

```
(global-set-key "\C-ce" 'ebib)
```

You can also byte-compile the source, either within Emacs with `M-x byte-compile-file`, or from your shell by going into the directory where you put `ebib.el` and typing:

```
emacs -batch -f batch-byte-compile ebib.el
```

This will create a file `ebib.elc`, which emacs will load instead of `ebib.el`. It will make Ebib run a bit faster.

In order to run, Ebib needs the file `~/.ebibrc`. This file contains essential definitions, and Ebib cannot and will not run without it. A sample `ebibrc` is provided, which you can copy to your home dir with the following command:

```
cp /path/to/ebibrc ~/.ebibrc
```

The sample `ebibrc` should be sufficient for most users, but if you want, you can customise it and define your own entry types. See section 5 for details.

[3] Note that in Emacs key sequences of `C-c <letter>` are reserved for the user, so no package should have bound them to any function. For the same reason, Ebib does not set this key automatically.

[4] If you do not know what your load path is set to, go to the `*scratch*` buffer, type `load-path` on an empty line, put the cursor right after it and type `C-j`. The value of `load-path` will then appear in the buffer.

3 The Ebib buffers

In the following sections, the various buffers that Ebib uses are discussed, and all the available key commands are listed. Note that in the keys, fields and strings buffer, you can get a quick overview of the available commands by pressing 'h'.

The keys buffer

The keys buffer contains the keys of all the entries in the database. It is the buffer that is active when Ebib is invoked. The following key commands are available:[5]

j, Down – go to next entry.

k, Up – go to previous entry.

g, Home – go to first entry.

G, End – go to last entry.

b, PgUp – move ten entries up.

Space, PgDn – move ten entries down.

q, C-x k – quit Ebib. This sets all variables to nil, unloads the database and quits Ebib. When Ebib is restarted with `M-x ebib`, it will start with an empty database. You will be prompted if you really want to quit, and see a warning if the database was modified. Note that in the keys buffer, `C-x k` does not behave in the normal way: it quits Ebib, rather than just killing the current buffer.

s – save the database. This saves the database to the file from which it was loaded. If the database was started from scratch, you will be prompted for a file name. Ebib will make a backup of the `.bib` file (named `<filename>.bib~`) if it is being saved for the first time after opening it.

w – write the database to a different file. This also saves the database, but to a different file than the one from which it was loaded. You are prompted for a file name. This file name will be the new file name, so if you use 's' after having used 'w', the file will be written to the new file, not to the one from which it was originally loaded.

c – close the database. This unloads the current database, but does not quit Ebib. You will be prompted if you really want to close it, and receive a warning if the database was modified.

o – open a `.bib` file. This prompts you for a file name to load. Note that you can only open a `.bib` file when the database is empty.

m – merge a `.bib` file. If you already have a database loaded (or created one from scratch) you can use this command to read BibTeX entries (and of course `@string` and `@preamble` definitions) from another file and add them to the database. If the file being merged contains entry keys or `@string` definitions that already occur in the database, they will be ignored and you will receive a warning message.

[5] Note that here and everywhere, command keys are case-sensitive; i.e., 'g' and 'G' run different commands.

a – add an entry. You are prompted for a new entry key. If you enter a key that already exists, the action is aborted with an appropriate error message. Otherwise, the new key is inserted in the keys buffer and made active. Focus then moves to the fields buffer, where you can edit the fields. (See section 'The fields buffer' below.)

/ – search the database. With this command, you can search through the entire database. You are prompted to enter a search string (which can be a regular expression). Searching will start at the current entry, not at the first entry of the database. If the search string is found, the corresponding entry is displayed, and the search string is highlighted everywhere it is found in the entry. If the search string was found in a multiline value, the plus sign before the field value is highlighted. Keep an eye out for this, because if the search string is not found in the first line of the multiline value, it will not be shown on screen, and only the highlighted plus sign indicates where the search string is found.

n – find next occurrence of the search string. Searching will be resumed from the entry following the current entry, and the next occurrence of the search string will be shown. Note that the search does not wrap: if the last entry of the database is reached, the search will not continue at the first entry. Hit 'g' and then 'n' to continue searching from the top.

f – writes the file name of the current database in the minibuffer. This may be useful sometimes.

d – delete the current entry. You are prompted for confirmation. Note that once an entry is deleted, it cannot be retrieved anymore.

x – export the current entry to a file. With this, you can write a single entry to another file. The entry will be appended to the file if it already exists, otherwise the file will be created. This command is intended for creating a new BIBTEX file out of entries of an existing one, for example if you want to submit a paper with a `.bib` file containing only the relevant entries. The file name to which you save an entry is remembered, so the next time you use 'x' it will default to the same file, although you can of course change it. Note: you may want to export the preamble and `@string` definitions as well. See below (key 'X') and section 'The strings buffer' (keys 'x' and 'X').

Return – make the entry under the cursor current. In the keys buffer, you can use the standard search functions (e.g., `C-s`) to search for an entry key. When one is found, the cursor is moved to it. After you exit the search function, you can hit RETURN to force Ebib to make that entry the current entry. (With `C-s`, this essentially comes down to having to hit RETURN twice when you found the desired key.)

z, C-x b – lower Ebib. This puts Ebib in the background and restores the windows that were in the Emacs frame before you started Ebib. Ebib will stay active, and your database will remain loaded. You can return to Ebib with `M-x ebib`. Note that in the keys buffer, `C-x b` has been redefined: it does not switch from the current buffer in the normal way.

e – edit the current entry. This puts you in edit mode, where you can edit the fields of the current entry. See section 'The fields buffer' below.

S – list the `@string` definitions in the database. You will be put in the strings buffer, where you can edit or delete `@string` definitions, or add new ones. See section 'The strings buffer'.

P – show the `@preamble` definition in the database. You will be put in multiline edit mode (see section 'The multiline edit buffer') where you can edit the text for the `@preamble` command in the database. Note that Ebib only allows you to define one `@preamble` definition, which will be put at the top of the database. This should not be a problem, because you can use BibTeX's concatenation character # to put multiple LaTeX commands in the preamble. The text that you enter is put in the `@preamble` command *literally*, so you have to type quotes around your strings.

X – export the `@preamble` definition to a file. If you want to create a new `.bib` file by exporting entries from the database, you may also want to export the preamble.

The fields buffer

When you hit 'e' or 'a' in the keys buffer, you will be placed in edit mode: focus moves to the fields buffer, and you can edit the fields of the current entry. The current field is highlighted. The following key commands are available in the fields buffer:

j, Down – go to the next field.

k, Up – go to the previous field.

g, Home – go to the first field.

G, End – go to the last field.

b, PgUp – go to the next set of fields. That is, when you are on one of the required fields, go to the first optional field, etc.

Space, PgUp – go to the previous set of fields. That is, when you are on one of the the ignored fields, go to the last optional field, etc.

q – quit editing the current entry and return focus to the keys buffer.

c – copy the contents of the current field to the kill ring. You can copy the field contents and use `C-y` in a non-Ebib buffer to yank it, or use 'p' to yank it to another (empty) field, possibly in another entry.

x – cut the contents of the current field. Like 'c', 'x' puts the contents of the current field in the kill ring.

p – paste the last element in the kill ring to the current field. It is not (yet) possible to move around the kill ring with this command. (You can of course use 'e' and then `C-y/M-y` to do this.) Note that no text will be pasted in the field already has a value. Use 'd' first to delete the fields value.

d – delete the value of the current field. The deleted contents will *not* be put in the kill ring, and is therefore irretrievably lost.

e – edit the current field. You are placed in the minibuffer where you can fill in a value for the current field, or edit the existing value. `C-g` cancels the edit action and retains the original value, if any. RETURN confirms the edit. If you edit the entry type, you must select a type from the ones

defined in the init file. (See section 5.) You can use tab-completion in this case. If you edit the `crossref` field, you must select a key from the database. Here, too, tab-completion works. If you edit a field that has a multiline value, you will be put in the multiline edit buffer (see section 'The multiline edit buffer').

r – toggle a field's "raw" status. The current field is marked as raw (an asterisk appears) or it becomes a "normal" field again when it was already marked as raw. If you hit 'r' on a field that has no value yet, the field is marked as raw and you are prompted for a value. This is an important option if you use `@string` definitions.

l – edit the current field as multiline. This puts you in Ebib's multiline edit buffer (see section 'The multiline edit buffer'), which allows you to give the current field a multiline value (i.e., a value that contains newlines). If you want to edit a field that already has a multiline value, you can also use 'e', but you must use 'l' if you want to enter a multiline value for a field that doesn't have one yet. Note that in the fields buffer, only the first line of the field's value is shown. A plus sign '+' is displayed in front of it to indicate that the value is actually longer.

s – insert an abbreviation from the `@string` definitions in the database. This is another way to edit a field value, with the restriction that you can only select one of the abbreviations in the database. Tab completion works, and the field will automatically be marked raw. Of course you do not *have* to use this command; it is also possible to enter an abbreviation with 'e' and 'r'. But it is a good way of ensuring that you do not mistype an abbreviation, and that the field is marked raw. Note that 's' does not work if the field already has a value.

C-x b, C-x k – these keys have been disabled in the fields buffer, because it would confuse Ebib if one of its buffers were inadvertently killed or put in the background. To leave Ebib, first hit 'q' to return to the keys buffer, and then either 'z' or 'q'.

The strings buffer

When you hit 'S' in the keys buffer, you will be put in the strings buffer, where you can edit the `@string` definitions in your database. The keys available here are partly the same as the keys in the fields buffer:

j, Down – go to the next string.

k, Up – go to the previous string.

g, Home – go to the first string.

G, End – go to the last string.

b, PgUp – go ten strings up.

Space, PgDn – go ten strings down.

q – quit the strings buffer and return focus to the keys buffer. Note that the strings buffer will disappear from the bottom window and the fields buffer is displayed again.

c – copy the text of the current string to the kill ring. You can copy the string value and use `C-y` in a non-Ebib buffer to yank it.

e – edit the value of the current string. You are placed in the minibuffer where you can edit the value. `C-g` cancels the edit action and retains the original value. Return confirms the edit. It is pointless to have an empty value for a string, so Ebib will not accept an empty string. If you edit a string value that is multiline, you will be placed in the multiline edit buffer.

l – edit the value of the current string as multiline. You are placed in the multiline edit buffer (see section 'The multiline edit buffer').

a – add a new `@string` definition. You will be prompted for a new abbreviation, which has to be unique, and for a string value.

d – delete the current `@string` definition from the database. You will be asked for confirmation.

x – export the current `@string` definition to a file. Use this in conjunction with the commands to export fields and the preamble.

X – export all the `@string` definitions to a file. Useful if you do not want to hit 'x' on every abbreviation separately.

The multiline edit buffer

Ebib has a special multiline edit buffer, which is used to edit field values that contain newlines (so-called *multiline fields*), and also to edit the contents of the `@preamble` command. Ebib enters multiline edit mode in one of three cases: when you press 'P' in the keys buffer, to edit the `@preamble` definition, when you hit 'l' in the fields or strings buffer, to edit the current field or string as multiline, or when you hit 'e' on a field or string whose value already is multiline.

The multiline edit buffer uses a special major mode, `ebib-multiline-edit-mode`, which is derived from `text-mode`. The changes with respect to `text-mode` are minor (see below), which means that any customisations you may have made to `text-mode` will be available in the multiline edit buffer.

The settings that are specific for `ebib-multiline-edit-mode` are the functions assigned to the key sequences `C-x b`, `C-x C-s` and `C-x k`. Either of these three key sequences can be used to leave the multiline edit buffer. The first two, `C-x b` and `C-x C-s` will store the text to the field, string or preamble, whichever you were editing. `C-x k` on the other hand discards your changes and retains the original value.

If you leave the mulitiline edit buffer with `C-x C-s` or `C-x b` and the buffer is empty (i.e., you deleted all the text, including the final newline), and you were editing a field value or the `@preamble` definition, the field value or preamble will be deleted.[6] If you do this while editing an `@string` definition, you will get an error message, because `@string` definitions cannot be empty.

4 Format of the `.bib` file

In principle, Ebib accepts everything in the `.bib` files it reads that BibTeX accepts. There are a few exceptions, however, that one may need to be aware of when one loads a self-created `.bib` file into Ebib. Like BibTeX, Ebib accepts both upper and lower case for the entry types and field names: they

[6] This is in fact the *only* way to delete the `@preamble` definition. Field values on the other hand, can also be deleted by hitting 'x' or 'd' on them in the fields buffer.

are not case-sensitive. Unlike BibTeX, however, the entry keys *are* case-sensitive. The reason for this is that they are so for LaTeX.

In entry types, field names, entry keys and `@string` abbreviations, Ebib will accept all printable ASCII characters except the following:[7]

```
" # % ' ( ) , = { }
```

Ebib does not handle `@comment` commands. Ebib will ignore them when loading a file, and if you then modify the file and save it, the `@comment` commands will be gone.

Note also that when Ebib finds a field that has not been defined in the init file (see section 5), it will store it, and also save it to the `.bib` file, but it will not display it. No warning is given when this happens, so if you load your self-made `.bib` file into Ebib, make sure all the fields you use are properly defined.

When Ebib encounters an entry type that it does not know, it will issue a warning message, and continue searching for the next entry. Note that if there is more than one unknown entry in the `.bib` file, only the last warning message will be visible after the file has been loaded. To see all warning messages, you have to switch to the `*Messages*` buffer.

5 The initialisation file

Ebib makes use of an init file `~/.ebibrc`. This file is loaded when Ebib is first loaded, and contains the definitions of the entry types and the field types. This information is essential to Ebib, because it cannot set up the database without it. Therefore, if Ebib cannot find this file, it will not run. For convenience, a default `ebibrc` file is provided. For most people, it will be sufficient to simply copy this file to their home directory.

For those that need or want to customise the entry types, this section explains how the init file is built up. It's basic Lisp syntax, so it shouldn't be too confusing. The init file consists of a set of entry type definitions that look like the following:

```
(defentry article                       ; name of entry type
  (author title journal year)           ; obligatory fields
  (volume number pages month note))     ; optional fields
```

This defines the entry type `article` with the obligatory fields `author`, `title`, `journal` and `year`, and with the optional fields `volume`, `number`, `pages`, `month` and `note`. Although Ebib indicates on the screen which fields are obligatory and which are optional, it does this purely for convenience: Ebib won't complain if you leave an obligatory field empty. It is BibTeX that will complain.

Note the format of the entry definition above: it starts with a parenthesis, then the keyword `defentry`, followed by the entry name. Then follows a list of obligatory fields, enclosed by parentheses. Next follows a list of optional fields, also enclosed by parenthesis. Finally, a closing parenthe-

[7] Actually, I found the BibTeX documentation somewhat lacking on this point. The characters listed here are not allowed in `@string` abbreviations, but I am not entirely sure if the same counts for the entry types, field names and key. The keys must probably adhere to LaTeX's rules for labels etc., but again I have found no documentation explaining what characters can appear in those. If you should know what exactly is and is not allowed, please let me know. If you do not, better err on the side of caution and use only letters, numbers, colon and dash in your keys, and only letters in the entry types and field names.

sis follows, which closes the first parenthesis before `defentry`. If you want to add any comments, you can use a semicolon. All the text after a semicolon up to the end of the line is ignored.

Note that it is legal to leave either the list of obligatory or optional fields empty. Such an entry type simply does not have fields of that particular type. When you do this, put an empty set of parenthesis:

```
(defentry misc
  ()
  (title author howpublished month year note))
```

Next to the obligatory and optional fields, it is also possible to define a set of ignored fields. These are fields that BibTeX will normally ignore, but they can be useful for storing additional information. These fields are identical for all entry types, and therefore they do not need to be defined for each entry type separately. Instead, they are defined with a line such as the following:

```
(def-ign-fields crossref url annote abstract keywords)
```

This defines `crossref`, `url`, `annote`, etc. as ignored fields. They will be made available for all entry types, but they do not have to be filled in. Note again that the entire definition must be enclosed within parentheses. If `~/.ebibrc` contains more than one `def-ign-fields` definitions, only the last one is used.

One thing to keep in mind if you edit the `.ebibrc` file is that *everything* must be typed in lower case. Do not use upper case, because Ebib will ignore everything that is written in upper case.[8]

6 Limitations

Obviously, Ebib is not perfect. Some of its more serious limitations are listed here.

- It is not possible to add `@comment` commands to the BibTeX file.
- You can only work on one `.bib` file at a time. It is not possible to open another `.bib` when one is already open.
- Backward searching is not possible.
- There is no customisation beyond what you can change in the source.

[8] To be more precise, when reading a `.bib` file, Ebib converts entry types and field names to lower case. The definitions in `.ebibrc` are not converted to lower case, however. As a result, entry types or field names defined as upper case will never match an entry type or field name read from a `.bib` file. Note, by the way, that writing `defentry` or `def-ign-fields` in upper case will result in an error from Emacs.