Ebib

Joost Kremers

29 Sep 2012

Ebib is a program with which you can manage BibTeX database files without having to edit the raw `.bib` files. It runs in GNU/Emacs, version 23.1 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, you can push a BibTeX key from Ebib to a LaTeX buffer, or, vice versa, when you're in a LaTeX buffer, you can consult your BibTeX database and insert a key from it into the document. Another advantage of Ebib is that it is completely controlled by key commands: no stressful mouse movements are required, as with most other (usually X-based) BibTeX database managers.

Installation

Package manager

The easiest way to install Ebib is to use Emacs' package manager. Ebib is available as a package from the Melpa package archive. If you add the Melpa archive to your `package-archive` list, you can install Ebib from the package manager. This will also install the Info file so you can access the Ebib manual within Emacs.

Note: the package manager (`package.el`) is part of Emacs 24. Users of Emacs 23 have to install it first. See Wikemacs for discussion.

Manual installation

You can also install Ebib manually by copying the file `ebib.el` to somewhere in your load path and add the following line to Emacs' init file `~/.emacs`:

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

On a default installation, the load path probably only contains system directories. If you want to run `ebib.el` from somewhere else (e.g., directly from the source directory), you can add this directory to your load path:

```
(add-to-list 'load-path "~/src/ebib"))
```

Note: 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.

Alternatively, you can load Ebib directly by putting the following in `.emacs`:

```
(load "/path/to/ebib")
```

The difference between `load` and `autoload` is that the former loads Ebib immediately and thus slows down Emacs' startup (a bit), while the latter loads Ebib only when it is called (and thus slows down Ebib's first time start up a bit).

It is recommended to byte-compile the source, Ebib runs quite a lot faster when it is byte-compiled. You can do this either within Emacs with `M-x byte-compile-file` (recommended on OS X), 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`

Installing the info file

If you install Ebib through the package manager, the Info file is installed automatically as well. However, if you install Ebib manually, or run it from the source directory, this is not the case. If you want the Info file to be accessible from within Emacs, you need to put it in a directory where Emacs can find it. One way to do this is to put `ebib.info` in one of the standard info directories. Make sure to run `install-info ebib.info dir`, so that Emacs knows about the new info file.

Alternatively, you can leave `ebib.info` in Ebib's source directory and add that dir to `Info-additional-directory-list`. It contains a suitable `dir` file, so there's no need to run `install-info`.

Starting Ebib

Once Ebib has been installed and is loaded, you can start 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 bind this command to a key sequence by putting something like the following in Emacs' init file:

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

You can of course choose any key combination you like. (In Emacs, key combinations of `C-c <letter>` are reserved for the user, so that no package may set them.)

Basic Usage

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 entry keys in the database, and one that contains the fields and values of the currently highlighted entry.

When Ebib is started, the current windows in Emacs are hidden and the Emacs frame is divided into two windows. The top one contains a buffer that is called the *index buffer*, while the lower window contains the *entry buffer*. When a database is loaded, the index buffer holds a list of all the keys in the database. You can move through these keys with the cursor keys. In the entry buffer, the fields of the currently highlighted entry are shown, with their values.

In this chapter, all basic functions of Ebib are described, so that you can get startet with it. At times, reference will be made to later chapters, where more specific functions are described.

Ebib has a menu through which most functions can be accessed. Especially some of the lesser used functions can only be accessed through the menu (unless you assign key shortcuts to them, of course.)

Getting Started

Ebib is started with the command `M-x ebib`. Entering this command hides all the windows in the current Emacs frame and replaces them with two windows: the top one contains the index buffer, the bottom one, taking up the larger part of the screen, contains the entry buffer. The index buffer is named `none`, to indicate that no database has been loaded. If you open a database, or start a new one, the index buffer will carry its name.

You can quit Ebib by typing `q`. You will be asked for confirmation, and you will receive a warning if you happen to have an unsaved database. The command `z` can also be used to leave Ebib. However, unlike `q`, which completely quits Ebib, `z` only lowers it, so that it remains active in the background. The `.bib` files that you have opened remain loaded, and you can return to them by typing `M-x ebib` again.

Opening a `.bib` File

Loading a `.bib` file into Ebib is done with the command `o`. Ebib reads the file that you specify, and reports how many entries it found, how many `@string` definitions it found, and whether a `@preamble` was found. Note that when Ebib reads a `.bib` file, it only reads entry types (e.g. `book, article, phdthesis` etc.) that it knows about. Fields (e.g. `author, title, year` etc.) that Ebib does not know about, are loaded (and saved) but not displayed, so they cannot be edited. Therefore, you should make sure that all the entry types and fields that your databases use are defined. A sensible set has been predefined, so that anyone who's using standard BibTeX entry types should have no problem loading an existing `.bib` file into Ebib. If, however, you have custom entry types, or custom fields in your `.bib` files, you should define them. This can be done by selecting "Options | Modify Entry Types" from the Ebib menu. (See also the chapter on customising Ebib, esp. Entry Types.)

Every time Ebib reads a `.bib` file, it produces a few log messages. These are written into a special buffer `*Ebib-log*`. If Ebib encounters entry types in the `.bib` file that it doesn't know, it will log a warning. If Ebib finds something that it believes to be incorrect, an error will be logged. If any warnings or errors occur while loading the `.bib` file, Ebib tells you so after loading the file. To view the log file, press `l` in the index buffer.

Note that even if it detects warnings or errors, Ebib will try to continue parsing the rest of the `.bib` file. That means that normally, only the entry in which an error occurs is not read. Entries occurring after the problematic one are read.

Navigating a `.bib` File

Once you've opened a `.bib` file, the keys of all the entries in the file are shown in alphabetical order in the index buffer in the top Ebib window. (In fact, it is possible to show more than just the entry key in this buffer. See Index Display Fields on how to accomplish this.) The first entry is highlighted, meaning it is the current entry. The fields it holds and their values are shown in the entry buffer in the bottom Ebib window. The first field is the type field, which tells you what kind of entry you're dealing with (i.e. `book`, `article`, etc.).

Below the type field, Ebib displays (up to) three sets of fields. The first set are the so-called obligatory 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 additional fields. These fields are usually ignored by BibTeX (note that BibTeX normally ignores *all* fields it does not know), although there are bibliography styles that treat some of these fields as optional rather than as additional; (i.e., the `harvard` styles do typeset the `url` field, if present.)

The first two groups of fields are different for each entry type, while the third group are common to all entry types. You can use the additional fields, for example, to add personal comments to the works in your database. Ebib by default defines the following additional fields: `crossref`, `url`, `annote`, `abstract`, `keywords`, `file` and `timestamp`. If these are not sufficient for you, you need to customise Ebib and add your own fields. (See Additional Fields, if you need to find out how to do that.)

To move around in the index buffer, you can use the `up` and `down` cursor keys, `C-p` and `C-n`, or for those more used to mutt's key bindings, `k` and `j`. Furthermore, `Space` and `PgDn` move a screenful of entries down, while `b` and `PgUp` move in the other direction. Lastly, `g` and `Home` move to the first entry, while `G` and `End` move to the last one.

Ebib is not restricted to opening just one `.bib` file at a time. You can open more files by just typing `o` again and entering the filename. Ebib numbers the databases: the number of each database is shown in the mode line of the index buffer, directly before the database name. The keys 1–9 provide a quick way of jumping from one database to another. Note that the numbering is dynamic: if you have three databases opened and then close the second, database 3 becomes database 2.

With the `left` and `right` cursor keys, you can move to the previous or next database. These keys wrap, so if you hit the `left` cursor key while the first database is active, you move to the last database. If you are done with a database and want to close it, type `c`. This closes the current database. It does not leave Ebib, and all other databases you have open will remain so.

Starting a New `.bib` File

If you want to start a new `.bib` file from scratch, you cannot just go and enter entries. You first have to give the database a name. So, to start a new database, type `o` first, and give the new file a name. Once you have done this, you can start adding entries to the database.

Editing the Database

Of course, being able to open and view `.bib` files is only half the fun. One needs to be able to edit the files as well. Ebib's essential editing facilities are discussed here.

Adding and Deleting Entries

To add an entry to a database, you type `a`. When you do this, Ebib first asks you for an entry key, as every entry must be identified by a unique key. Just type a name for the new entry (say `jones1998`). Since the entry key must be unique, Ebib will complain if you enter a key that already exists.

You can also have Ebib automatically generate entry keys: if you set the customisation option Autogenerate Keys, Ebib does not ask you for a key when you add a new entry. Instead, it creates a temporary key (of the form `<new-entry>`). When you have finished entering the field values for the new entry, Ebib automatically replaces the temporary key with a key that is based on the contents of the `author` (or `editor`), `year` and `title` fields.

Note that if you should later decide that you want to change the key of an entry, you can do so with the command `E`. So if you have an entry with the key `jones1998` and you want to add another entry by Jones from 1998, you can call the new one `jones1998b` and rename the existing one to `jones1998a`. Similarly, it is possible to let Ebib recreate an autogenerated key by pressing `K`.

Deleting an entry is done with `d`. Be careful with this: you will be asked for confirmation, but once you've confirmed, the entry is gone, and it is not possible to bring it back. There is no undo in Ebib. (If you haven't saved the database yet, it is still possible to retrieve the deleted entry from the `.bib` file, and otherwise it may still be in the backup file that Ebib creates. See Saving a Database.)

Editing Fields Values

Editing the field values for an entry is done in the lower of the two Ebib buffers, the so-called entry buffer. You can move focus to the entry buffer by typing the command `e` in the index buffer.

You can move between fields with the same keys that you use to move between entries in the index buffer: the cursor keys `up` and `down`, `C-p` and `C-n`, or `j` and `k`. `Space` and `PgDn` move to the next set of fields, while `PgUp` and `b` move to the previous set of fields. `g` and `G`, and `Home` and `End` also work as expected.

Editing a field value can be done with `e`. (In fact, in the entry buffer, `RET` is equivalent to `e`.) For most fields, Ebib simply asks you for a string value in the minibuffer. (Here, `RET` confirms the edit, while `C-g` cancels it.) Although BibTeX requires that field values be surrounded by braces {} (or double quotes ""), but Ebib does not use those, even though it can of course handle them when they are used in an existing `.bib` file) you do not need to type these. Ebib adds them when it saves the `.bib` file.

Some fields, however, are handled in a special way. The first of these is the `type` field: if you edit this field, you must enter one of the predefined entry types. Ebib won't allow you to enter anything else. You can use tab-completion in this case. Similarly, if you edit the `crossref` field, Ebib requires that you fill in a key from the database. Here, too, you can use tab-completion.

Note that if you're adding a new entry, Ebib automatically puts you in the entry buffer after you've typed the entry key: you don't have to type `e` to move to the entry buffer. When creating a new entry, it is best to set the `type` field first, because the `type` field determines which other fields are available for an entry.

Note also that after editing a field, Ebib (usually) puts you on the next field. This is convenient if you're creating a new entry and need to fill out several fields in a row.

If you're done editing the fields of the entry, type `q` to move focus back to the index buffer. (Note: keys may have different functions in the index buffer and the entry buffer. `q` is a typical example: in the entry buffer, it quits editing the entry and moves focus back to the index buffer. In the index buffer, however, `q` quits Ebib.)

Editing Multiline Values

Apart from the `type` and `crossref` field, there is another field that Ebib handles in a special way when you edit its value. This is the `annote` field. Most field values normally consist of a single line of text. However, because the `annote` field is meant for creating annotated bibliographies, it would not be very useful if you could only write one line of text in this field. Therefore, when you edit the `annote` field, Ebib puts you in the so-called *multiline edit buffer*. This is essentially a text mode buffer that allows you to enter as much text as you like. To store the text and leave the multiline edit buffer, type `C-c | q`.

If you want to leave the multiline edit buffer without saving the text you have just typed, type `C-c | c`. This command cancels the edit and leaves the multiline edit buffer. The text that is stored in the field you were editing is not altered.

Multiline values are not restricted to the `annote` field. Any field can in fact hold a multiline value. (Except of course the `type` and `crossref` fields.) To give a field a multiline value, use `l` instead of `e`. You will again be put in the multiline edit buffer, where you can edit the value. Note that you can use `l` even if a field already has a single line value. Ebib will just make that the first line in the multiline edit buffer.

When a field has a multiline value, only the first line is shown in the entry buffer, for space reasons. To indicate that the value is multiline, a plus sign `+` is placed in front of the value.

By the way, the `e` key is smart about the way an entry must be edited. If you press `e` on a field that already has a multiline value, regardless of the fact whether it is the `annote` field or not, Ebib puts you in the multiline edit buffer. Therefore, you need `l` only if you want to give a field a multiline value when it doesn't have one yet.

For more details on working with the multiline edit buffer, see The Multiline Edit Buffer.

Copy, Cut, Paste (Yank), and Delete

A few more commands are available when you're in the entry buffer editing field values. The commands `c`, `x` and `y` implement a copy and paste system: `c` copies the contents of the current field to the kill ring, `x` kills the contents of the current field to the kill ring, and `y` yanks (pastes) the most recently killed text in the kill ring. You can type `y` repeatedly to get the same effect you get in Emacs when you type `M-y` after an initial `C-y`: every additional use of `y` moves back in the kill ring.

Lastly, there is the command `d`, which deletes the contents of the current field without storing the text in the kill ring. (It asks for confirmation, though, just to make sure.)

Note that `y` only works when the current field does not have a value yet. This is to prevent you from accidentally overwriting a field value. If you do want to yank text into a field that already has a value, simply hit `d` first to delete the text.

Saving a Database

When you have undertaken any kind of editing action on a database, it is marked as modified, which is indicated in the mode line for the index buffer. A modified database can be saved by typing `s`. This saves the database to the file it was loaded from without asking for confirmation. (It is similar to `C-x C-s` in Emacs.) If you're saving a file for the first time after loading it, Ebib creates a backup file. (Ebib honours `backup-directory-alist` when saving backups. Note that you can also disable backups altogether with the option Create Backups.

If you have multiple databases open, have made changes in more than one of them, and want to save all of them without going through each yourself, you can save all databases at once through the menu.

The menu also provides a way to save the database to another name. This command is similar to `C-x C-w` in Emacs, so that after using it, the new `.bib` file becomes associated with the database.

Searching

Ebib provides several search methods. First, if you are in the index buffer, the normal Emacs incremental searches, `C-s` and `C-r`, function as expected. You can use them to search entry keys. Note that once you've found the key you're searching, you must hit `RET` to quit the search and again `RET` to make the entry you found active. Ebib does not update the entry buffer during incremental search, as this would be rather pointless: you're only interested in the entry you're searching for, not in the entries you pass along the way.

Of course, it is also possible to search the database itself. If you type `/`, Ebib asks you for a search term. This can be a regular expression, to allow for flexibility in searching. After hitting `RET`, Ebib will start searching the database (starting from the current entry, *not* from the first entry!) and will display the entry with the first occurrence of the search string that it finds. All the occurrences of the search string in that entry are highlighted.

Ebib searches all the fields of each entry. It is not possible with `/` to specify the fields to search. Note that if the search term is found in a field with a multiline value, Ebib will highlight the `+` sign that it displays in front of the field value. Keep an eye out for this when doing a search, because Ebib only shows the first line of multiline values, and if the search term appears in another line, the highlighted `+` is the only indication that the search term was found. (Well, that and the fact that Ebib does *not* say `Search string not found`, of course…)

A search term may of course appear more than once in the database. To search for the next occurrence, type `n`. This will continue searching for the search string in the rest of the database. Again, the first entry found to contain the search string is displayed. Note that `n` does not wrap: if the end of the database is reached, Ebib stops searching. To continue searching from the top, hit `g` and then `n`.

The functions described here form Ebib's basic search functionality. Ebib also has a much more powerful search mechanism in the form of *virtual databases*. These are described later. (See Virtual Databases.)

LaTeX Integration

Having a BibTeX database manager running inside Emacs has an additional advantage: it makes it trivially easy to insert BibTeX keys in your LaTeX documents. In fact, this functionality doesn't just work with LaTeX, but also with other text modes that have some form of citation commands, as discussed below.

Ebib provides two functions for this. First, if you're in a LaTeX buffer, you can call the function `ebib-insert-bibtex-key`. When you invoke this command, Emacs prompts you for a key from the database(s) associated with the current buffer, a citation command (that has to be typed *without* the backslash) and any optional argument(s) the command allows. You can type the key using TAB-completion, and after hitting `RET`, Emacs puts a BibTeX citation at the cursor position in the current buffer with the key you selected.

You can also do it the other way around: if you're in the index buffer in Ebib, you can *push* an entry to a LaTeX buffer. To do this, use the command `p`. Ebib will ask you for a buffer to push the entry to, a citation command and also any optional arguments, and then insert a citation at the current cursor position in the buffer you've supplied.

The citation command that `ebib-insert-bibtex-key` and the command key `p` ask for have to be defined before you can use them. (By default, only the `\cite` command has been predefined for LaTeX.) For details on setting this up, see Citation Commands.

Another Ebib command is available for LaTeX documents: the command `ebib-create-bib-from-bbl` creates a `.bib` file from the `.bbl` file associated with LaTeX document in the current buffer. This makes it easy to create a `.bib` file containing just the BibTeX entries that are used in the document. (One disadvantage is that cross-referenced entries aren't always in the `.bbl` file, depending on how your BibTeX style handles them. After using `ebib-create-bib-from-bbl` it may therefore be necessary to check whether you have all cross-referenced entries.)

There is another function that is available outside Ebib: `ebib-entry-summary`. This command reads the key under the cursor in the current buffer and displays the field values associated with that key in a `*Help*` buffer. This allows you to quickly check a reference in a text.

Probably the easiest way to use both `ebib-insert-bibtex-key` and `ebib-entry-summary` is to bind them to a key sequence. For example, you could put the following in your `~/.emacs`:

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

This binds `C-c b` to the command `ebib-insert-bibtex-key` in AUCTeX's LaTeX mode. (Note that commands of the form `C-c <letter>` are reserved for the user, and should therefore not be set by any package. For this reasons, Ebib does not set this command automatically.)

Orgmode and markdown

Orgmode can handle various types of links and new link types can be created by the user. If you use orgmode to write papers and want to use bibliographic references, it is easy to set up orgmode to use Ebib. If you add the following line to `~/.emacs`:

```
(org-add-link-type "ebib" 'ebib)
```

you can create org links of the following form:

```
[ebib:Jones1992][Jones (1992)]
```

Orgmode will display this link simply as `Jones (1992)` in your buffer. If you click this link (or press `C-c C-o` on it), you will be taken to Ebib and shown the the entry `Jones1992` in the active database.

A citation string for orgmode is predefined, so you can use `ebib-insert-bibtex-key` in orgmode buffers to insert links of the type above into your orgmode documents.

Note that such org links aren't properly translated to LaTeX citation commands when you export your org file. For that, you'll need to set up a different link type for each citation command and provide functions for exporting them to LaTeX. The orgmode documentation explains how this is done.

Another document format that provides support for automatically generating citations is Pandoc markdown. Pandoc's handling of citations is more sofisticated than orgmode's, as it can automatically translate citations in markdown text into LaTeX (specifically: `natbib` or `biblatex`) commands, but it also supports citations when converting to other document formats. In order to use Ebib with Pandoc markdown, nothing needs to be set up: Ebib has predefined citation formats for the citations forms that Pandoc markdown supports: `text`, `paren` and `year`. The `text` format produces citations of the form `@Abney1987 [p. 50]`, the `paren` format produces `[cf. @Abney1987, p. 50]` and the `year` format, which suppresses the author in the (LaTeX/html/what have you) output, produces `[-@Abney1987 p. 50]`. Of these three, the `paren` type can contain multiple citations (e.g., `[cf. @Abney1987, p. 50; @Dahl2004, p. 204]`), which Ebib can handle as well. See Citation Commands for further details.

Consulting Databases from within a LaTeX File

The commands `ebib-insert-bibtex-key` and `ebib-entry-summary` must consult the database or databases loaded in Ebib, and Ebib tries to be smart about which database(s) to consult. Usually, a LaTeX file has a `\bibliography` command somewhere toward the end, which names the `.bib` file or files that contain the bibliography entries. If you consult a BibTeX database from within a LaTeX file, Ebib first looks for a `\bibliography` command, reads the `.bib` files from it, and then sees if those files happen to be open. If they are, Ebib uses them to let you pick an entry key (in the case of `ebib-insert-entry-key`) or to search for the entry (in the case of `ebib-entry-summary`).

Of course, it may be the case that the LaTeX file is actually part of a bigger project, and that only the master file contains a `\bibliography` command. To accommodate for this, Ebib checks whether the (buffer-local) variable `TeX-master` is set to a filename. If it is, it reads that file and tries to find the `\bibliography` command there. (Note: `TeX-master` is an AUCTeX variable, which is used to keep track of multi-file projects. If you don't use AUCTeX, this functionality doesn't work, and Ebib will only check the current file for a `\bibliography` command.)

Note that if one of the `.bib` files in the `\bibliography` command isn't loaded, Ebib issues a warning message about this, and continues to check for the next `.bib` file. These warning messages appear in the minibuffer, but are probably directly overwritten again by further messages or prompts Ebib produces, so check the `*Messages*` buffer if Ebib doesn't seem to be able to find an entry that you're sure is in one of your databases.

Another thing to keep in mind is that Ebib only looks for a `\bibliography` command once: the first time either `ebib-insert-bibtex-entry` or `ebib-entry-summary` is called. It stores the result of this search and uses it the next time either of these commands is used. Therefore, if you make a change to the `\bibliography` command, you must reload the file (use `M-x revert-buffer`) to make sure Ebib rereads the `\bibliography` command.

If no `\bibliography` command is found at all, either in the LaTeX file itself, or in the master file, Ebib simply consults the current database, i.e. the database that was active when Ebib was lowered with `z`.

Cross-referencing

BibTeX has a cross-referencing facility. Suppose you have an entry `jones1998`, which appeared in a book that is also in your database, say under `miller1998`. You can tell BibTeX that `jones1998` is contained in `miller1998` by putting `miller1998` in the `crossref` field. When BibTeX finds such a cross-reference, all the fields of `jones1998` that don't have a value inherit their values from `miller1998`. At the very least, this saves you some typing, but more importantly, if two or more entries cross-reference the same entry, BibTeX automatically includes the cross-referenced entry in the bibliography (and puts a reduced reference in the cross-referencing entries).

When you fill in the `crossref` field in Ebib, Ebib displays the values of the cross-referenced entry in the entry buffer. To indicate that they are just inherited values, they are marked with `ebib-crossref-face`, which by default is red. (You can customise it, of course. See the customisation option Crossref Face.) These values are just displayed for convenience: otherwise, Ebib treats these fields as if they are empty. That is, they cannot be edited (to edit them, you need to edit the cross-referenced entry), and it's not possible to copy these values to the kill ring.

If you're viewing an entry that has a cross-reference and you want to go to the cross-referenced entry you can type `F`. This command reads the value of the `crossref` field and then displays that entry. If you want to do the reverse, i.e., see if the current entry is cross-referenced by any other entries, you can use the same key `F`: if you type `F` on an entry that does not have a cross-reference, Ebib makes the key of the current entry the current search string and searches for its first occurrence after the current entry. Note that after Ebib has jumped to the first cross-referencing entry, you cannot type `F` again to find the next one. This command will take you back to the cross-referenced entry. In order to find the next cross-referencing entry, you have to type `n`, as with a normal search.

Note that if you want to use BibTeX's cross-referencing options, you need to set the option Save Xrefs first. This tells Ebib to save all entries with a `crossref` field first in the `.bib` file. Without this, BibTeX's cross-referencing will not work reliably.

Selecting Entries

Commands in the index buffer generally operate on one single entry, or on all entries. For some, however, it may sometimes be useful to perform them on more than one entry, but not necessarily all of them. This can be achieved by selecting entries. You can select the entries you want to perform a command on with the key `m`. This selects (or unselects) the current entry. Selected entries are displayed in inverse video. Note that the face properties of selected entries can be customised through the customisation option Selected Face.)

Of the commands discussed so far, two can be used on selected entries: `d` and `p`. Note, however, that it is not enough to select the entries you want and then type any of these commands. If you do so, they will behave as if no entries were selected. To get these commands to work on the selected entries, you have to type a semicolon before them. That is, `; d` deletes all selected entries, `; p` pushes all selected entries to a LaTeX buffer. The command `m` itself can also be used with the `;` prefix. If there are any selected entries, `; m` unselects them all. Otherwise, `; m` selects all entries.

When using `; p` to push all selected entries to a LaTeX buffer, they are put in a single citation command, separated by commas. Ebib does not create a citation command for each entry separately.

Printing the Database

Sometimes it may be useful to have a `.pdf` file or print-out of your database. Although Ebib does not actually do the printing itself, it can create a LaTeX file for you that you can compile and print. In fact, there are two ways of doing this.

The print options are available in the Ebib menu when the index buffer is active. You can print the entries as index cards or as a bibliography.

If you print your entries as a bibliography, Ebib creates a simple LaTeX document that essentially contains a `\nocite{*}` command followed by a `\bibliography` command referring to the `.bib` file belonging to the current database. You can then run the usual sequence of LaTeX, BibTeX, LaTeX, LaTeX on this file, creating a document containing a list of all the references in your database.

If you choose to print as index cards, Ebib also creates a LaTeX file. However, instead of simply providing a `\nocite{*}` command, this file contains a `tabular` environment for each entry in the database listing all the fields of that entry and their values.

The entries are separated by a `\bigskip`, but if you set the option `Print Newpage` in the customisation buffer (or in the Print menu), the entries are separated by a `\newpage`, so that every entry is on a separate page. The latter option is useful when printing actual index cards (though you'd probably have to change the page size with the `geometry` package as well).

By default, the index cards only show single-line field values. That is, multiline values are normally excluded. If you want to include multiline values in the print-out, you have to set the option `Print Multiline` in the Options menu or in Ebib's customisation buffer. (See The Customisation Buffer.) With this option set, Ebib includes all multiline values in the LaTeX file that it creates. Note however that Ebib does not change anything about the formatting of the text in a multiline value. So if you plan to make (heavy) use of this option, make sure that the way you type your text conforms to LaTeX's conventions (e.g. empty lines to mark paragraphs, etc.) and doesn't contain any characters such as `&` that are illegal in LaTeX. (Or, alternatively, use LaTeX code in your multiline fields.)

As mentioned, when you "print" the database, Ebib really just creates a LaTeX file. More precisely, it creates a temporary buffer and writes the LaTeX code into it, and then saves the contents of that buffer to a file. After it has done that, Ebib lowers itself and instruct Emacs to open the file in a buffer, which will then be properly set up as a LaTeX buffer. From there you can run LaTeX and view the result.

Before doing all this, Ebib asks you which file to write to. Be careful with this: since this is supposed to be a temporary file, Ebib simply assumes that if you provide a filename of an existing file, it can overwrite that file without warning!

A better way to tell Ebib which file to use is to set the option `Print Tempfile` in Ebib's customisation buffer to some temporary file. When this option is set, Ebib will always use this file to write to, and will not ask you for a filename anymore.

Note that both print options operate on all entries of the database. If there are selected entries in the database, however, only those are printed.

There are two more customisation options for printing the database. These are `Print Preamble` and `LaTeX Preamble`. With these options, you can specify what Ebib should put in the preamble of the LaTeX files it creates. Use this if you want to use specific packages (e.g. `\usepackage{a4}` or `\usepackage{times}`). This is especially useful for printing a bibliography, since by default, Ebib uses BibTeX's standard bibliography style. With the option `LaTeX Preamble` you can set your preferred bibliography style. Details are discussed in the chapter on customisation, see The Customisation Buffer.

Calling a Browser

With more and more scientific literature becoming available on-line, it becomes common to store URLs and DOIs in a BibTeX database. Sometimes you may want to load such a URL or a DOI in your browser. Ebib provides a convenient way for doing so.

If you type `u` in the index buffer, Ebib takes the first URL stored in the `url` field of the current entry and passes it to your browser. Furthermore, in the entry buffer, you can use `u` on *any* field. If you happen to have more than one URL stored in the relevant field, and you want to pass the second (or third, etc.) to the browser, you can use a prefix argument. So typing `M-2 u` sends the second URL to your browser, `M-3 u` the third, and so on.

It is not even necessary that the relevant field contains *only* URLs. It may contain other text mixed with the URLs: Ebib simply searches the URLs in the field and ignores the rest of the text. Ebib considers every string of characters that starts with `http://` or `https://` and that does not contain whitespace or any of the characters `" ' <` or `>` as a URL. Furthermore, Ebib regards everything that is enclosed in a LaTeX `\url{...}` command as a URL. This behaviour is controlled by a regular expression that can be customised. (See Url Regexp.)

Similarly, with the key `i` in the index buffer you can send a DOI to a browser. The DOI must be stored in the `doi` field. Ulike URLs, there can only be one DOI in this field. The whole contents of the field is assumed to be the DOI and is sent to the browser unchanged. A DOI is normally resolved through the URL `http://dx.doi.org/`, but if you prefer a different URL, you can customize the option Doi Url.

There exists an Emacs function `browse-url`, which provides a nifty interface to calling an external browser. In principle, Ebib uses this function. However, if this function is not present on your installation, you can set the option Browser Command to call the browser.

As just explained, if you press `u` in the index buffer, Ebib searches the `url` field of the current entry for URLs. If you have the habit of putting your URLs in another field, however, you may change the customisation option Standard Url Field and tell Ebib to use another field for searching the URLs. The same can be done for the DOI field.

Viewing Files

If you have electronic versions of the papers in your database stored on your computer, you can use Ebib to call external viewers for these files. The interface for this is similar to that for calling a browser: if you press `f` in the index buffer, Ebib searches the `file` field for a filename and when it finds one, calls an appropriate viewer.

Just as with `u`, you can use `f` in the entry buffer as well, in which case it can be used on any field, not just the `file` field. It is also possible to have more than one filename in a field: you can select the one you want to view with the prefix argument.

Just as in the case of URLs, you can customise several things about the file view functionality. The option Standard File Field allows you to customise the field that `f` extracts filenames from when pressed in the index buffer. Extracting filenames is done with a regular expression, which can be customised through the option File Regexp.

The option File Search Dirs allows you to tell Ebib which directories it needs to search for files. The default value is `~`, which means Ebib just looks in your home dir. Since this is probably not where you keep your files, you may want to customise this. Note that you can specify more than one directory.

Note that Ebib does not search directories recursively. It is possible, however, to put subdirectories in the filenames. That is, if you put something like `a/Abney1987.pdf` in the `file` field, Ebib searches for the relevant file in a subdirectory `a/` of the directories listed in the option File Search Dirs. As an extra service, Ebib also searches for the base filename, i.e., `Abney1987` in this particular case. This can come in handy when you keep the papers on your reading list in a separate directory.

Ebib can call different external programs depending on the file type of the relevant file, but you have to specify which programs to call. The option File Associations allows you to do this. By default, `.pdf` and `.ps` files are handled, by `xpdf` and `gv`, respectively. You can specify further file types by their extensions (do not include the dot). The program is searched for in `PATH`, but you can of course specify the full path to the program.

Advanced Features

The features discussed in the previous chapter should be sufficient to get started using Ebib. However, Ebib has several more advanced features, which are described in this chapter.

Screen Layout

By default, Ebib takes over the entire Emacs frame it is started in. If you have a wide enough screen, however, it may be more convenient to have Ebib take up only part of the frame, so that you can have the LaTeX text you're working on and Ebib visible at the same time. The option Layout allows you to do this, by giving you the ability to choose between a full-frame or a split-frame layout.

In the split-frame layout, the Ebib windows are displayed on the right of the current frame, with the left part free for your document. In this layout, some aspects of Ebib behave somewhat differently. Most importantly, the multiline edit buffer is not displayed in the lower Ebib window, but in the non-Ebib window on the left. (Obviously, after leaving the multiline edit buffer, the original buffer is restored to that window.)

Furthermore, pressing `z` in the index buffer leaves Ebib, but keeps the buffers visible. You can get back to Ebib with the command `M-x ebib` (or any key bound to it, of course), or simply by manually switching to the index buffer. If you want to remove the Ebib buffers from the frame but keep Ebib in the background, you can use `Z` (i.e. uppercase `Z`) in the index buffer. (Note that `Z` is also available in the full-frame layout, but there it is identical to `z`.)

Lastly, the command `ebib-entry-summary` checks whether the Ebib buffers are visible in the frame. If they are, it does not output the entry info in a `*Help*` buffer, but rather displays the entry in Ebib itself.

Preloading `.bib` Files

Chances are that you will be doing most of your work with one or a few `.bib` files, and you may find yourself opening the same file or files every time you start Ebib. If so, you can tell Ebib to always load specific `.bib` files on startup. To do this, specify the files in Ebib's customisation buffer, under the option Preload Bib Files.

By default, `.bib` files are searched for in your home directory. Since this is most likely not where you keep the files, you need to specify either the file's full path or a relative path starting from your home

directory. Alternatively, you can customize the option Preload Bib Search Dirs to specify one or more directories in which Ebib should search the `.bib` files.

@Preamble Definition

Apart from database entries, BibTeX allows three more types of elements to appear in a `.bib` file. These are `@comment`, `@preamble` and `@string` definitions. Ebib provides facilities to handle the latter two. `@comment` definitions cannot be added to a `.bib` file through Ebib, and if Ebib finds one in a `.bib` file, it is simply ignored (and dropped from the file when you save it).

`@preamble` and `@string` definitions can be handled, however. Ebib allows you to add one `@preamble` definition to the database. In principle, BibTeX allows more than one such definition, but really one suffices, because you can use the concatenation character `#` to include multiple TeX or LaTeX commands. So, rather than having two `@preamble` definitions such as:

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

you can write this in your `.bib` file:

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

Creating or editing a `@preamble` definition in Ebib is done by hitting `P` (uppercase `P`) in the index buffer. Ebib uses the multiline edit buffer for editing the text of the `@preamble` definition, which means that as discussed above, `C-c | q` stores the `@preamble` text and returns focus to the index buffer, while `C-c | c` returns focus to the index buffer while abandoning any changes you may have made. (For details on using the multiline edit buffer, see The Multiline Edit Buffer.)

In order to create a `@preamble` as shown above in Ebib, you only have to type the text between the braces. Ebib takes care of including the braces of the `@preamble` command, but otherwise it saves the text exactly as you enter it. So in order to get the preamble above, you'd have to type the following in Ebib:

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

Note that when Ebib loads a `.bib` file that contains more than one `@preamble` definition, it concatenates all the strings in them in the manner just described and saves them in one `@preamble` definition.

@String Definitions

If you press `S` (that's a uppercase `S`) in the index buffer, Ebib hides the entry buffer in the lower window and replaces it with the *strings buffer*. In this buffer, you can add, delete and edit `@string` definitions.

Adding a `@string` definition is done with the command `a`. This will first ask you for an abbreviation and then for the value to be associated with that abbreviation. Once you've entered these, Ebib will sort the new abbreviation into the buffer.

Moving between the `@string` definitions can be done in the usual way: the cursor keys up and down, `C-p` and `C-n` and `k` and `j` move up and down. `Space` and `PgDn` move ten strings down, while `b` and `PgUp` move in the other direction. The keys `g`, `G`, `Home` and `End` also function as expected.

To delete a `@string` definition, use `d`. To edit the value of a definition, use `e`. There is also a command `c`, which copies the value of the current `@string` definition to the kill ring. Unlike in the entry buffer, there are no corresponing commands `y` and `x`. (In fact, `x` does exist, but has another function.) Yanking from the kill ring can be done with `C-y/M-y` in the minibuffer when you edit a `@string`'s value. Cutting a `@string`'s value is pointless, because a `@string` definition must have a value.

Having defined `@string` definitions, there must of course be a way to use them. Just giving a field a string abbreviation as value will not do, because Ebib puts braces around the value that you enter when it writes the `.bib` file, so that BibTeX will not recognise the abbreviation, and will not expand it. BibTeX will only recognise an abbreviation if it appears in the `.bib` file outside of any braces.

To accomplish this, you must mark a field's value as *raw*. A raw field is a field whose value is not surrounded by braces when the database is saved, so that BibTeX recognises it as an abbreviation. To mark a field raw, press `r`. An asterisk will appear before the field, indicating that it is raw. Pressing `r` again will change the field back to normal. If you press `r` on a field that does not have a value yet, Ebib will ask you for one.

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.

An easy way to enter a `@string` abbreviation as a field value is to use the key `s` instead of `e`. If you type `s`, Ebib asks you for a `@string` abbreviation to put in the current field, and automatically marks the field as raw. With this command, Ebib only accepts `@string` definitions that are in the database, so that by using `s` you can make sure you don't make any typos. Note that you can use tab completion to complete a partial string.

Managing keywords

Ebib provides some options for handling keywords. By default, there is a `keyword` field in the list of additional fields. Editing this field is a bit different from other fields, however. Instead of just entering a string and hitting ENTER to store it and return to the entry buffer, you should enter a single keyword

and hit enter. The keyword will then be added to the keywords already present and you are asked to enter the next keyword. If you've added all keywords you want, you just hit ENTER to finish.

The advantage of doing it this way is that you can reuse keywords: once you've added a keyword to one entry, Ebib remembers it. The next time you want to use the same keyword for a different entry, you just need to type the first (few) letters, hit TAB and the keyword will be completed. That makes it easier to ensure you use the exact same keywords in different entries.

Note that Ebib's keyword functionality is not used to check the contents of keyword fields. It is simply a way to make it easier to stick to specific keywords, which should make it easier to categorise and search your entries. It is still possible to edit the `keyword` field directly. To do so, use a prefix argument: `C-u e` (or any other prefix argument) on the `keyword` field will allow you to edit the entire contents in the normal way. Use this method if you want to remove single keywords. (Blanking the entire `keyword` field is quicker with `x` or `d`.)

Remembering keywords is practical, but it is even more useful if remembered keywords can be saved, so that they are available the next time you start Ebib. There are two ways of doing this: first, there is an option `ebib-keywords-list` that you can use to store keywords. (See Keywords List.) Keywords stored in this option will be available for TAB completion to all databases in Ebib. New keywords, however, will not automatically be stored. If you find you need a keyword not on the list and want to make it permanent, you'll have to add it to `ebib-keywords-list` yourself.

The other way of making keywords permanent is by storing them in a file. Ebib offers two ways of doing this (which are mutually exclusive, so you have to choose one). You can either configure a single keyword file, with keywords that are available to all databases, or you can configure per-directory keyword files, with keywords that are available for all `.bib` files in the same directory. You can set up keyword files by configuring the option Keywords File. You can either set it to use a single keyword file, in which case you need to specify a file with its full path, or you can use per-directory keyword files, in which case you must provide a filename without a path. That is, if you use per-directory keyword files, the files have the same name in each directory. The default name is `ebib-keywords.txt`, but you can change that if you like, of course.

Keyword files have a very simple format: they are text files with one keyword per line. So you can easily create or edit keyword files by hand, or have them autogenerated by some other programme. Keep in mind, though, that Ebib does not check for changes to keyword files. If you have a single keyword file, it is loaded when Ebib starts up; per-directory keyword files are loaded when the first `.bib` file in that directory is opened. If you open a second `.bib` file from the same directory, Ebib won't reload the keywords file.

When you close a database, Ebib checks if you have added new keywords to it and asks you if you want to save them. You can tell Ebib to save new keywords automatically by setting the option Keywords File Save On Exit to `always`. Note that this doesn't save the keywords when you enter them, only when you close the database or quit Ebib. You can also set this option to `never`, which means Ebib will discard new keywords when the database is closed. Note that there are also two menu options for saving keywords.

The option Keywords Use Only File controls whether Ebib uses only the keyword file, or both the keyword file and the configured keyword list. This option is only useful when you have configured a keyword file. In that case, setting this option to use both the keyword list and the keyword file tells Ebib to offer keywords from both sources when you edit the keyword field. Otherwise, only the keyword file is used.

It is also possible to tell Ebib to sort the keywords in the `keywords` field in alphabetical order. Set the option Keywords Field Keep Sorted if you want to do this. Note that setting this option also automatically removes duplicates.

Lastly, you can configure the separator used between keywords in the keyword field. By default, it is set to `";  "`, i.e., semicolon plus space. If you change it, keep in mind that Ebib does not add a space between keywords, so if you want a space, make sure to add it to the separator.

Sorting the `.bib` File

By default, the entries in the database are saved to the `.bib` file in alphabetical order according to entry key. If you only deal with the `.bib` file through Ebib, you may not care in which order the entries are saved. However, it may sometimes be desirable to be able to specify the sort order of entries in more detail. (Apparently, this can be useful with ConTeXt, for example.)

You can specify a sort order in Ebib's customisation buffer. To sort the entries, you must set at least one sort level (that is, a field to sort the entries on). You can also specify more than one sort level: if two entries have identical values for the first sort level, they will be sorted on the second sort level. E.g., if the first sort level is `author` and the second is `year`, then the entries are sorted by author, and those entries that have identical values for the `author` field are sorted by year.

A sort level is not restricted to a single field. You can specify more fields for a single sort level. Within a single sort level, a second sort field is used if the first sort field does not have a value. For example, books that have an editor instead of an author will have an empty `author` field. If you sort the database on the `author` field, such entries will all appear at the beginning of the `.bib` file, which is most likely not what you want.

To remedy this, you can specify both the `author` and the `editor` fields for the first sort level. Ebib will then sort an entry on its `author` field if it has a value, and will otherwise use the value of the `editor` field.

The difference between two sort fields within one sort level and two sort levels is that a second sort *field* is an alternative for the first field when it has no value, while a second sort *level* is an additional sort criterion when two or more entries cannot be sorted on the first level, because they have identical values.

By default, the option `Sort Order` has no value, which means that the entries in the `.bib` file are sorted according to entry key. Those that wish to customise the sort order will usually want to set the first sort level to `author editor`, and the second to `year`. In that way, the entries in the `.bib` file are sorted according to author/editor, and entries with the same author/editor are sorted by year.

Entries that cannot be sorted on some sort level, because the sort fields are empty, are sorted on entry key. (Keep in mind that if the first sort level yields *no value* for a specific entry, Ebib does *not* use the second sort level to sort that entry. It uses the entry key. The second sort level is only used if the first yields *identical* values for two or more entries.)

Note that if you have set the option `Save Xrefs First` (see Cross-referencing), it is pointless to set a sort order. Saving cross-referencing entries first messes up any sort order, so Ebib simply ignores the sort order if `Save Xrefs First` is set.

Merging and Importing

As described in the previous chapter, adding entries to a database can be done manually with the key `a`. There are other ways of adding entries to a database, however.

In the index buffer, the Ebib menu has an option to merge a second .bib file into the current database. Ebib reads the entries in this file and adds them to the database. Duplicate entries (that is, entries with an entry key that already exists in the database) will not be loaded. Ebib logs a warning about each duplicate entry to its log buffer and displays a warning after loading the `.bib` file when this happens.

Another way to add entries to a database is to import them from an Emacs buffer. If, for example, you find ready-formatted BibTeX entries in a text file or e.g. on the internet, you can copy & paste them to any Emacs buffer (e.g. the `*scratch*` buffer), and then execute the command `M-x ebib-import`. Ebib then goes through the buffer and loads all BibTeX entries it finds into the current database (i.e. the database that was active when you lowered Ebib). If you call `ebib-import` while the region is active, Ebib only reads the BibTeX entries in the region.

Exporting Entries

Sometimes it can be useful to copy entries from one database to another, or to create a new `.bib` file with several entries from an existing database. For this purpose, Ebib provides exporting facilities. To export an entry to a `.bib` file, use the command `x`. Ebib will ask you for a filename to export the entry to. (If you have already exported an entry before, Ebib will present the filename you used as default, but you can of course change it.)

For obvious reasons, Ebib appends the entry to the file that you enter if it already exists, it does not overwrite the file. If this is not what you want, delete the file first, as Ebib provides no way to do this.

If you have more than one database open in Ebib, it is also possible to copy entries from one database to another. To do this, use the `x` command with a numeric prefix argument. E.g., if the database you want to export an entry to is the second database, type `M-2 x` to export the current entry to it. The number of the database is given in the modeline of the index buffer.

If the database you're copying an entry to already contains an entry with the same entry key, Ebib won't copy the entry, and issues an appropriate warning message.

Note that the command `x` can operate on selected entries. So to export several entries in one go select them and type `; x`. You can use a prefix argument in the normal way: `M-2 ; x` exports the selected entries to database 2.

Apart from entries, it is also possible to export the `@preamble` and `@string` definitions. The `@preamble` definition is exported with the command `X` in the index buffer. `@string` definitions can be exported in the strings buffer: `x` in this buffer exports the current string, while `X` exports all `@string` definitions in one go. All these commands function in the same way: when used without a prefix argument, they ask for a filename, and then append the relevent data to that file. With a numeric prefix argument, they copy the relevant data to the corresponding open database.

Timestamps

Ebib provides the possibility to add a timestamp to every new entry, recording the time it was added to the database. The timestamp is recorded in the (additional) field `timestamp`. (By default, this field is not shown, but you can make it visible by checking the option "Show Hidden Fields" in the Options menu.)

You can tell Ebib to create timestamps by setting the option `Use Timestamp` in Ebib's customisation buffer. With this option set, a timestamp is included in entries added to the database with `a`. Ebib will

also add a timestamp to entries imported from a buffer or merged from a file, and to entries exported to another database or to a file. When importing or exporting entries, existing timestamps will be overwritten. The logic behind this is that the timestamp records the date and time when the entry was added to the database, not when it was first created.

Note that if this option is unset, the timestamp of an entry is retained when it's imported or exported. Therefore, if you record timestamps and want to im-/export entries without changing their timestamps, temporarily unset this option.

Ebib uses the function `format-time-string` to create the timestamp. The format string that Ebib uses can be customised in Ebib's customisation buffer. The default string is `"%a %b %e %T %Y"`, which produces a timestamp of the form `"Mon Mar 12 01:03:26 2007"`. Obviously, this string is not suited for sorting, so if you want to be able to sort on timestamps, you'll need to customise the format string. See the documentation for `format-time-string` on the options that are available.

Multiple Identical Fields

Under normal circumstances, a BibTeX entry only contains one occurrence of each field. If BibTeX notices that an entry contains more than one occurrence of an obligatory or optional field, it issues a warning. Ebib is somewhat less gracious, it simply takes the value of the last occurrence without giving any warning. (Note, by the way, that BibTeX will use the value of the *first* occurrence, not the last.) When additional fields appear more than once in an entry, BibTeX does not warn you, since it ignores those fields anyway. Here, too, Ebib's standard behaviour is to ignore all but the last value.

However, some online reference management services "use" this feature of BibTeX in that they put multiple `keywords` fields in the BibTeX entries that they produce. If you were to import such an entry into Ebib, you would lose all your keywords except the last one. To remedy this, you can tell Ebib that it should allow multiple occurrences of a single field in a BibTeX entry. You can do this by setting the customisation option Allow Identical Fields.

With this option set, Ebib collapses the multiple occurrences into a single occurrence. All the values of the different occurrences are collected and stored in the single occurrence, separated by semicolons. That is, Ebib does not retain the multiple occurrences, but it does retain the values. So suppose you have an entry that contains the following `keywords` fields:

```
@book{jones1998,
    author = {Jones, Joan},
    year = {1998},
    ...
    keywords = {sleep},
    keywords = {winter},
    keywords = {hybernation}
}
```

If you load this entry into Ebib with the option `Allow Identical Fields` set, you will get the following:

```
@book{jones1998,
    author = {Jones, Joan},
    year = {1998},
```

```
    ...
    keywords = {sleep; winter; hybernation}
}
```

Virtual Databases

In the previous chapter, Ebib's basic search functionality was discussed. (See Searching.) Ebib also provides a much more sophisticated search and filtering mechanism in the form of *virtual databases*.

A virtual database is a database that is not associated with any `.bib` file. Rather, it is created from another database by selecting entries from it based on a specific search pattern, called a *filter*. This allows you, for example, to select all entries from a database that contain the string "Jones" in their `author` field. A filter can be as complex as you want: you can select all entries that do *not* contain "Jones" in the `author` field, or all entries that contain "Jones" in either the `author` or the `editor` field, or all entries that contain "Jones" in the `author` field, and "symbiotic hybernation" in the `keyword` field, etc. Basically, the filter can consist of an arbitray number of search criteria combined with the logical operators `and`, `or` and `not`.

Simple Selection

Creating a virtual database is simple: press `&`, and Ebib will ask you for a field to select on, and for a regular expression to select with. So if you want to select all entries that contain "Jones" in the `author` field, you press `&` and type `author` as the field and `Jones` as the regexp to filter on.

Ebib will then create a virtual database containing the entries matching your selection criterion. A virtual database has the same name as the database it is based on, prepended with `V:`. It also has a number like any other database, and you can move back and forth to other databases with the number or cursor keys.

If you don't want to filter on one specific field but rather want to select all entries that match a certain regexp in any field, you can type `any` as the field to filter on. So specifying `any` as the field and `Jones` as the regexp, the virtual database will select all entries that have a field that contains "Jones" in them.

Complex Filters

Once you have a virtual database, it remains associated with the database it was created from. This means that you can refine or extend the selection (i.e. the filter) that the virtual database is based on. If, in the current example, you want to include all the entries that have "Jones" in the `editor` field, you have to perform a logical `or` operation: you want to select an entry if it contains "Jones" in the `author` field (which you already did) *or* if it contains "Jones" in the `editor` field.

A short sidenote: the first impulse in a case like this might be to use `and` instead of `or`: after all, you want to select all entries that contain "Jones" in the `author` field *and* all entries that contain "Jones" in the `editor` field. However, the filter that you build up is used to test each entry *individually* whether it meets the selection criterion. An entry meets the criterion if it contains "Jones" in the `author` field *or* if it contains "Jones" in the `editor` field. Therefore, `or` is the required operator in this case. If you would use `and`, you would only get those entries that contain "Jones" in both the `author` *and* `editor` fields.

To perform a logical `or` operation, press the key `|`. As before, you will be asked which field you want to filter on, and which regexp you want to filter with. Ebib will then update the virtual database with all entries in the original database that match the additional criterion.

It is also possible to perform a logical `and` on the virtual database. Use this if you want to select those entries that contain "Jones" in the `author` field and e.g. "symbiotic hybernation" in the `keyword` field. A logical `and` operation is done with the key `&`. (Note: this is the same key that is used to create a virtual database. In fact, you can also create a virtual database with `|`: when used in a normal database, `&` and `|` are equivalent. They are only different in virtual databases.)

Both the `&` and `|` commands can be used with the negative prefix argument `M--` (or `C-u -`, which is identical). In this case, the search criterion is negated. That is, the negative prefix argument performs a logical `not` operation on the search criterion.

That is, if you want to select all entries from a database that do *not* contain "Jones" in the `author` field, you can do this by typing `M-- &` and then filling out the relevant field and regexp. This prefix argument is available both in real and in virtual databases.

There is another way of performing a logical `not` operation, which is only available in virtual databases: by pressing the key `~`, you invert the current filter. That is, if you have a virtual database with all the entries containing "Jones" in the `author` or in the `editor` field, and you press `~`, the selection is inverted, and now contains all entries that do *not* have "Jones" in the `author` or `editor` field.

Although `~` and the negative prefix argument to `&` or `|` both perform logical `not` operations, they are *not* equivalent: `~` negates the entire filter built up so far, while the negative prefix argument only negates the single selection criterion you enter with it.

If you want to know what the filter for the current virtual database is exactly, you can type `V`. This command displays the current filter in the minibuffer. The filter is specified as a Lisp expression, meaning that the operators appear before their operands, not in between them. That is, `x and y` is written as `(and x y)`.

With a prefix argument (any prefix argument will do), the command `V` not only displays the current filter, but also reapplies it. This can be useful when you've made changes to the source database: Ebib does not automatically update a virtual database when its source database is modified.

Properties of Virtual Databases

Virtual databases differ from normal databases in several ways. First, they cannot be modified: you cannot add or delete entries, and you cannot modify the contents of fields. It is also not possible to import entries to them or merge another file with them. Furthermore, it is not possible to export entries to them or from them.

A virtual database cannot be saved in the normal way with `s`, and the menu option to save all databases ignores virtual databases. If you want to save a virtual database, you can use the command `w`. This command not only saves the virtual database, it also changes it into a normal database, and detaches it from its original source database, so that you can modify it without affecting the source database.

The print bibliography command also doesn't work with virtual databases. The reason for this is that the virtual database is not associated with an actual `.bib` file, so there is no file to create a list of references from. However, it is possible to print index cards frm a virtual database.

The Multiline Edit Buffer

As mentioned several times before, 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. This section discusses the details of this buffer.

Ebib enters multiline edit mode in one of three cases: when you edit the `@preamble` definition, when you hit `l` in the entry buffer to edit the current field as multiline, or when you hit `e` on the `annote` field, or on a field whose value already is multiline.

The mode that is used in the multiline edit buffer is user-configurable. The default value is `text-mode`, but if you prefer to use some other mode, you can specify this through the customisation options. (Personally, I use `markdown-mode` in the multiline edit buffer, so that I can use `markdown` to write annotations, which provides an easy way to create headers, use bold and italic, etc., in plain text.)

Three commands are relevant for interacting with Ebib when you're in the multiline edit buffer, which are bound to key sequences in the minor mode `ebib-multiline-edit-mode`, which is activated automatically in the multiline edit buffer.

`ebib-quit-multiline-edit`, bound to `C-c | q`, leaves the multiline edit buffer and stores the text in the database. If you invoke this command when you've deleted all contents of the buffer (including the final newline!) and you were editing a field value or the `@preamble`, the field value or preamble is deleted. (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 entry buffer.) If you were editing a `@string` value, Ebib will just complain, because string definitions cannot be empty.

`ebib-cancel-multiline-edit`, bound to `C-c | c`, also leaves the multiline edit buffer, but it does so without storing the text. The original value of the field, string or preamble will be retained. If the text was modified, Ebib will ask for a confirmation before leaving the buffer.

`ebib-save-from-multiline-edit`, bound to `C-c | s`, can be used in the multiline edit buffer to save the database. This command first stores the text in the database and then saves it. Because Ebib does not do an autosave of the current database, it is advisable to save the database manually every now and then to prevent data loss in case of crashes. It would be annoying to have to leave the multiline edit buffer every time you want to do this, so this command has been provided to allow you to do this from within the buffer.

Admittedly, the key combinations of the multiline edit buffer are somewhat awkward. The reason for this is that these commands are part of a minor mode, which restricts the available keys to combinations of `C-c` plus a non-alphanumeric character. However, it is possible to change the key commands, if you wish. Ebib itself provides a method to change the second key of these commands: see Modifying Key Bindings for details. You could change the `|` to e.g., `c` or `C-c`, if these do not conflict with any key commands in the major mode used for the multiline edit buffer.

Even more drastically, you could put something like the following in your `~/.emacs`:

```
(eval-after-load 'ebib
  '(progn
     (define-key ebib-multiline-mode-map "\C-c\C-c" 'ebib-quit-multiline-edit)
     (define-key ebib-multiline-mode-map "\C-c\C-q" 'ebib-cancel-multiline-edit)
     (define-key ebib-multiline-mode-map "\C-c\C-s" 'ebib-save-from-multiline-edit)))
```

This sets up `C-c C-c`, `C-c C-q` and `C-c C-s` for use in the multiline edit buffer. Since such key combinations are restricted for use with major modes, however, Ebib cannot set these up automatically, but as an Emacs user, you are free do do as you like, of course.

The Options Menu

In the index buffer, Ebib's menu has an Options submenu. This menu gives you quick access to Ebib's customisation buffer, and it also provides checkboxes for several settings that can be toggled on and off. All of these settings have defaults that can be defined in the customisation buffer. Setting or unsetting them in the Options menu only changes them for the duration of your Emacs session, it doesn't affect the default setting.

The same is true for the printing options that are in the Print menu. When set or unset in the menu, the default values specified in the customisation buffer do not change.

The Ebib Buffers

This chapter lists all the key commands that exist in Ebib, with a short description and the actual command that they call. The latter information is needed if you want to customise Ebib's key bindings. (See Modifying Key Bindings.)

The Index Buffer

`Up` go to previous entry. `(ebib-prev-entry)`

`Down` go to next entry. `(ebib-next-entry)`

`Right` move to the next database. `(ebib-next-database)`

`Left` move to the previous database. `(ebib-prev-database)`

`PgUp` scroll the index buffer down. `(ebib-index-scroll-down)`

`PgDn` scroll the index buffer up. `(ebib-index-scroll-up)`

`Home` go to first entry. `(ebib-goto-first-entry)`

`End` go to last entry. `(ebib-goto-last-entry)`

`Return` make the entry under the cursor current. Use after e.g. `C-s`. `(ebib-select-entry)`

`Space` equivalent to `PgDn`.

`1-9` jump to the corresponding database.

`/` search the database. `(ebib-search)`

`&` Create a virtual database, or perform a logical `and` on the current virtual database. With negative prefix argument: apply a logical `not` to the selectional criterion. `(ebib-virtual-db-and)`

`|` Create a virtual database, or perform a logical `or` on the current virtual database. With negative prefix argument: apply a logical `not` to the selectional criterion. `(ebib-virtual-db-or)`

`~` Perform a logical `not` on the current virtual database. (`ebib-virtual-db-not`)

`?` display Ebib info. (`ebib-info`)

`a` add an entry. (`ebib-add-entry`)

`b` equivalent to `Pgup`.

`c` close the database. (`ebib-close-database`)

`d` delete the current entry. (`ebib-delete-entry`)

`; d` delete all selected entries.

`e` edit the current entry. (`ebib-edit-entry`)

`E` edit the key of the current entry. (`ebib-edit-keyname`)

`f` extract a filename from the `file` field and send it to an appropriate viewer. With numeric prefix argument, extract the *n*-th filename.

`F` follow `crossref` field. (`ebib-follow-crossref`)

`g` equivalent to `Home`.

`G` equivalent to `End`.

`h` show the info node on the index buffer. (`ebib-index-help`)

`i` extract a DOI from the `doi` field, append it to a URL and open that in a browser (`ebib-browse-doi`)

`j` equivalent to `Down`.

`J` jump to another database. This accepts a numeric prefix argument, but will ask you for a database number if there is none. (`ebib-switch-to-database`)

`k` equivalent to `Up`.

`l` show the log buffer. (`ebib-show-log`)

`m` select (or unselect) the current entry. (`ebib-select-entry`)

`; m` unselect all selected entries.

`n` find next occurrence of the search string. (`ebib-search-next`)

`N` search for entries cross-referencing the current one. (`ebib-search-crossref`)

`C-n` equivalent to `Down`.

`M-n` equivalent to `PgDn`.

`o` open a `.bib` file. (`ebib-load-bibtex-file`)

`p` push an entry to a LaTeX buffer (`ebib-push-bibtex-key`)

`; p` push the selected entries to a LaTeX buffer.

`C-p` equivalent to `Up`.

`M-p` equivalent to PgUp.

`P` show and edit the `@preamble` definition in the database. `(ebib-edit-preamble)`

`q` quit Ebib. This sets all variables to nil, unloads the database(s) and quits Ebib. `(ebib-quit)`

`s` save the database. `(ebib-save-current-database)`

`S` show and edit the `@string` definitions in the database. `(ebib-edit-strings)`

`u` extract a URL from the `url` field and send it to a browser. With numeric prefix argument, extract the *n*-th url.

`V` Display the filter of the current virtual database in the minibuffer. With prefix argument: reapply the filter. `(ebib-print-filter)`

`x` export the current entry to a file, or, when used with numeric prefix argument, to another database. `(ebib-export-entry)`

`; x` export the selected entries to a file, or, when used with a numeric prefix argument, to another database.

`C-x b` equivalent to `z`.

`C-x k` equivalent to `q`.

`X` export the `@preamble` definition to a file or, when used with a numeric prefix argument, to another database. `(ebib-export-preamble)`

`z` move focus away from the Ebib windows. `(ebib-leave-ebib-windows)`

`Z` put Ebib in the background. `(ebib-lower)`

Functions not bound to any key:

- `ebib-print-filename`
- `ebib-customize`
- `ebib-merge-bibtex-file`
- `ebib-write-database`
- `ebib-save-all-databases`
- `ebib-print-entries`
- `ebib-latex-entries`
- `ebib-toggle-hidden`
- `ebib-toggle-timestamp`
- `ebib-toggle-identical-fields`
- `ebib-toggle-print-multiline`
- `ebib-toggle-layout`

The Entry Buffer

`Up` go to the previous field. (`ebib-prev-field`)

`Down` go to the next field. (`ebib-next-field`)

`PgUp` go to the previous set of fields. (`ebib-goto-prev-set`)

`PgDn` go to the next set of fields. (`ebib-goto-next-set`)

`Home` go to the first field. (`ebib-goto-first-field`)

`End` go to the last field. (`ebib-goto-last-field`)

`Space` equivalent to `PgDn`.

`b` equivalent to `PgUp`.

`c` copy the contents of the current field to the kill ring. (`ebib-copy-field-contents`)

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

`e` edit the current field. (`ebib-edit-fields`)

`f` extract a filename from the current field and send it to an appropriate viewer. With numeric prefix argument, extract the *n*-th filename.

`g` equivalent to `Home`.

`G` equivalent to `End`.

`h` show the info node on the entry buffer. (`ebib-entry-help`)

`j` go to the next field. (`ebib-next-field`)

`k` go to the previous field. (`ebib-prev-field`)

`l` edit the current field as multiline. (`ebib-edit-multiline-field`)

`C-n` equivalent to `Down`.

`M-n` equivalent to `PgDn`.

`C-p` equivalent to `Up`.

`M-p` equivalent to `PgUp`.

`q` quit editing the current entry and return focus to the index buffer. (`ebib-quit-entry-buffer`)

`r` toggle a field's "raw" status. (`ebib-toggle-raw`)

`s` insert an abbreviation from the `@string` definitions in the database. (`ebib-insert-abbreviation`)

`u` extract a URL from the current field and send it to a browser. With numeric prefix argument, extract the *n*-th url.

`x` cut the contents of the current field. Like `c`, `x` puts the contents of the current field in the kill ring. (`ebib-cut-field-contents`)

`y` yank the last element in the kill ring to the current field. Repeated use of `y` functions like `C-y/M-y`. Note that no text will be yanked if the field already has a value. (`ebib-yank-field-contents`)

The Strings Buffer

`Up` go to the previous string. (`ebib-prev-string`)

`Down` go to the next string. (`ebib-next-string`)

`PgUp` go ten strings up. (`ebib-strings-page-up`)

`PgDn` go ten strings down. (`ebib-strings-page-down`)

`Home` go to the first string. (`ebib-goto-first-string`)

`End` go to the last string. (`ebib-goto-last-string`)

`Space` equivalent to `PgDn`.

`a` add a new `@string` definition. (`ebib-add-string`)

`b` equivalent to `PgUp`.

`c` copy the text of the current string to the kill ring. (`ebib-copy-string-contents`)

`d` delete the current `@string` definition from the database. You will be asked for confirmation. (`ebib-delete-string`)

`e` edit the value of the current string. (`ebib-edit-string`)

`g` equivalent to `Home`.

`G` equivalent to `End`.

`h` show the info node on the strings buffer. (`ebib-strings-help`)

`j` equivalent to `Down`.

`k` equivalent to `Up`.

`l` edit the value of the current string as multiline. (`ebib-edit-multiline-string`)

`C-n` equivalent to `Down`.

`M-n` equivalent to `PgDn`.

`C-p` equivalent to `Up`.

`M-p` equivalent to `PgUp`.

`q` quit the strings buffer and return focus to the index buffer. (`ebib-quit-strings-buffer`)

`x` export the current `@string` definition to a file or, when used with a prefix argument, to another database. (`ebib-export-string`)

`X` export all the `@string` definitions to a file or, when used with a prefix argument, to another database. (`ebib-export-all-strings`)

Customisation

Ebib can be customised through Emacs' standard customisation interface. The only thing that cannot be customised in this way are the key bindings. If you wish to customise those, you have to use the file `~/.ebibrc`.

The Customisation Buffer

Ebib's customisation group is a subgroup of the `Tex` group. It can be invoked by typing `M-x customize-group RET ebib RET`, or going to the Options menu and selecting "Customize Ebib". This chapter gives a short description of all the options available in the customisation buffer.

Default Type

Default value: `article`.

The default type is the default entry type given to a new entry. Every entry in the Ebib database must have a type, because the type defines which fields are available. When a new entry is created, Ebib gives it a default type, which can be customised through this option.

Preload Bib Files

Default value: `nil`.

This option allows you to specify which file(s) Ebib is to load when it starts up. Specify one file per line, press the `INS` button to add more files. You can complete a partial filename with `M-TAB`.

Preload Bib Search Dirs

Default value: `~`.

This is a list of directories Ebib searches for `.bib` files to be preloaded. Note that only the directories themselves are searched, not their subdirectories.

Create Backups

Default value: `t`.

Whether to create backups of `.bib` files. You can unset this option if you have your files under revision control, for example.

Keywords list

Default value: `nil`.

List of keywords to offer for TAB completion when editing the `keyword` field.

Keywords File

Default value: `""`.

File to save keywords in. This can be a single file for all `.bib` files, or a separate keyword file for each directory containing `.bib` files. In the former case, provide a full filepath, in the latter case, a single filename without path.

Keywords File Save On Exit

Default value: `ask`

Whether to save keyword files on exit. Possible values are `ask` (ask whether to save), `always` (save automatically) and `never` (do not save).

Keywords Use Only File

Default value: `nil`.

If a keyword file is provided, should keywords be taken *only* from the file, or also from the keyword list.

Keywords Field Keep Sorted

Default value: `nil`.

When set, the `keywords` field is automatically sorted alphabetically when you add keywords to it. In addition, duplicates are automatically removed.

Keywords Separator

Default value: `"; "`

String to insert between keywords in the `keyword` field.

Additional Fields

Default value: `crossref url annote abstract keywords file timestamp`.

The additional fields are those fields that are available for all entry types, and which BibTeX generally ignores. This option allows you to specify which additional fields you wish to use in your database. Specify one field per line, press the `INS` button to add more fields.

Layout

Default value: `full`.

When set to `full`, Ebib takes over the entire frame when it runs. Alternatively, you can select `custom`, so that you can specify the width of the Ebib windows yourself. In this case, Ebib takes up the right part of the frame, leaving the left part free. See Screen Layout for details.

Width

Default value: 80.

The width of the Ebib windows when `ebib-layout` is set to `custom`.

Index Window Size

Default value: 10.

This option lets you specify the size of the index window at the top of the Ebib screen.

Index Display Fields

Default value: `nil`.

This option allows you to specify fields that should be displayed next to the entry key in the index buffer. By default, the index buffer only shows the key of each entry, but if this is too little information, you can use this option to display e.g. the title of each entry as well.

Uniquify Keys

Default value: `nil`.

When you add a new entry to the database or when Ebib autogenerates a key (when Autogenerate Keys is set), a check is made to make sure that the new key doesn't already exist in the database. If it does, Ebib normally aborts the operation but when this option is set, Ebib instead tries to create a unique key by appending `b` or `c`, etc. to it.

Autogenerate Keys

Default value: `nil`.

This option specifies whether Ebib should autogenerate keys or not. If set, adding a new entry will not ask for a key. Instead, a temporary key `<new-entry>` is used. When you leave the entry buffer, this key is replaced with an autogenerated key.

Ebib uses the function `bibtex-generate-autokey` to generate the key. This function has a number of customisation options, which are described in its documentation string.

Citation Commands

Default value:

```
((any
  (("cite" "\\cite%<[%A]%>{%K}")))
 (org-mode
  (("ebib" "[ebib:%K][%A]")))
 (markdown-mode
  (("text" "@%K%< [%A]%>")
   ("paren" "[%(%<%A %>@%K%<, %A%>%; )]")
   ("year" "[-@%K%< %A%>]"))))
```

With the command `ebib-insert-bibtex-key` or with the command key `p` in the index buffer, you can insert a BibTeX key into a LaTeX buffer. This option allows you to define the commands that are available through tab completion when these functions ask for a citation command.

Citation commands are defined for specific major modes or for all modes (under the heading `any`). Each command consists of an identifier, which you type when Ebib prompts you for a citation command, plus a format string. This format string can contain a few directives, which are used to add the citation key and any optional arguments. This format string can contain the following directives:

`%K` the entry key to be inserted.

`%A` an argument, for which the user is prompted.

`%<...%>` optional material surrounding a `%A` directive.

`%(...%<sep>)` a so-called *repeater*, which contains material that can be repeated. If present, the repeater must contain the entry key directive `%K`.

In the simplest case, the format string contains just a `%K` directive: `\cite{%K}`. In this case, `%K` is replaced with the citation key and the result inserted. Usually, however, citation commands allow for optional arguments that are formatted as pre- or postnotes to the citation. For example, using the `natbib` package, you have citation commands available of the form:

```
\citet[cf.][p. 50]{Jones1992}
```

In order to be able to insert such citations, the format string must contain `%A` directives:

```
\citet[%A][%A]{%K}
```

With such a format string, Ebib asks the user to provide text for the two arguments and inserts it at the locations specified by the directives. Of course, it is possible to leave the arguments empty (by just hitting `RET`). With the format string above, this would yield the following citation in the LaTeX buffer:

```
\citet[][]{Jones1992}
```

This is completely harmless, because LaTeX will simply ignore the empty arguments. However, you may prefer for the brackets not to appear if the arguments are empty. In that case, you can wrap the brackets and the `%A` directives in a `%<...%>` pair:

```
\citet%<[%A]%>%<[%A]%>{%K}
```

Now, if you leave the arguments empty, Ebib produces the following citation:

```
\citet{Jones1992}
```

Note however, that in this case there is in fact one problem associated with this format string. If you fill out the first argument but not the second, Ebib produces the wrong format string:

```
\citet[cf.]{Jones1992}
```

If only one optional argument is provided, `natbib` assumes that it is a postnote, while what you intended is actually a prenote. Therefore, it is best not to make the second argument optional:

```
\citet%<[%A]%>[%A]{%K}
```

This way, the second pair of brackets is always inserted, regardless of whether you provide a second argument or not.

`Natbib` commands also accept multiple citation keys. When you call `ebib-insert-bibtex-key` from within a LaTeX buffer, you can only provide one key, but when you're in Ebib, you can mark multiple entry keys and then use `; p` to push them to a buffer. In this case, Ebib asks you for a separator and then inserts all keys into the position of `%K`:

```
\citet{Jones1992,Haddock2004}
```

It is, however, also possible to specify in the format string that a certain sequence can be repeated and how the different elements should be separated. This is done by wrapping that portion of the format string that can be repeated in a `%(...%)` pair. Normally, you'll want to provide a separator, which is done by placing it between the `%` and the closing `)`:

```
\citet[%A][%A]{%(%K%,)}
```

This format string says that the directive `%K` can be repeated and that multiple keys must be separated with a comma. The advantage of this is that you are no longer asked to provide a separator. However, this is not the only thing one can do, because it is also possible to put `%A` directives in the repeating part. This is useful for the `biblatex` package. `Biblatex` has so-called *multicite* commands, which take the following form:

```
\footcites[cf.][p. 50]{Jones1992}[][p. 201]{Haddock2004}
```

Multicite commands can take more than one citation key in braces `{}` and each of those citation keys can take two optional arguments in brackets `[]`. In order to get such citations, you can provide the following format string:

```
\footcites%(%<[%A]%>[%A]{%K}%)
```

Here, the entire sequence of two optional arguments and the obligatory citation key is wrapped in `%(...%)`, so that Ebib knows it can be repeated. If you now mark multiple entries in Ebib, press `; p` and select the `footcites` command, Ebib will put all the keys in the citation, asking you for two arguments for each citation key.

Of course it is also possible to combine parts that are repeated with parts that are not repeated. In fact, that already happens in the previous example, because the part `\footcites` is not repeated. But the part that is not repeated may contain `%A` directives as well:

```
\footcites%<(%A)%>(%A)%(%<[%A]%>[%A]{%K}%)
```

Multicite commands in `biblatex` take two additional arguments surrounded with parentheses. These are pre- and postnotes for the entire sequence of citations. They can be accommodated as shown.

The function `ebib-insert-bibtex-key` can of course also be called from non-LaTeX buffers. For example, org-mode can use bibliographic links, and so can Pandoc markdown. For those two modes, citation commands have been predefined, which may provide some additional inspiration in case you want to create your own format strings.

Multiline Major Mode

Default value: `text-mode`.

This specifies the major mode used in the multiline edit buffer. Note that the value *must* be a command for a major mode.

Sort Order

Default value: `nil`.

The use of this option is explained above, see Sorting the .bib file. To create a sort order, click the `INS` button to create a sort level, and then click the `INS` button under that sort level to enter a sort field. If you want to add more than one sort field to the sort level, simply hit `INS` again.

Save Xrefs First

Default value: `nil`.

For cross-referencing to work, the cross-referencing entries must appear in the `.bib` file *before* the cross-referenced entries. In order to tell Ebib to save all entries with a `crossref` field first, you must set the option `Save Xrefs First` in Ebib's customisation buffer. With this option set, BibTeX's cross-referencing options work as intended.

Crossref Face

Default value: `((t (:foreground "red")))`.

Field values inherited from a cross-referenced entry are marked with this face. By default, the face has red as foreground colour.

Selected Face

Default value:
`((t (:inverse-video t)))`

When entries are selected (with `m`), they are highlighted in this face. By default, it uses the text property `highlight`.

Use Timestamp

Default value: `nil`.

If this option is set, Ebib will add a `timestamp` field to every new entry, recording the date and time it was added to the database. See Timestamps.

Timestamp Format

Default value: `"%a %b %e %T %Y"`.

This option specifies the format string that is used to create the timestamp. The format string is used by `format-time-string` to create a time representation. The default value produces a timestamp of the form `"Mon Mar 12 01:03:26 2007"`. See the documentation for the Emacs function `format-time-string` for the forms that the format string can take.

Standard Url Field

Default value: `url`.

This is the field that Ebib searches for URLs if you press `u` in the index buffer.

Url Regexp

Default value: `\\url{\(.*\)}\|https?://[^ '<>\"\n\t\f]+`.

This is the regular expression that Ebib uses to search for URLs in a field. With the default value, Ebib considers everything that is in a LaTeX `\url{...}` command as a URL, and furthermore every string of text that starts with `http://` or `https://` and does not contain whitespace or one of the characters `' "` < or >.

Browser Command

Default value: `nil`.

If this option is unset, Ebib uses the Emacs function `browse-url` to start a browser. If this function does not exist, you can set this option. For example, if you use the Firefox browser, set this option to `firefox`.

For this to work, the browser that you use must be able to handle a URL on the command line.

Standard Doi Field

Default value: `doi`.

The field Ebib uses to extract the DOI if you press `i` in the index buffer.

Doi Url

Default value: `http://dx.doi.org/%s`

The URL used to open a DOI. It must contain exactly one instance fo `%s`, which will be replaced with the DOI.

Standard File Field

Default value: `file`.

This is the field that Ebib searches for filenames if you press `f` in the index buffer.

File Associations

Default value: `(("pdf" . "xpdf") ("ps" . "gv"))`.

The programs used to view files. By default, programs for `.pdf` and `.ps` files are specified, which should be available on most linux systems. If you prefer other programs or are running on Windows, you can specify them here. Note that Ebib searches the PATH for the programs, but you can also specify full path names. Of course, it is also possible to add new associations.

Note that GNU/Emacs 23 comes with `doc-view-mode`, which provides a way to view `.pdf` and `.ps` files inside Emacs. (The files are converted to `.png` format first.) If you prefer to use this mode, simply leave the program field blank for the relevant file type.

File Regexp

Default value: `[^?|\:*<>\" \n\t\f]+`.

Regural expression used to find files in a field. The default value essentially means that every string of characters not containing any of the characters `? | \ : * < > "` or space, newline, tab of formfeed is recognised as a file name.

Note that URLs can easily by recognised by the prefix `http:`, but recognising files is not so straightforward. It is therefore not advisable to put anything but filenames in the `file` field.

File Search Dirs

Default value: `~`.

List of directories that Ebib searches for files. Note that searching is not recursive: only the files listed here are searched, not their subdirectories.

Print Preamble

Default value: `nil`.

This option specifies the preamble that is to be added to the LaTeX file Ebib creates for printing the database as index cards. You can set your own `\usepackage` commands, or anything else you may need.

Print Newline

Default value: `nil`.

With this option set, Ebib puts every entry on a separate page when printing index cards. When this option is unset, the entries are separated by a small amount of whitespace only.

Print Multiline

Default value: `nil`.

When this options is set, Ebib includes multiline field values when you print index cards. When unset, multiline values are excluded, which saves space.

Latex Preamble

Default value: `\bibliographystyle{plain}`.

This option specifies the preamble to be added to the LaTeX file for creating a list of references from the database. The default is to use the `plain` style, but you may want to specify your own BibTeX packages and options.

Print Tempfile

Default value: `nil`.

This option specifies the name and location of the temporary file Ebib creates with the commands `ebib-print-database` and `ebib-latex-database`. When unset, Ebib will ask for a filename each time either of these commands is called.

Allow Identical Fields

Default value: `nil`.

If this option is set, Ebib stores the values of multiple occurrences of a single field within an entry in a single occurrence of that field, separated by semicolons (see Multiple Identical Fields).

Entry Types

Default value: see below.

This option allows you to customise the entry types that Ebib uses. Each entry type has a name, a set of obligatory fields and a set of optional fields. You can add, alter or delete single fields in an entry type, or whole entry types.

If you want to add an entry type, hit the `INS` key on the top level and give the new entry a name, then add obligatory and/or optional fields. It is not necessary that an entry type has both obligatory and optional fields, you can define an entry that has only obligatory or only optional fields.

The default entry types and fields are the following:

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

book
  author title publisher year
  editor volume number series address edition month note

booklet
  title
  author howpublished address month year note

inbook
  author title chapter pages publisher year
  editor volume series address edition month note

incollection
  author title booktitle publisher year
  editor volume number series type chapter pages address edition month note

inproceedings
  author title booktitle year
  editor pages organization publisher address month note

manual
  title
```

```
  author organization address edition month year note

misc
  --
  title author howpublished month year note

mastersthesis
  author title school year
  address month note

phdthesis
  author title school year
  address month note

proceedings
  title year
  editor publisher organization address month note

techreport
  author title institution year
  type number address month note

unpublished
  author title note
  month year
```

Modifying Key Bindings

If you are unhappy about Ebib's standard key bindings and would like to change them, or if you would like to bind a command that is only available through the menu to a key, you can do so by creating a file `~/.ebibrc` and writing your preferred key bindings in it. A key binding definition is built up as follows:

```
(ebib-key <buffer> <key> <command>)
```

`<buffer>` is either `index`, `entry` or `strings`, for the corresponding buffer. `<key>` is a standard Emacs key description, and `<command>` is the Ebib command to be associated with the key. The commands that can be used here are listed in The Ebib Buffers. Note that it is possible to bind more than one key to a single function: just add as many `ebib-key` statements as necessary.

As an example, the following binds `C-s` to `ebib-search` in the index buffer, so that the database can be searched with `C-s` as well as with `/`:

```
(ebib-key index "\C-s" ebib-search)
```

If you want to unbind a key, you can simply leave out `<command>`. So if you want to bind the command `ebib-delete-entry` to `D` rather than `d`, you need to put the following in `.ebibrc`:

```
(ebib-key index "D" ebib-delete-entry)
(ebib-key index "d")
```

The first line binds `D` to the command `ebib-delete-entry`. The second line unbinds `d`.

If a command can be called with a prefix key (as for example `ebib-delete-entry` can), `ebib-key` will automatically rebind the prefixed version as well. So in the example above, the first line not only binds `D`, it also binds `; D`. Similarly, the second line not only unbinds `d`, but also `; d`.

Note that if you bind the print commands to a key (`ebib-print-entries` and `ebib-latex-entries`) they are automatically set up to accept the prefix key as well.

It is also possible to redefine the prefix key itself. To do this, you must specify `select-prefix` for `<buffer>`. The value of `<command>` is irrelevant here, so it can be left out:

```
(ebib-key select-prefix ":")
```

This sets up `:` as the new prefix key. Doing this automatically unbinds the existing prefix key.

As a final option, `ebib-key` allows you to redefine the second key in the key bindings of the multiline edit buffer:

```
(ebib-key multiline "&")
```

This piece of code changes the commands `C-c | q`, `C-c | c` and `C-c | s` to `C-c & q`, `C-c & c` and `C-c & s`, respectively.