
BIBTOOL

A Tool to Manipulate BibTeX Files

Version 2.41

Gerd Neugebauer

Abstract

BibTeX provides an easy to use means to integrate citations and bibliographies into L^AT_EX documents. But the user is left alone with the management of the BibTeX files. The program BIBTOOL is intended to fill this gap. BIBTOOL allows the manipulation of BibTeX files which goes beyond the possibilities — and intentions — of BibTeX. The possibilities of BIBTOOL include sorting and merging of BibTeX data bases, generation of uniform reference keys, and selecting of references used in a publication.

This file is part of BIBTOOL Version 2.41

Copyright ©1997 Gerd Neugebauer

BIBTOOL is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 1, or (at your option) any later version.

BIBTOOL is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this documentation; see the file COPYING. If not, write to the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.

Gerd Neugebauer

Mainzer Str. 8

56321 Rhens (Germany)

WWW: <http://www.uni-koblenz.de/~gerd>

Net: gerd@informatik.uni-koblenz.de

gerd@imn.th-leipzig.de

gerd@intellektik.informatik.th-darmstadt.de

Contents

1	Introduction	5
1.1	Related Programs	5
1.2	Using BIBTOOL — Some Instructive Examples	6
1.2.1	Sorting and Merging	7
1.2.2	Key Generation	7
1.2.3	Normalization	9
1.2.4	Extracting Entries for a Document	10
1.2.5	Extracting Entries Matching a Regular Expression	10
1.2.6	Translating ISO 8859-1 Characters	11
1.3	Interfacing BIBTOOL with Other Programming Languages	11
1.4	Getting BIBTOOL, Hot News, and Bug Reports	11
1.5	Contributing to BIBTOOL	12
A	Reference Manual	13
A.1	Command Line Usage and Resource Files	13
A.2	Input File Specification and Search Path	16
A.3	Output File Specification and Status Reporting	18
A.4	Parsing and Pretty Printing	18
A.5	Sorting	22
A.6	Regular Expression Matching	23
A.7	Selecting Items	24
A.7.1	Extracting by <code>aux</code> Files	24
A.7.2	Extracting with Regular Expressions	25
A.8	Key Generation	26
A.9	Format Specification	31
A.9.1	Constant Parts	31
A.9.2	Formatting Fields	32
A.9.3	Conjunctions	40
A.9.4	If-Then-Else	40
A.9.5	Alternatives	40
A.9.6	Grouping	41
A.9.7	Expanding \TeX / \LaTeX Macros	41
A.9.8	Name Formatting	42

A.9.9 Example	43
A.10 Field Manipulation	45
A.10.1 Adding or Deleting Fields	45
A.10.2 Field Rewriting	46
A.10.3 Field Ordering	48
A.11 Semantic Checks	49
A.11.1 Finding Double Entries	49
A.11.2 Regular Expression Checks	50
A.12 Strings — also called Macros	51
A.13 Statistics	52
B Limitations	53
B.1 Limits of BIBTOOL	53
B.2 Bugs and Problems	53
C Sample Resource Files	55
C.1 The Default Settings	55
C.2 Useful Translations	56
C.3 Other Resource Files	57

1

Introduction

1.1 Related Programs

`BIBTEX` [Lam94, Pat88a, Pat88b] is a system for integrating bibliographic information into `LATEX` [Lam94] documents. `BIBTEX` is designed to serve exactly this purpose. It has shown that various tasks in relation with managing bibliographic databases are not covered by `BIBTEX`. Usual activities on bibliographic databases include

- inserting new entries
- editing
- using citations in documents
- sorting and merging of bibliographic data bases
- extraction of bibliographic data bases

Since only the integration in documents is covered by `BIBTEX` several utilities emerged to fill the gaps. We will sketch some of them shortly.

`BIBTEX` is a program by Oren Patashnik to select publications used in a `LATEX` document and format them for inclusion into this document. This program should be part of each `TEX` installation.

`bibclean` is a program by Nelson H.F. Beebe to pretty-print `BIBTEX` files. It also can act as syntax checker. The C sources can be compiled on several systems.

`bibindex/biblook` is a pair of programs by Nelson H.F. Beebe to generate an index for a `BIBTEX` file and use it to perform a fast lookup of certain entries. The programs so far run only under UNIX.

bibsort is a UNIX shell script by Nelson H.F. Beebe to sort a $\text{BIB}_{\text{T}}\text{E}_{\text{X}}$ file.

bibclean is a UNIX shell script by Nelson H.F. Beebe to extract entries from a $\text{BIB}_{\text{T}}\text{E}_{\text{X}}$ file which are used in a $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ document.

lookbibtex/bibdestringify are Perl scripts by John Heidemann to extract entries from a $\text{BIB}_{\text{T}}\text{E}_{\text{X}}$ file which are used in a $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ document and to remove strings from a $\text{BIB}_{\text{T}}\text{E}_{\text{X}}$ file.

bibtools is a collection of UNIX shell scripts by David Kotz to add and extract entries to bibliographic databases. Several small programs are provided to perform special tasks.

bibview is a Perl script by Dana Jacobsen to extract entries from a $\text{BIB}_{\text{T}}\text{E}_{\text{X}}$ file which are used in a $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ document.

BibCard is a program by William C. Ogden running under X11/xview which provides a means to edit bibliographic databases.

hyperbibtex Something similar for Macintosh computers.

xbibtex/bibprocess/bibsearch are programs by Nicholas J. Kelly and Christian H. Bischof running under X11 which provides a means to edit bibliographic databases, add fields to a $\text{BIB}_{\text{T}}\text{E}_{\text{X}}$ file and extract certain entries from a $\text{BIB}_{\text{T}}\text{E}_{\text{X}}$ file.

bibview is an X11 program by Holger Martin, Peter Urban, and Armin Liebl to search in and manipulate $\text{BIB}_{\text{T}}\text{E}_{\text{X}}$ files. It is similar to BibCard and hyperbibtex.

$\text{BIB}_{\text{T}}\text{E}_{\text{X}}$ -Mode is an extension of the editor GNU-Emacs to provide means to edit $\text{BIB}_{\text{T}}\text{E}_{\text{X}}$ files. Several useful operations are provided.

This is a selection of some programs I have heard of. I have tested some of them and I have skipped through the documentation of others. Thus the description may be too short or incomplete. Some additional information can be found in [GMS94, Chapter 13].

Most of those utilities are tailored towards a particular operating system and thus they are not available on other platforms. Most of these program are made to perform a single task. Often they can not be configured to suit a personal taste of a user.

Still there are some points not covered by the utilities mentioned above. **BIBTOOL** tries to provide the missing features and integrate others into a single tool.

1.2 Using **BIBTOOL** — Some Instructive Examples

BIBTOOL has been developed on UN*X and UN*X-like machines. This has influenced many of the design decisions. Version 1 was controlled using numerous command line options. This way of controlling has been supplemented in version 2 by the concept of a resource file. This resource file allows the modification of the various internal parameters determining the behavior of **BIBTOOL**.

When BIBTOOL has been compiled correctly there should be an executable file named `bibtool`¹. We will assume that you are running BIBTOOL from a command line interpreter. There you can simply issue the command

```
bibtool
```

Now BIBTOOL will start reading from the standard input lines obeying the rules of a BIBTEX file.² The entries read are pretty-printed on the standard output. It is obvious that this behavior is not very useful in itself. The origin of this kind of interface lies in the concepts of UN*X where many commands can act as filters.

Usually we do not intend to use BIBTOOL in this way. Thus we need a way to specify an input file. This is simply done by adding the file name as argument after the command name like in

```
bibtool file.bib
```

The result of this command can at once be seen on the screen. The contents of the file `file.bib` is pretty printed.

Now that we have seen the simplest case of the application of BIBTOOL we will see the case of a useful application of BIBTOOL. This application is the sorting and merging of BIBTEX databases.

1.2.1 Sorting and Merging

BIBTEX files can be sorted by specifying the command line option `-s`. The given files are sorted according to the reference key. Several files can be given at once in which case BIBTOOL will sort and merge those files.

```
bibtool -s file1.bib file2.bib
```

With the command line option the files are sorted in reverse ASCII order.

```
bibtool -S file1.bib file2.bib
```

If you want to sort the BIBTEX files according to the authors then the following invocation should do the trick:

```
bibtool -- sort.format="%N(author)" file1.bib file2.bib
```

This means that the sorting order is determined by the (normalized) author field.

1.2.2 Key Generation

Once you have a reference and you insert it into a BIBTEX file you have to assign a reference key to it. The problem is to find a key which is unique and meaningful, i.e. easy to remember. The easiest way to remember a key is to use an algorithm to create it and remember the algorithm—which is the same for all keys.

One algorithm which comes to mind is to use the author and (an initial part) of the title. Alternatively we can use the author and the year. But the problem is with industrious authors writing more than one publication per year. The necessary disambiguation of such

¹Maybe with an additional extension.

²We assume that no resource file can be found. Resource files will be described later.

references is not very intuitive. However, BIBTOOL has the capability to describe desired keys. Thus, the alternatives described above can be realized.

For this section we want to use the following BIB_TE_X entry as our example:³ Suppose it is contained in a file named `sample.bib`.

```
@ARTICLE{article-full,
  author = {L[eslie] A. Aamport},
  title = {The Gnats and Gnus Document Preparation System},
  journal = {\mbox{G-Animal's} Journal},
  year = 1986,
  volume = 41,
  number = 7,
  pages = "73+",
  month = jul,
  note = "This is a full ARTICLE entry",
}
```

First, we want to see how we can make keys consisting of author and title. This is one of my favorite algorithms thus it is rather easy to use it. You simply have to run the command

```
bibtool -k sample.bib -o sample1.bib
```

Afterwards the following entry can be found in the file `sample1.bib`:

```
@Article{      aamport:gnats,
  author       = {L[eslie] A. Aamport},
  title        = {The Gnats and Gnus Document Preparation System},
  journal       = {\mbox{G-Animal's} Journal},
  year         = 1986,
  volume       = 41,
  number       = 7,
  pages        = "73+",
  month        = jul,
  note         = "This is a full ARTICLE entry"
}
```

You see that the reference key has been changed. It now consists of the last name and the first relevant word of the title, separated by a colon. Sometimes it might be desirable to incorporate the initial names as well. This can be achieved by the command

```
bibtool -K sample.bib -o sample1.bib
```

The resulting reference key is `aamport.la:gnats`. The initials are appended after the first name. Thus the usual lexicographic order on the keys will (hopefully) bring together the publications of the same first author.

Another alternative is to use the author and the year. This can be achieved with the following command:⁴

```
bibtool -f %n(author):%2d(year) sample.bib -o sample1.bib
```

³Shamelessly stolen from the BIB_TE_X `xamples.bib` file.

⁴Note that some command line interpreters (like the UN*X shells) require the format string to be quoted (enclosed in single quotes).

The resulting key is `Aairport:86`. Note that the last example works as desired for our sample file. But for a real application of this technique a deep understanding of the key generation mechanism as described in section A.8 is necessary.

1.2.3 Normalization

BIBTOOL can be used to normalize the appearance of BIB_TE_X databases. As an example we can consider the different forms of delimiters for fields. BIB_TE_X allows the use of braces or double quotes. Now it can be desirable to use one style only. For this purpose the rewriting facility of BIBTOOL can be applied.

```
bibttool -- 'rewrite.rule={"^\\"([^\#]*\)\\"$" "{\1}"}' -o out.bib
```

Since this seems to be rather cryptic we will have a closer look at this example. First we have to mention that the outer quotes are there because the UN*X shell (csh) treats some characters special and we want to avoid this to happen to the rewrite rule given. A similar quoting mechanism might be required for all command line interpreters.

The rewrite rule is applied to any field. The first string—called pattern—which is enclosed in double quotes is matched against the contents of the field. If a match is found then the matching substring is replaced by the replacement text in the second string.

The pattern is a regular expression like the ones used in Emacs. The first character is the hat (^). This character anchors the match at the beginning of the line. The last character is the dollar sign which anchors the end at the end of the field value. Thus only complete matches are considered.

Since we want to find those fields whose values are enclosed in double quotes they are given after the hat and before the dollar. To avoid a misinterpretation as the end of the pattern they have to be quoted with the backslash (\).

Next we have the parentheses \(. . \). They are instructions to memorize the matching substring in a register. Since it is the first instruction of this kind the register number 1 is used.

Now we come to the point where we have to specify the contents of the string. For this purpose we use a character class—written as [...]. Since the first character in this class specification is a hat this class consists of all characters but those given after the hat. Thus all characters but the hash sign (#) are allowed.

The star (*) after the character class indicates that an arbitrary number of characters of this class are allowed.

We have used the complicated construction with a character class to avoid wrong results which would have resulted when this rewrite rule is applied to a concatenated field value like the following one:

```
author = "A. U. Thor" # " and "# "S. O. Meone"
```

Such fields are left unchanged by the rewrite rule given above. We could have used the point (.) instead of the character class since the point matches any character. But this would have led to the syntactic wrong result:

```
author = {A. U. Thor" # " and "# "S. O. Meone}
```

But we have to complete the explanation of the rewrite rule. The remaining part is the replacement text. Here we just have to note that the substring `\1` is not copied verbatim but replaced with the contents of the first register. This register contains the contents of the field without the delimiting double quotes.

Thus we have a solution to our initial problem which is conservative in the sense that it sometimes fails but never produces a wrong result.

1.2.4 Extracting Entries for a Document

BIBTOOL can be used to extract the references used in a document. For this purpose BIBTOOL analyzes the `.aux` file and takes the information given there. This includes the names of the `BIBTEX` files. Thus no `BIBTEX` files have to be given in the command line. Instead the `.aux` file has to be specified — preceded by the option `-x`.

```
bibtool -x document.aux -o document.bib
```

The second option `-o` followed by a file name specifies the destination of the output. This means, instead of writing the result to the standard output stream the result is written into this file.

1.2.5 Extracting Entries Matching a Regular Expression

BIBTOOL can be used to extract the references which fulfill certain criteria. Those criteria can be specified utilizing regular expressions.⁵ As a special case we can extract all entries containing a certain substring of the key:

```
bibtool -X tex all.bib -o some.bib
```

This instruction selects all entries containing the substring `tex` in the key. The second option `-o` followed by a file name specifies the destination of the output. Thus instead of writing the result to the standard output stream the result is written into this file.

Next we want to look up all entries containing a substring in some of its fields. For this purpose we search for the string in all fields first:⁶

```
bibtool -- select{"tex"} all.bib -o some.bib
```

Note that the comparison is not done case sensitive; however this can be customized (see page 25).

Finally we want to select only those entries containing the substring in any one of certain fields. For this purpose we simply specify the names of those fields in the `select` instruction:

```
bibtool -- select{title booktitle $key "tex"} all.bib -o some.bib
```

This example extracts all entries containing the substring `tex` in the title field, the booktitle field, or the reference key.

After we have come so far we can say that the first example in this section is in fact a short version of the following command:

⁵Those features are only usable if the regular expression library has been enabled during the configuration of BIBTOOL—which is the default.

⁶Note that some command line interpreters (e.g the UN*X shells) might need additional quoting of the `select` instruction since it contains special characters.

```
bibttool -- select{$key "tex"} all.bib -o some.bib
```

As a simple case of extraction we might want to extract all books from a bibliography. This can be done with the following command:

```
bibttool -- select{book} all.bib -o some.bib
```

A similar method can also be applied for other entry types.

1.2.6 Translating ISO 8859-1 Characters

Sometimes you need to translate some special characters into $\text{BIB}_{\text{T}}\text{E}_{\text{X}}$ sequences. Suppose you have edited a $\text{BIB}_{\text{T}}\text{E}_{\text{X}}$ file and by mistake used those nice characters that are incompatible with standard ASCII as used in $\text{BIB}_{\text{T}}\text{E}_{\text{X}}$. You can use $\text{BIB}_{\text{T}}\text{OOL}$ to do the trick:

```
bibttool -r iso2tex -i iso.bib -o ascii.bib
```

1.3 Interfacing $\text{BIB}_{\text{T}}\text{OOL}$ with Other Programming Languages

$\text{BIB}_{\text{T}}\text{OOL}$ can be used as a means for other programming languages to access $\text{BIB}_{\text{T}}\text{E}_{\text{X}}$ data bases. In this course $\text{BIB}_{\text{T}}\text{OOL}$ reads the $\text{BIB}_{\text{T}}\text{E}_{\text{X}}$ file and prints it in a normalized form which makes it easy for the host programming language to parse it and get the information about the entries and fields.

In addition $\text{BIB}_{\text{T}}\text{OOL}$ can already preselect several entries or do other useful transformations before the host programming language even sees the contents. Thus it is fairly easy to write a CGI script (e.g. in Perl) which utilizes $\text{BIB}_{\text{T}}\text{OOL}$ to extract certain entries from a $\text{BIB}_{\text{T}}\text{E}_{\text{X}}$ data base and presents the result on a HTML page.

Currently the distribution of $\text{BIB}_{\text{T}}\text{OOL}$ contains frames of programs in Perl and Tcl which can be used as a basis for further developments.

1.4 Getting $\text{BIB}_{\text{T}}\text{OOL}$, Hot News, and Bug Reports

Usually $\text{BIB}_{\text{T}}\text{OOL}$ can be found on the CTAN or one of its mirrors. Thus you can get $\text{BIB}_{\text{T}}\text{OOL}$ via ftp or extract it from a CDROM containing a dump of the CTAN. The CTAN (Comprehensive $\text{T}_{\text{E}}\text{X}$ Archive Network) consists of the following sites:

```
ftp.dante.de (Germany)
ftp.tex.ac.uk (UK)
ftp.cdrom.com (US mirror for shsu.edu)
```

$\text{BIB}_{\text{T}}\text{OOL}$ can be found in the following directory:

```
tex-archive/biblio/bibtex/utils/bibttool
```

I have set up a WWW page for BibTool. It contains a short description of the features and links to the documentation and the current downloadable version in source form. The URL is:

```
http://www.uni-koblenz.de/~gerd/ftp/BibTool
```

In addition, this page contains a description of the current version of BIBTOOL and a list of changes in the last few releases.

If you encounter problems installing or using BIBTOOL you can send me a bug report to my email address `gerd@informatik.uni-koblenz.de`. Please include the following information into a bug report:

- The version of BIBTOOL you are using.
- Your hardware specification and operating system version.
- The C compiler you are using and its version. (Only for compilation and installation problems)
- The resource file you are using. Try to reduce it to the absolute minimum necessary for demonstrating the problem.
- A *small* BIBTEX file showing the problem.
- The command line options of an invocation of BIBTOOL making the problem appear.
- A short justification why you think that the behavior is an error.

I have had the experience that compiling this information has helped me find my own problems in using software. Thus I could fix several problems before sending a bug report.

On the other side I have unfortunately also had the experience that I have got complains about problems in my software. After several questions it turned out that the program had not been used correctly.

Oh, sure. There have been bugs and I suppose there are still some bugs in BIBTOOL. I am grateful for each hint which helps me in eliminating these bugs.

1.5 Contributing to BIBTOOL

As you might have read BIBTOOL is free software in the sense of the Free Software Foundation. This means that you can use it as a whole or parts of it as long as you do not deny anyone to have the sources and use it freely⁷

If you feel morally obliged to provide compensation for the use of this program I have the following suggestions.

- Proofread this documentation and report any errors you find as well as additional material to put in.
- Provide additional contributed pieces to BIBTOOL. E.g. useful resource files which could be included into the library.
- Write a useful program and release it to the public without making profit, preferably under the GNU general public license or the GNU artistic license.

⁷See the file COPYING for details.

A

Reference Manual

This part of the documentation tries to describe all commands and options. Since the behavior of BIBTOOL can be adjusted at compile time not all features may be present in your executable. Thus watch out and complain to the *installer* if something is missing.

A.1 Command Line Usage and Resource Files

BIBTOOL can be controlled either by arguments given in the command line or by commands given in a file (or both). Those command files are called resource files. If BIBTOOL is installed correctly you should have the executable command `bibtool` (maybe with an additional extension). Depending on your computer and operating system you can start BIBTOOL in different ways. This can be done either by issuing a command in a command line interpreter (shell), by clicking an icon, or by selecting a menu item. In the following description we will concentrate on the use in a UN*X like shell. There you can type simply

```
bibtool
```

Now BIBTOOL is waiting for your input. As you type BIBTOOL reads what you type. This input is interpreted as data conforming Bib_TE_X file rules. The result is printed when BIBTOOL is finished. You can terminate the reading phase with your End-Of-File character (e.g. Control-C on UN*X, or Control-Z on MS-D*S)

This application in itself is rather uninteresting. Thus we come to the possibility to give arguments to BIBTOOL. The simplest argument is `-h` as in

```
bibtool -h
```

This command should print the version number and a short description of the command line arguments to the screen.

The next application is the specification of resources. Resource files can be given in the command line after the flag `-r`.

```
bibtool -r resource_file
```

In this way an arbitrary number of resource files can be given. Those resource files are read in turn and the commands contained are evaluated. If no resource file is given in the command line BIBTOOL tries to find one in standard places. First of all the environment variable BIBTOOLRSC is searched. If it is defined then the value is taken as a resource file name. Otherwise the default resource file (usually the file `.bibtoolrsc`) is tried to be read in the home directory (determined by the environment variable HOME) or the current directory.

The resource files are searched similar to the searching mechanism for BIBTEX files (see section A.2). The extension `.rsc` is tried and a search path can be used. This search path is initialized from the environment variable BIBTOOL. Initially only the current directory is on the search path. The search path can also be set in a resource file (for following resource file reading). This can be achieved by setting the resource `resource.search.path`.

```
resource.search.path = path
```

When an explicit resource file is given in the command line the defaults are not used. To incorporate the default resource searching mechanism the command line option `-R` can be used:

```
bibtool -R
```

Now let us consider some examples. Suppose that the current directory contains a default resource file (named `.bibtoolrsc`) and an additional resource file `my_rsc`.

The following invocation of BIBTOOL uses only the resource file `my_rsc`:

```
bibtool -r my_rsc -i sample
```

If you want to initialize the resources from the default resource file before you can use the `-R` *before* the inclusion of the resource file:

```
bibtool -R -r my_rsc -i sample
```

If you add the `-R` argument after the resource specification then the default resource is evaluated after your resource file. Thus settings are potentially overwritten:

```
bibtool -r my_rsc -R -i sample
```

Additionally note that resource files are evaluated at once whereas input files are read in one chunk at the end. Thus you can not specify one set of parameters to be used for one file and another set of parameters for the next file. This is impossible within one invocation of BIBTOOL¹

As a consequence of this behavior the last example is equivalent to the following invocations:

```
bibtool -r my_rsc -i sample -R  
bibtool -i sample -r my_rsc -R
```

Now we have to describe the commands allowed in a resource file. The general form of a resource command is of the form

```
name = value
```

¹This might be changed in the next major revision (3.0).

`name` is the resource name which conforms the rules of `BiBTeX` reference keys. Thus `name` can be composed of all characters but whitespace characters and

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

Resource names are currently composed of letters and the period. The next component is an optional equality sign (`=`). The equality sign recommended as it helps detecting syntax problems. Whitespace characters surrounding the equality sign or separating resource name and resource value are ignored. The resource value can be of the following kind:

- A number composed of digits only.
- A string conforming the rules of resource names, i.e. made up of all but the forbidden characters described above.
- A string containing arbitrary characters delimited by double quotes (`"`) not containing double quotes. Parentheses and curly brackets have to come in matching pairs.
- A string containing arbitrary characters delimited by curly brackets (`{}`). Parentheses and curly brackets have to come in matching pairs.

You can think of resource names as variables or functions in a programming language. Resource commands simply set the variables to the given value, add the value to the old value, or initiate a action. There are different types of resources

- Boolean resources can take only the values `on` and `off`. `on`, `t`, `true`, `1`, and `yes` are interpreted the same. Everything else is interpreted as `off`.
- Numeric resources can take numeric values only.
- String resources can take arbitrary strings.

Usually whitespace characters are ignored. There is one exception. The characters `%` and `#` act as comment start characters if given between resource commands. All characters to the end of the line are ignored afterwards.

Now we come the description of the first resource available. To read in additional resource files the resource file may contain the resource

`resource = {additional/resource/file}`

Thus the resource given above has the same functionality as the command line option `-r` described above. Path names should be specified in the normal manner for your operating system.

One resource command useful for debugging is the `print` resource. The resource value is immediately written to the error stream. The output is terminated by a newline character. Certain translations are performed during the reading of a resource which can be observed when printing. Each sequence of whitespace characters is translated into a single space.

To end this subsection we give an example of the `print` resource. In this sample we also see the possibility the omit the equality sign and use quotes as delimiters.

```
print "This is a stupid message."
```

Finally we can note that the commands given in a resource file can also be specified on the command line. This can be achieved with the command line option `--`. The next command line argument is taken as a resource command.

```
bibtool -- resource_command
```

This can be used to issue resource command which do not have a command line counterpart. One example we have already seen. The `print` instruction can be used from the command line with the following

```
bibtool -- print{hello_world}
```

Be aware that command line interpreters have different ideas about what to do with a command line before passing the arguments to a program. Thus it might be necessary to carefully quote the arguments. Especially if the command contains spaces it is very likely that quoting is needed.

E.g. in UN*X shells it is in general a good strategy to enclose command line arguments in single quotes (') if they contain whitespace or special characters like `\`, `$`, `&`, `!`, or `#`.

Summary

<code>-h</code>		Show a list of command line options.
<code>-R</code>		Immediately evaluate the instructions from the default file.
	<code>print {message}</code>	Write out the text <i>message</i> .
<code>-r file</code>	<code>resource = file</code>	Immediately evaluate the instructions from the resource file <i>file</i> .
	<code>resource.search.path</code>	List of directories to search for resource files.
<code>-- rsc</code>	<code>rsc</code>	Evaluate the resource instruction <i>rsc</i> .

A.2 Input File Specification and Search Path

An arbitrary number of input files can be specified. Input files can be specified in two ways. The command line option `-i` is immediately followed by a file name.

```
bibtool -i input_file
```

The resource name input can be used to specify additional input files.

```
input = {input_file}
```

Input files are processed in the order they are given. If no input file is specified the standard input is used to read from.

Depending on the special configuration of BIBTOOL there are two ways of searching for BIB_TE_X files. The native mode of BIBTOOL uses a list of directories and a list of extensions to find a file. Alternatively the `kpathsea` library can be used which provides additional features like the recursive searching in subdirectories. First we look at the native BIBTOOL searching mechanism.

The files are searched in the following way. The file is can't be opened as given then the extension `.bib` is appended and another read is tried. In addition directories can be given which are searched for input files. The search path can be given in two different ways. First, the resource name `bibtex.search.path` can be set to contain a search path specification.

$$\text{bibtex.search.path} = \{\text{directory1:directory2:directory3}\}$$

The elements of the search path are separated by colons. Thus colons are not allowed as parts of directories. Another source of the search path is the environment variable `BIBINPUTS`. This environment variable is usually used by `BIBTEX` to specify the search path. The syntax of the specification is the same as for the resource `bibtex.search.path`. To check the appropriate way to set your environment variable consult the documentation of your shell, since this is highly dependent on it.

To allow adaption to operating systems other than `UN*X` the following resources can be used. The name of the environment `bibtex.env.name` overwrites the name of the environment variable which defaults to `BIBINPUTS`.

$$\text{bibtex.env.name} = \{\text{ENVIRONMENT_VARIABLE}\}$$

The first character of the resource `env.separator` is used as separator of directories in the resource `bibtex.search.path` and the environment variable given as `bibtex.env.name`.

$$\text{env.separator} = \{:\}$$

The default character separating directories in a file name is the slash (`/`). The first character of the resource `dir.file.separator` can be used to change this value.

$$\text{dir.file.separator} = \{\backslash\}$$

Note that the defaults for `env.separator` and `dir.file.separator` are set at compile time to a value suitable for the operating system. Usually you don't have to change them at all. E.g. for `MSD*S` machines the `env.separator` is usually set to `;` and the `dir.file.separator` is usually set to `\`.

If the `kpathsea` library is used for searching `BIBTEX` files then some of the resources described above have no effect. They are replaced by their `kpathsea` counterparts. Most probably you are using the `kpathsea` library already in other `TEX` related programs. Thus I just have to direct you to the documentation distributed with the `kpathsea` library for details.

Summary

	<code>bibtex.env.name={var}</code>	Use the environment variable <i>env</i> to add more directories to the search path for <code>BIBTEX</code> (input) files.
	<code>bibtex.search.path={path}</code>	Use the list of directories <i>path</i> to find <code>BIBTEX</code> (input) files.
	<code>dir.file.separator={c}</code>	Use the character <i>c</i> to separate the directory from the file.
	<code>env.separator={c}</code>	Use the character <i>c</i> to separate directories in a path.
<code>-i file</code>	<code>input.file{file}</code>	Add the <code>BIBTEX</code> file <i>file</i> to the list of input files.

A.3 Output File Specification and Status Reporting

By default, the processed $\text{BIB}\text{T}_\text{E}\text{X}$ entries are written to the standard output. This output can be redirected to a file using the command line option `-o` as in

```
bibtool -o output_file
```

The resource name `output.file` can also be used for this purpose.

```
output.file = {output_file}
```

No provisions are made to check if the output file is the same as a input file.

A second output stream is used to display error messages and status reports. The standard error stream is used for this purpose.

The messages can roughly be divided in three categories: error messages, warnings, and status reports. Error messages indicate severe problems. They can not be suppressed. Warnings indicate possible problems which could (possibly) have been corrected. Usually they are displayed but they can be suppressed. Status reports are messages during the processing which indicate actions currently performed. Usually they are suppressed but they can be enabled.

Warning messages can be suppressed using the command line option `-q`. This option toggles the boolean quiet value.

```
bibtool -q
```

The same effect can be obtained by assigning the value `on` or `off` to the resource `quiet`:

```
quiet = on
```

Status reports are useful to see the operations performed. They can be enabled using the command line option `-v`. This option toggles the boolean verbose value.

```
bibtool -v
```

The same can also be achieved with the boolean resource `verbose`:

```
verbose = on
```

Another output stream can be used to select the string definitions. This is described in section A.12 on macros.

Summary

<code>-o file</code>	<code>output.file {file}</code>	Direct output to the file <i>file</i> .
<code>-q</code>	<code>quiet=on</code>	Suppress warnings. Errors can not be suppressed.
<code>-v</code>	<code>verbose=on</code>	Enable informative messages on the activities of <code>BIBTOOL</code> .

A.4 Parsing and Pretty Printing

The first and simplest task we have to provide on $\text{BIB}\text{T}_\text{E}\text{X}$ files is the parsing and pretty printing. This is not superfluous since $\text{BIB}\text{T}_\text{E}\text{X}$ is rather pedantic about the accepted syntax. Thus I decided to try to be generous and correct as may errors as I can.

Each input file is parsed and stored in an internal representation. `BIBTEX` simply ignores any characters between entries. `BIBTOOL` stores the comments and attaches them to the entry immediately following them. Normally anything between entries is simply discarded and a warning printed. The boolean resource `pass.comments` can be used to change this behavior.

```
pass.comments = on
```

If this resource is on then the characters between entries are directly passed to the output file. This transfer starts with the first non-space character after the end of an entry.

The standard `BIBTEX` styles support a limited number of entry types. Those are predefined in `BIBTOOL`. Additional entry types can be defined using the resource `new.entry.type` as in

```
new.entry.type = {Anthology}
```

This option can also be used to redefine the appearance of entry types which are already defined. Suppose we have defined *Anthology* as above. Afterwards we can redefine this entry type to be printed in upper case with the following option:

```
new.entry.type = {ANTHOLOGY}
```

Each undefined entry type leads to an error message.

The internal representation is printed in a format which can be adjusted by certain options. Those options are available through resource files or by specifying resources on the command line.

`print.line.length` This numeric resource specifies the desired width of the lines. Lines which turn out to be longer are tried to split at spaces and continued in the next line. The value defaults to 77.

`print.indent` This numeric resource specifies indentation of normal items, i.e. items in entries which are not strings or comments. The value defaults to 2.

`print.align` This numeric resource specifies the column at which the '=' in non-comment and non-string entries are aligned. This value defaults to 18.

`print.align.key` This numeric resource specifies the column at which the '=' in non-comment and non-string entries are aligned. This value defaults to 18.

`print.align.string` This numeric resource specifies the column at which the '=' in string entries are aligned. This value defaults to 18.

`print.align.preamble` This numeric resource specifies the column at which preamble entries are aligned. This value defaults to 11.

`print.align.comment` This numeric resource specifies the column at which comment entries are aligned.² This value defaults to 10.

²This is mainly obsolete now since comments do not have to follow any syntactic restriction.

`print.comma.at.end` This boolean resource determines whether the comma between fields should be printed at the end of the line. If it is `off` then the comma is printed just before the field name. In this case the alignment given by `print.align` determines the column of the comma.

`print.equal.right` This boolean resource specifies whether the = sign in normal entries is aligned right. If turned off then the = sign is flushed left to the field name. This value defaults to `on`.

`print.newline` This numeric resource specifies the number of newlines between entries. This value defaults to 1.

`print.use.tab` This boolean resource specifies if the TAB character should be used for indenting. This use is said to cause portability problems. Thus it can be disabled. If disabled then the appropriate number of spaces are inserted instead. This value defaults to `on`.

`print.wide.equal` This boolean resource determines whether the equality sign should be forced to be surrounded by spaces. Usually this resource is `off` which means that no spaces are required around the equality sign and they can be omitted if the alignment forces it.

The resource values described above are illustrated by the following examples. First we look at a string entry.

```
@STRING{macro = "This is a rather long replacement text which exceeds one
                line"}
                print.align.string                print.line.length
```

Next we look at an unpublished entry. It has a rather long list of authors and a long title. It shows how the lines are broken.

```
                print.align.key
@Unpublished{  unpublished-key,
author        = "First A. U. Thor and Seco N. D. Author and Third A. Uthor
                and others",
title         = "This is a rather long title of an unpublished entry which
                exceeds one line",
note         = "Some useless comment"
}
print.indent  print.align                print.line.length
```

The field names of an entry are usually printed in lower case. This can be changed with the resource `new.field.type`. The argument of this resource is an equation where left of the '=' sign is the name of a field and on the right side is its print name. They should only contain allowed characters.

```
new.field.type { author = AUTHOR }
```

This feature can be used to rewrite the field types. Thus it is completely legal to have a different replacement text than the original field:

```
new.field.type { OPTauthor = Author }
```

String names are used case insensitive by $\text{BIB}_{\text{T}}\text{E}_X$. $\text{BIB}_{\text{T}}\text{OOL}$ normalizes string names before printing. Per default string names are translated to lower case. Currently two other types are supported: translation to upper case and translation to mixed case, i.e. the first letter upper case and the others in lower case.

The translation is controlled by the resource `symbol.type`. The value is one of the strings `lower`, `upper`, and `cased`. The resource can be set as in

```
symbol.type = upper
```

The macro names are passed through the same normalization apparatus as field types. Thus you can force a rewriting of macro names with the same method as described above. You should be careful when choosing macro names which are also used as field types.

The reference key is usually translated to lower case letters unless a new key is generated (see section A.8). In this case the chosen format determines the case of the key. Sometimes it can be desirable to preserve the case of the key as given (even so $\text{BIB}_{\text{T}}\text{E}_X$ does not mind). This can be achieved with the boolean resource `preserve.key.case`. Usually it is turned off (because of backward compatibility and the memory used for this feature). You can turn it on as in

```
preserve.key.case = on
```

If it is turned on then the keys as they are read are recorded and used when printing the entries. The internal comparisons are performed case insensitive. This is not influenced by the resource `preserve.key.case`. Especially this holds for sorting which does not recognized differences in case.

Summary

<code>new.entry.type{<i>type</i>}</code>	Define a new entry type <i>type</i> .
<code>new.field.type{<i>type</i>}</code>	Define a new field type <i>type</i> .
<code>pass.comments=<i>on</i></code>	Do not discard comments but attach them to the entry following them.
<code>preserve.key.case=<i>on</i></code>	Do not translate keys to lower case when reading.
<code>print.align.comment=<i>n</i></code>	Align comment entries at column <i>n</i> .
<code>print.align.key=<i>n</i></code>	Align the key of normal entries at column <i>n</i> .
<code>print.align.string=<i>n</i></code>	Align the = of string entries at column <i>n</i> .
<code>print.align=<i>n</i></code>	Align the = of normal entries at column <i>n</i> .
<code>print.comma.at.end=<i>on</i></code>	Put the separating comma at then end of the line instead of the beginning.
<code>print.indent=<i>n</i></code>	Indent normal entries to column <i>n</i> .
<code>print.line.length=<i>n</i></code>	Break lines at column <i>n</i> .
<code>print.print.newline=<i>n</i></code>	Number of empty lines between entries.
<code>print.use.tab=<i>on</i></code>	Use the TAB character to compress multiple spaces.
<code>print.wide.equal=<i>off</i></code>	Force spaces around the equal sign.
<code>symbol.type=<i>type</i></code>	Translate symbols according to <i>type</i> : upper, lower, or cased.

A.5 Sorting

The entries can be sorted according to a certain sort key. The sort key is by default the reference key. Sorting can be enabled with the command line switches `-s` and `-S` as in

```
bibttool -s
bibttool -S
```

The first variant sorts in ascending ASCII order (including differentiation of upper and lower case). The second form sorts in descending ASCII order. The same effect can be achieved with the boolean resource values `sort` and `sort.reverse` respectively.

```
sort = {on}
sort.reverse = {on}
```

The resource `sort` determines whether or not the entries should be sorted. The resource `sort.reverse` determines whether the order is ascending (off) or descending (on) ASCII order of the sort key. The sort key is initialized from the reference key if not given otherwise.

Alternatively the sort key can be constructed according to a specification. This specification can be given in the same way as a specification for key generation. This is described in section A.8 in detail.

The associated resource name is `sort.format`. Several formats are combined as alternatives.

```
sort.format = {%N(author)}
sort.format = {%N(editor)}
```

Those two lines are equivalent with the single resource

```
sort.format = {%N(author) # %N(editor)}
```

This means that the sort key is set to the (normalized) author names if an author is given. Otherwise it tries to use the normalized editor name. If everything fails the sort key is empty.

Let us reconsider the unprocessed example on page 8. Without any `sort.format` instructions this entry would be sorted in under “article-full”. With the `sort.format` given above it would be sorted in under “Aamport.LA”.

Note that in ASCII order the case is important. Usually the uppercase letters all come before the lowercase letters.

Beside the normal entries the macros (string entries) are sorted. This happens in per default. The resource `sort.macros` can be used to turn off this feature as in

```
sort.macros = {off}
```

An example of sorting can be seen in section 1.2.1 on page 7.

Summary

<code>-S</code>		Enable sorting of entries in reverse sorting order.
<code>-s</code>	<code>sort</code>	Enable sorting of entries.
	<code>sort.format{spec}</code>	Add disjunctive branch <i>spec</i> to the sort key specifier.
	<code>sort.macros=off</code>	Turn off the sorting of string entries.
	<code>sort.reverse=on</code>	Reverse the sorting order.

A.6 Regular Expression Matching

BIBTOOL makes use of the GNU regular expression library. Thus a short excursion into regular expressions is contained in this manual. Several examples of the application of regular expressions can be found also in other sections of this manual.

A concise description of regular expressions is contained in the document `Regex/regex.texi` contained in the BIBTOOL distribution. In any cases of doubt this documentation is preferable. The remainder of this section contains a short description of regular expressions.

Ordinary characters match only to themselves or their upper or lower case counterpart.

Any character not mentioned as special is an ordinary character. Among others letters and digits are ordinary characters.

E.g. the regular expression `abc` matches the string `abc`.

The period (`.`) matches any single character.

E.g. the regular expression `a.c` matches the string `abc` but it does not match the string `abbc`.

The star (`*`) is used to denote any number of repetitions of the preceding regular expression. If no regular expression precedes the star then it is an ordinary character.

E.g. the regular expression `ab*c` matches any string which starts with a followed by an arbitrary number of b and ended by a c. Thus it matches `ac` and `abbc`. But it does not match the string `abcc`.

The plus (`+`) is used to denote any number of repetitions of the preceding regular expression, but at least one. Thus it is the same as the star operator except that the empty string does not match. If no regular expression precedes the plus then it is an ordinary character.

E.g. the regular expression `ab+c` matches any string which starts with a followed by one or more b and ended by a c. Thus it matches `abbc`. But it does not match the string `ac`.

The question mark (`?`) is used to denote an optional regular expression. The preceding regular expression matches zero or one times. If no regular expression precedes the question mark then it is an ordinary character.

E.g. the regular expression `ab?c` matches any string which starts with a followed by at most one b and ended by a c. Thus it matches `abc`. But it does not match the string `abbc`.

The bar (`|`) separates two regular expressions. The combined regular expression matches a string if one of the alternative separated by the bar does.

E.g. the regular expression `abc|def` matches the string `abc` and the string `def`.

Parentheses ($()$) can be used to group regular expressions. A group is enclosed in parentheses. It matches a string if the enclosed regular expression does.

E.g. the regular expression $a(b|d)c$ matches the strings abc and adc .

The dollar ($\$$) matches the empty string at the end of the string. It can be used to anchor a regular expression at the end. If the dollar is not the end of the regular expression then it is an ordinary character.

E.g. the regular expression $abc\$$ matches the string $aaaabc$ but does not match the string $abcdef$.

The hat (\wedge) matches the empty string at the beginning of the string. It can be used to anchor a regular expression at the beginning. If the hat is not the beginning of the regular expression then it is an ordinary character. There is one additional context in which the hat has a special meaning. This context is the list operator described below.

E.g. the regular expression $\wedge abc$ matches the strings $abcccc$ but does not match the string $aaaabc$.

The brackets ($[\]$) are used to denote a list of characters. If the first character of the list is the hat (\wedge) then the list matches any character not contained in the list. Otherwise it matches any characters contained in the list.

E.g. the regular expression $[abc]$ matches the single letter strings a , b , and c . It does not match d .

The regular expression $[\wedge abc]$ matches any single letter string not consisting of an a , b , or c .

The backslash (\backslash) is used for several purposes. Primarily it can be used to quote any special character. Thus if a special character is preceded by the backslash then it is treated as if it were an ordinary character.

If the backslash is followed by a digit d then this construct is the same as the d th matching group.

E.g. the regular expression $(an)\backslash 1as$ matches the string $ananas$ since the first group matches an .

If the backslash is followed by the character n then this is equivalent to entering a newline.

If the backslash is followed by the character t then this is equivalent to entering a single TAB character.

A.7 Selecting Items

A.7.1 Extracting by aux Files

BIBTOOL includes a module to extract BIB_TE_X entries required for a document. This is

accomplished by analyzing the `aux` file of the document. The `aux` file is usually produced by \LaTeX . It contains the information which $\text{BIB}\TeX$ files and which references are used in the document. Only those entries mentioned in the `aux` file are selected for printing. Since the $\text{BIB}\TeX$ files are already named in the `aux` file it is not necessary to specify an input file.

To use an `aux` file the command line option `-x` can be given. This option is followed by the name of the `aux` file.

```
bibttool -x file.aux
```

Multiple files can be given this way. As always the same functionality can be requested with a resource. The resource `extract.file` can be used for this purpose.

```
extract.file = {file.aux}
```

A small difference exists between the two variants. the command line option automatically sets the resource `print.all.strings` to `off`. This has to be done in the resource file manually. One special feature of $\text{BIB}\TeX$ is supported. If the command `\nocite{*}` is given in the \LaTeX file then all entries of the bibliography files are included in the bibliography. The same behavior is imitated by the extracting mechanism of `BIBTOOL`.

An example of extracting can be seen in section 1.2.4 on page 10.

A.7.2 Extracting with Regular Expressions

Another selecting mechanism uses regular expressions to select items. This feature can be used in addition to the selection according to `aux` files.

The regular expression syntax is identical to the one used in GNU Emacs. For a description see section A.6.

The resource `select` allows to specify which fields should be used to select entries. The general form is as follows:

```
select = {field1 ... fieldn "regular_expression"}
```

If no field is specified then the regular expression is searched in each field.

If no regular expression is specified then any value is accepted; i.e. the regular expression `"."` is used.

Any number of selection rules can be given. An entry is selected if one of those rules selects it. The select rule selects an entry if this entry has a field named `field` which has a substring matching `regular_expression`. The field can be missing in which case the regular expression is tried to match against any field in turn.

The pseudo fields `$key`, `$type`, and `@type` can be used to access the key and the type of the entry. See page 38 for details. The routines used there are the same as those used here.

The boolean resource `select.case.sensitive` can be used to determine whether the selection is performed case sensitive or not:

```
select.case.sensitive = off
```

A regular expression can be specified in the command line using the option `-X` as in

```
bibttool -X regular_expression
```

The fields compared against this regular expression are given in the string valued resource `select.fields`. Initially this resource has the value `$key`. In general the value is a list of fields and pseudo fields to be considered. The elements of the list are separated by spaces. If the list is empty then all fields and the key are considered for comparison.

Thus the following setting means that the regular only the fields `author` and `editor` are considered when doing a selection.

```
select.fields = "author editor"
```

Without changing the resource `select.fields` the command line given previously is equivalent to the (longer) command

```
bibttool -- select{$key "regular_expression"}
```

Note that the resources `select.case.sensitive` and `select.fields` are used for all regular expressions following their definition until they are redefined. This means that it is possible to specify that some comparisons are done case sensitive and others are not done case sensitive. Finally the resource `extract.regex` can be used as in

```
extract.regex = {regular_expression}
```

This is equivalent to specifying a single regular expression to be matched against the key. This feature is kept for backward compatibility only. It is not encouraged and will vanish in a future release.

Summary

<code>-x</code>	<code>extract.file{file}</code>	Extract the entries from <code>aux</code> file.
	<code>extract.regexp{expr}</code>	Discouraged backward compatibility command.
<code>-X regex</code>	<code>select{spec}</code>	Select certain entries.
	<code>select.case.sensitive=off</code>	Turn off the case insensitive comparison.
	<code>select.fields{fields}</code>	Determine fields for <code>-X</code> .

A.8 Key Generation

The key generation facility provides a mean to uniformly replace the reference keys by generated values. Some algorithms are hardwired, namely the generation of short keys or long keys either unconditionally or only when they are needed. Additionally a free formatting facility is provided. This can be used to specify your own algorithm to generate keys. The generation of new keys can be enabled using the command line option `-f` in the following way:

```
bibttool -f format
```

This command adds `format` disjunctively to the formatting instructions already given. The same effect can be achieved with the resource `key.format`.

```
key.format = {format}
```

Some values of `format` have a special meaning. Fixed formatting rules are used when one of them is in effect. The special values are described below. To illustrate their results we consider the following `BIBTEX` database entries:

```

@Unpublished{unpublished-key,
  author   = "First A. U. Thor and Seco N. D. Author and Third A. Uthor
             and others",
  title    = "This is a rather long title of an unpublished entry which
             exceeds one line",
  ...
}
@Article{,
  author = {L[eslie] A. Aamport},
  title = {The Gnats and Gnus Document Preparation System},
  ...
}
@BOOK{whole-collection,
  editor = "David J. Lipcoll and D. H. Lawrie and A. H. Sameh",
  title = "High Speed Computer and Algorithm Organization",
  ...
}
@MISC{misc-minimal,
  key = "Missilany",
  note = "This is a minimal MISC entry"
}

```

short If a field named `key` is present then its value is used. Otherwise if an `author` or `editor` field are present, then this field is used. The short version uses last names only. Afterwards a `title` or `booktitle` field is appended, after the `fmt.name.title` separator has been inserted. Finally if all else fails then the default key `default.key` is used. The result is disambiguated (cf. `key.base`).

To see the effect we apply `BIBTOOL` to the example entries given earlier with the command line argument `-- key.format=short`. This results in the following keys (remaining lines skipped):

```

@Unpublished{      thor.author.ea:this,
@Article{         aamport:gnats,
@Book{           lipcoll.lawrie.ea:high,
@Misc{           missilany,

```

long The long version acts like the short version but incorporates initials when formatting names.

If `BIBTOOL` is applied to the example entries given earlier with the command line argument `-- key.format=long` we get the following keys:

```

@Unpublished{      thor.fau.author.snd.ea:this,
@Article{         aamport.la:gnats,

```

```
@Book{          lipcoll.dj.lawrie.dh.ea:high,
@Misc{          missilany,
```

`new.short` This version formats like `short` but only if the given key field is empty. This is obsoleted by the resource `preserve.keys` and will be withdrawn in a future release.

If `BIBTOOL` is applied to the example entries given earlier with the command line argument `-- key.format=short.need` we get the following keys:

```
@Unpublished{  unpublished-key,
@Article{      aamport:gnats,
@Book{         whole-collection,
@Misc{         misc-minimal,
```

`new.long` This version formats like `long` but only if the given key field is empty. This is obsoleted by the resource `preserve.keys` and will be withdrawn in a future release.

If `BIBTOOL` is applied to the example entries given earlier with the command line argument `-- key.format=short.need` we get the following keys:

```
@Unpublished{  unpublished-key,
@Article{      aamport.la:gnats,
@Book{         whole-collection,
@Misc{         misc-minimal,
```

`empty` The empty version clears the key entirely. The result does not conform to the `BIBTEX` syntax rules. This feature can be useful if a resource file must be used which generates only new keys. In this case a first pass can clear the keys and the given resource file can be applied in a second pass to generate all keys.

If `BIBTOOL` is applied to the example entries given earlier with the command line argument `-- key.format=empty` we get the following keys:

```
@Unpublished{  ,
@Article{      ,
@Book{         ,
@Misc{         ,
```

In contrast to the command line option, the resource instruction only modifies the formatting specification. The key generation has to be activated explicitly. This can be done using the command line option `-F` as in

```
bibttool -F
```

Alternatively the boolean resource `key.generation` can be used in a resource file:

```
key.generation = on
```

Usually all keys are regenerated. This can have the unpleasant side-effect to invalidate citations in old documents. For this situation the resource `preserve.keys` is meant. This

resource is usually `off`. If it is turned `on` then only those entries receive new keys if they do not have a key already. This means that the input contains only a sequence of whitespace characters (which is not accepted by `BIBTEX`) as in the following example:

```
@Article{,
  author = {L[eslie] A. Aamport},
  title = {The Gnats and Gnus Document Preparation System},
  journal = {\mbox{G-Animal's} Journal},
  year = 1986,
  volume = 41,
  number = 7,
  pages = "73+",
  month = jul,
  note = "This is a full ARTICLE entry",
}
```

Even if `preserve.keys` is `on`, `BIBTOOL` still changes all keys to lower case by default. This can be suppressed by switching `preserve.key.case` to `on` (see section A.4).

When the `key.format` is not empty then the keys are disambiguated by appending letters or numbers. Thus there can not occur a conflict which would arise when two entries have the same key. The disambiguation uses the resource `key.number.separator`. If a key is found (during the generation) which is already been used then the valid characters from the value of this resource is appended. Additionally a number is added. The appearance of the number can be controlled with the resource `key.base`. This resource can take the values `upper`, `lower`, and `digit`. The effect can be seen in the following table:

generated key	digit	lower	upper
key	key	key	key
key	key*1	key*a	key*A
key	key*2	key*b	key*B
key	key*3	key*c	key*C
key	key*4	key*d	key*D

As we have seen there are options to adapt the behavior of formatting. Before we explain the free formatting specification in section A.9 we will present the formatting options. Those options can be activated from a resource file or with the corresponding feature to specify resource instructions on the command line.

`preserve.keys` This boolean resource determines whether exiting keys should be left unchanged when new keys are generated. The default value is `off`.

`preserve.key.case` This boolean resource determines whether keys should be recorded and used exactly as read opposed to normalizing them by translating all uppercase letters to lower case. The default value is `off`.

`default.key` The value of this resource is used if nothing else fits. The default value is `**key*`.

key.base The value of this resource is used to determine the kind of formatting the disambiguating number. Possible values are **upper**, **lower**, and **digit**. Uppercase letters, lower case letters, or digits are used respectively.

key.number.separator The value of this resource is used to separate the disambiguating number from the rest of the key. The default value is *****.

key.expand.macros The value of this boolean resource is used to indicate whether macros should be expanded while generating a key. The default value is **off**.

fmt.name.title The value of this resource is used by the styles **short** and **long** to separate names and titles. The default value is **:.**

fmt.title.title The value of this resource is used to separate words inside titles. The default value is **:.**

fmt.name.name The value of this resource is used to separate different names (where the **BIBTEX** file has **and**) when formatting names. The default value is **..**

fmt.inter.name The value of this resource is used to separate parts of multi-word names when formatting names. The default value is **-.**

fmt.name.pre The value of this resource is used to separate names and first names when formatting names. The default value is **..**

fmt.et.al The value of this resource is used to format **and others** parts of a name list. The default value is **.ea.**

The key style **short** can be formulated in terms of the format specification given in section A.9 as follows:

```
{
  { %-2n(author)
    # %-2n(editor)
  }
  { %s($fmt.name.title) %-1T(title)
    # %s($fmt.name.title) %-1T(booktitle)
    #
  }
}
#
{ { %s($fmt.name.title) %-1T(title)
  # %s($fmt.name.title) %-1T(booktitle)
  }
}
# %s($default.key)
```

The syntax and meaning of such format specifications is explained in section A.9.

Summary

	<code>preserve.keys=off</code>	Do not generate new keys if one is already present.
	<code>preserve.key.case=on</code>	Do not translate keys to lower case when reading.
	<code>default.key={key}</code>	Key used if nothing else applies.
	<code>fmt.et.al={ea}</code>	String used to abbreviate further names.
	<code>fmt.inter.name={s}</code>	String used between parts of names.
	<code>fmt.name.name={s}</code>	String used between names.
	<code>fmt.name.pre={s}</code>	String separating first and last names.
	<code>fmt.name.title={s}</code>	String used to separate names from titles.
	<code>fmt.title.title={s}</code>	String used to separate words in titles.
	<code>key.base={base}</code>	Kind of numbers or letters for disambiguating keys.
	<code>key.expand.macros=off</code>	Turn of macro expansion for key generation.
<code>-f</code>	<code>key.format{fmt}</code>	Set the specification for key generation to <i>fmt</i> .
<code>-F</code>	<code>key.generation=on</code>	Turn on key generation.
	<code>key.number.separator={}</code>	

A.9 Format Specification

A.9.1 Constant Parts

The simplest component of a format is a constant string. Such strings are made up of any character except whitespace and the following ten characters

" # % ' () , = { }

This choice of special characters is the same as the special characters of `BIBTEX`. Since no means is provided to include a special character into a format string we guarantee that the resulting key string is conform to the `BIBTEX` rules.

For example the following strings are legal constant parts of a format:

```
Key
the_name.of-the-@author-is:
```

Now we come to explain the meaning of the special characters. The first case consists of the whitespace characters. They are simply ignored. Thus the following format strings are equal:³

```
Author Or Editor
AuthorOrEditor
A u t h o r   O r   E d i t o r
```

³Well, this is not the whole truth. Internally it makes a difference whether there is a space or not. In the presence of spaces more memory is used. But you shouldn't worry too much about this.

A.9.2 Formatting Fields

The next component of formats are made up formatting instructions which are starting with a % character. The general idea has been inspired by formatting facilities of C. Since there are several different types of information in a BibTeX entry we provide several primitives for formatting. The simplest form is for instance

```
%N(author)
```

The % character is followed by a single character — here N — which indicates the way of formatting and the name of the field to be formatted enclosed in parenthesis. The example above requests to format the field `author` according to formatting rules for names (N). The general form is

```
%sign pre.post qualifier letter(field)
```

In this specification *sign* is + or -. + means that all characters will be translated to upper case. - means that all characters will be translated to lower case. If no sign is given, the case of the field is preserved.

pre and *post* are positive integers whose meaning depends on the format letter *letter*. *qualifier letter* is a one letter specification indicating the desired formatting type optionally preceded by the qualifier #. Possible values are as described in the following list:

- p Format names according to the format specifier number *post*. In a list of names at most *pre* names are used. If there are more names they are treated as given as **and others**. *pre* defaults to 2 and *post* defaults to 0.

See section A.9.8 for a description of how to specify name formats.

Example

```
author = {A. U. Thor and S. O. Meone and others}
```

With the above item we get the following results:

```
%p(author)    Thor.Meone.ea
%1p(author)   Thor.ea
%-2p(author)  thor.meone.ea
%+1p(author) THOR.EA
```

- n Format last names only.

In a list of names at most *pre* last names are used. If there are more names they are treated as given as **and others**. If *post* is greater than 0 then at most *post* characters per name are used. Otherwise the whole name is considered.

pre defaults to 2 and *post* defaults to 0.

This is the same as using the p format specifier with the post value of 0. The *post* value of the n specifier is used as the *len* value of the first item of the name format specifier. (See also section A.9.8)

Example

```
author = {A. U. Thor and S. O. Meone and others}
```

With the above item we get the following results:

```
%n(author)    Thor.Meone.ea
%1n(author)   Thor.ea
%-2n(author)  thor.meone.ea
%+1n(author)  THOR.EA
%.3n(author)  Tho.Meo.ea
```

N Format names with last names and initials.

In a list of names at most *pre* last names are used. If there are more names they are treated as given as **and others**. If *post* is greater than 0 then at most *post* characters per name are used. Otherwise the whole name is considered.

pre defaults to 2 and *post* defaults to 0.

This is the same as using the **p** format specifier with the *post* value of 1. The *post* value of the **n** specifier is used as the *len* value of the first item of the name format specifier. (See also section A.9.8)

Example

```
author = {A. U. Thor and S. O. Meone and others}
```

With the above item we get the following results:

```
%N(author)    Thor.AU.Meone.SO.ea
%1N(author)   Thor.AU.ea
%-2N(author)  thor.au.meone.so.ea
%+1N(author)  THOR.AU.EA
%.3N(author)  Tho.AU.MeO.SO.ea
```

d Format a number, e.g. a year.

The first number in the field is searched. At most *pre* digits — counted from the right — are used. E.g. the field "june 1958" formatted with **%2d** results in 58.

pre defaults to a large number except in when the negative sign is present. Then it defaults to 1.

If no number is contained in the field then this specifier fails. Thus the specifier **%0d** can be used to check for a number.

Positive and negative signs make no sense in specifying translations since numbers have no uppercase or lowercase counterparts. Thus they have a different meaning in this context.

If the positive sign is given then the specifier does not fail at all. Instead of failing a single 0 is used.

If the negative sign is given then the result is padded with 0 if required. In this case the specifier does not fail at all. Even if no number is found then an appropriate number of 0s is used.

Example

```
pages = {89--123}
```

With the above item we get the following results:

```
%d(pages)      89
%1d(pages)     9
%4d(pages)     89
%-4d(pages)    0089
%-5.2d(pages)  00123
%.3d(pages)    fails
%+.3d(pages)   0
%0d(pages)     succeeds with empty result
```

D Format a number.

This format specifier acts like the `d` specifier except that the number is not truncated. Thus a large number comes out complete and not only the last few digits.

Example

```
pages = {89--123}
```

With the above item we get the following results:

```
%D(pages)      89
%1D(pages)     89
%4D(pages)     89
%-4D(pages)    0089
%-5.2D(pages)  00123
%.3D(pages)    fails
%+.3D(pages)   0
%0D(pages)     89
```

s Take a field as is (after translation of special characters).

At most *pre* characters are used.

pre defaults to a large number.

Example

```
author = {A. U. Thor and S. O. Meone and others}
```

With the above item we get the following results:

```
%s(author)     A.-U.-Thor-and-S.-O.-Meone-and-others
%8s(author)    A.-U.-Th
%-8s(author)   a.-u.-th
%+8s(author)   A.-U.-TH
%0s(author)    succeeds with empty result
```

T Format sentences. Certain words are ignored.

At most *pre* words are used. The other words are ignored. If *pre* is 0 then no artificial

limit is forced. If *post* is positive then at most *post* letters of each word are considered. Otherwise the complete words are used.

New words to be ignored can be added with the resource `ignored.word`.

pre defaults to 1 and *post* defaults to 0.

Example

```
title = {The Whole Title}
```

With the above item we get the following results:

```
%T(title)      Whole
%2T(title)     Whole-Title
%2.1T(title)   W-T
%-T(title)     whole
%+T(title)     WHOLE
```

t Format sentences. In contrast to the format letter **T** no words are ignored.

At most *pre* words are used. The other words are ignored. If *pre* is 0 then no artificial limit is forced. If *post* is positive then at most *post* letters of each word are considered. Otherwise the complete words are used.

pre defaults to 1 and *post* defaults to 0.

Example

```
title = {The Whole Title}
```

With the above item we get the following results:

```
%t(title)      The
%2t(title)     The-Whole
%2.1t(title)   T-W
%-t(title)     the
%+t(title)     THE
```

W Format word lists.

This specifier acts like **T** except that nothing is inserted between words.

Example

```
title = {The Whole Title}
```

With the above item we get the following results:

```
%W(title)      Whole
%2W(title)     WholeTitle
%2.1W(title)   WT
%-W(title)     whole
%+W(title)     WHOLE
```

w Format word lists.

This specifier acts like **t** except that nothing is inserted between words.

Example

```
title = {The Whole Title}
```

With the above item we get the following results:

```
%w(title)      The
%2w(title)     TheWhole
%2.1w(title)   TW
%-w(title)     the
%+w(title)     THE
```

#p Count the number of names.

If no *sign* is given or the *sign* is + then the following rules apply. If the count is less than *pre* or the count is greater than *post* then this specifier fails. Otherwise it succeeds without adding something to the key.

The construction **and others**, which indicates an unspecified number of additional authors, counts as one single author.

If the *sign* is - then the specifier succeeds if and only if the specifier without this sign fails. Thus the - acts like a negation of the condition.

If *post* has the value 0 than this is treated like ∞ .

If *a* is the number of names separated by **and** then

`%l.h#p` succeeds if and only if $l \leq a \leq h$.

`%-l.h#p` succeeds if and only if $l > a$ or $a > h$.

pre and *post* both defaults to 0.

Example

```
author = {A. U. Thor and S. O. Meone and others}
```

With the above item we get the following results:

```
%2#p(author)    succeeds with empty result
%4#p(author)    fails
%-4#p(author)   succeeds with empty result
%3.4#p(author)  succeeds with empty result
%-3.4#p(author) fails
```

#n Is the same as **#p**.

#N Is the same as **#p**.

#s Count the number of allowed characters.

If no *sign* is given or the *sign* is + then the following rules apply. If the count is less than *pre* or the count is greater than *post* then this specifier fails. Otherwise it succeeds without adding something to the key.

If the *sign* is - then the specifier succeeds if and only if the specifier without this sign fails. Thus the - acts like a negation of the condition.

If *post* has the value 0 than this is treated like ∞ .

pre and *post* both default to 0.

If *a* is the number of allowed characters then

`%l.h#p` succeeds if and only if $l \leq a \leq h$.

`%-l.h#p` succeeds if and only if $l > a$ or $a > h$.

Example

```
title = {The Whole Title}
```

With the above item we get the following results:

```
%#s(title)          succeeds with empty result
%13,13#s(title)     succeeds with empty result
%10,16#s(title)     succeeds with empty result
%-10,16#s(title)    fails
```

#w Count the number of words. All words are considered as valid. The division into words is performed after deTeXing the field.

If no *sign* is given or the *sign* is + then the following rules apply. If the count is less than *pre* or the count is greater than *post* then this specifier fails. Otherwise it succeeds without adding something to the key.

If the *sign* is - then the specifier succeeds if and only if the specifier without this sign succeeds. Thus the - acts like a negation of the condition.

If *post* has the value 0 than this is treated like ∞ .

pre and *post* both default to 0.

If *a* is the number of words separated by whitespace then

`%l.h#p` succeeds if and only if $l \leq a \leq h$.

`%-l.h#p` succeeds if and only if $l > a$ or $a > h$.

Example

```
title = {The Whole Title}
```

With the above item we get the following results:

```
%#w(title)          succeeds with empty result
%3,3#w(title)       succeeds with empty result
%1,6#w(title)       succeeds with empty result
%-1,1#w(title)      fails
```

#t Is the same as **#w**.

#W Count the number of words. Certain words are ignored. The ignored words are determined by the resource `ignored.word`. The division into words is performed after deTeXing the field.

If no *sign* is given or the *sign* is + then the following rules apply. If the count is less than *pre* or the count is greater than *post* then this specifier fails. Otherwise it succeeds without adding something to the key.

If the *sign* is - then the specifier succeeds if and only if the specifier without this sign fails. Thus the - acts like a negation of the condition.

If *post* has the value 0 than this is treated like ∞ .

pre and *post* both default to 0.

If *a* is the number of words separated by whitespace which are not marked to be ignored then

`%l.h#p` succeeds if and only if $l \leq a \leq h$.

`%-l.h#p` succeeds if and only if $l > a$ or $a > h$.

Example

```
title = {The Whole Title}
```

With the above item we get the following results:

```
%#W(title)      succeeds with empty result
%2,2#W(title)   succeeds with empty result
%1,6#W(title)   succeeds with empty result
%-1,6#W(title)  fails
```

`#T` Is the same as `#W`.

The field specification (*field*) selects the field of the entry to be formatted. As usual in `BIBTEX` the case of the letters is ignored. If the field does not exist then the formatting fails and continues at the next alternative (see below).

But the field is not only sought in the current entry. According to the behaviour of `BIBTEX` the special field `crossref` is taken into account. If a field is missing then the entry named in the `crossref` field is also considered. Since this dereferencing contains the potential danger of an infinite loop the number of dereferencing steps is restricted by the numeric resource `crossref.limit`. The number of uses of the `crossref` field is limited by the value of this resource. The default of this resource is 32.

Usually a value of 1 would be sufficient for `BIBTEX` files conforming to the standard styles. Nevertheless other applications can be imagined where a higher value is desirable.

To turn off the `crossref` feature complete you can set the value of `crossref.limit` to 0. In this case only the fields found in the entry itself are considered.

In addition to the ordinary fields of an entry there are several pseudo fields. They are listed below.

`$key` This pseudo field contains the old reference key — before generating a new one. If none has been given then the access fails.

`$sortkey` This pseudo field contains the string according to which the sorting is performed. It defaults to the reference key.

\$default.key This pseudo field contains the value of the resource **default.key** similarly the resources **fmt.name.title**, **fmt.title.title**, **fmt.name.name**, **fmt.inter.name**, **fmt.name.pre**, and **fmt.et.al** can be accessed.

\$source This pseudo field contains the name of the file the entry has been read from. If this file can not be determined, e.g. because the entry has been read from **stdin**, then this pseudo field is empty.

\$type This pseudo field contains the type of the entry, i.e. the string following the initial **@** of an **BIBTEX** entry. E.g. **article**. It is always present.

@type This pseudo field is matched against the type of the entry. If they are identical (ignoring cases) then the type is returned. Otherwise the access fails.

In an article item the specification **%s(@Article)** succeeds and returns **Article** whereas **%s(@Book)** fails.

\$day This pseudo field contains the current day as a two digit number or the empty string if this value is not available. The date and time values are determined at the beginning of the **BIBTOOL** run and does not reflect the execution time used by **BIBTOOL**.

On some systems the timing function might be missing or returning strange values. In this case the timing fields simply return the empty string.

\$month This pseudo field contains the current month as a two digit number or the empty string if this value is not available.

\$mon This pseudo field contains the current month name as a string or the empty string if this value is not available.

\$year This pseudo field contains the current year as a four digit number or the empty string if this value is not available.

\$hour This pseudo field contains the current hour as a two digit number or the empty string if this value is not available.

\$minute This pseudo field contains the current minute as a two digit number or the empty string if this value is not available.

\$second This pseudo field contains the current second as a two digit number or the empty string if this value is not available.

\$user This pseudo field contains the contents of the environment variable **\$USER** or the empty string if this value is not available. On **UNIX** systems this variable usually contains the name of the user. This can be used to write a logging information into a field.

\$hostname This pseudo field contains the contents of the environment variable **\$HOSTNAME** or the empty string if this value is not available.

A.9.3 Conjunctions

Conjunctions are formatting instructions evaluated in sequence. The conjunctions are simply written by successive formatting instructions. A conjunction succeeds if every part succeeds. The empty conjunction always succeeds.

Suppose an $\text{BIB}\text{T}_{\text{E}}\text{X}$ entry contains fields for `editor` and `year`. Then the following conjunction succeeds:

```
%-3n(editor) : %2d(year)
```

If the value of the `editor` field is "E.D. Itor" and the `year` field contains "1992" then the result is `itor:92`.

A.9.4 If-Then-Else

Depending on the presence of a (pseudo-) field formatting instructions can be issued. This corresponds to an if-then-else statement in a PASCAL-like language. The syntax is as follows:

```
(field){then-part} {else-part}
```

If the access to the (pseudo-) field as described in A.9.2 succeeds then the *then-part* is evaluated. Otherwise the *else-part* is evaluated. Both parts may be empty. Nevertheless the braces are required.

Let us look at an example. The following construction can be used to format a field `author` if it is present or print an constant string.

```
(author){%N(author)}{--no-author--}
```

A.9.5 Alternatives

Alternatives (disjunctives) are separated by the hash mark (`#`). The general form is

```
alternative-1 # alternative-2 # ... # alternative-n
```

The alternatives are evaluated from left to right. The first one that succeeds terminates the processing of all alternatives with success. If no alternative is successful then the whole construct fails.

An alternative can be empty. The empty alternative succeeds without any other effect.

The example given in subsection A.9.4 can be also written as

```
%N(author) # --no-author--
```

If the `author` field is accessible the first alternative succeeds and terminates the construct. Otherwise the constant string is used. This constant string always succeeds.

A.9.6 Grouping

Any number of constructs can be enclosed in braces (`{}`) for grouping. Thus the precedence of operators can be bypassed.

Coming back to our example from the previous subsection. To complicate the example we want to append an optional title, or an constant string. This is accomplished as follows.

```
{ %N(author) # --no-author-- } { %T(title) # --no-title-- }
```

The grouping allows to restrict the range of the alternative operator `#` in this example.

Another example shows how the alternative together with grouping can be used to share a format specification for certain types of entries:

```
{%0s(@book) # %0s(@proceedings)} --book-or-proc--
```

The `%0s` specifier is used to check for the existence of a certain field without actually adding anything to the output. Other constructs may serve for the same purpose. This construct is applied to the pseudo-fields `@book` and `@proceedings`. The access to the pseudo-field fails if requested in another type of entry. Those two checks are combined to form a disjunction. Thus the following code—the constant in this example—is reached only if we are in a book or in a proceedings entry. It is not reached in an article.

A.9.7 Expanding \TeX / \LaTeX Macros

When fields are formatted certain \LaTeX macros may be replaced by pure text. Each macro not defined is simply ignored. Initially no \LaTeX macro is defined. The resource `tex.define` can be used to define \LaTeX macros. The syntax is very close to \LaTeX . The simplest form is the following definition.

```
tex.define = {macro=replacement text}
```

This resource defines a simple macro which is replaced by the replacement text. This replacement text may in turn contain macros.

In addition to this simple macro also macros involving arguments can be defined. As in \LaTeX 's `\newcommand` the number of arguments is appended after the macro name.

```
tex.define = {macro[arg]=replacement text}
```

The number of arguments may not exceed 9. The actual parameters are addressed by writing `#n`, where `n` is the number of the argument.

For instance, this feature can be used to ignore certain arguments of macros.

Note that there should be no spaces between the macro head and the equality sign (`=`). Similarly any unwanted spaces after the equality sign may have strange effects.

Usually the macro name starts with a backslash (`\`). If the macro name starts with another character then this character is made active (cf. [Knu89]). This feature is especially useful for translating characters with an extended ASCII code (≥ 128) to the appropriate \TeX macros.

For instance the following definition forces the expansion of the macro `\TeX` to the string `TeX`.

```
tex.define = {\TeX=TeX}
```

Without this definition the title `The \TeX{}book` would result in `book`. With this definition the same title results in `TeXbook`.

Suppose you have an input file containing 8-bit characters (e.g. ISO 8859-1 encoding). The following definition can be used to map this character into a pure ASCII string⁴

```
tex.define = {ü=ue}
```

Some useful definitions can be found in the libraries distributed with BIBTOOL (see also appendix C).

A.9.8 Name Formatting

Names are a complicated thing. BIBTOOL tries to analyze names and “understand” them correctly. According to the `BIBTEX` definition a name consists of four types of components:

- The first names are any names before the last names which start with an upper case letter.
E.g. for the name “Ludwig van Beethoven” the first name is “Ludwig”.
- The last name is the last word (or group of words) which does not belong to the junior part.
E.g. for the name “Ludwig van Beethoven” the last name is “Beethoven”.
- The von part are the names before the last name which start with lower case letters.
E.g. for the name “Ludwig van Beethoven” the von part consists of the word “van”.
- The junior part of a name is an appendix following the last name. BIBTOOL knows only a small number of words that can appear in the junior part: junior, jr., senior, sen., Esq., PhD., and roman numerals up to XXX.

Everything except the last name is optional. Each part can also consist of several words. More on names can be found in [Lam94] and [Pat88b].

BIBTOOL provides a means to specify how the various parts of a name should be used to construct a string. This string can be used as part of a key with the `%p` format specifier (see above).

BIBTOOL uses a small number of name format specifiers.⁵ Initially most of them are undefined. The name format specifier 0 is initially set to the value `%1[fmt.inter.name]`. The name format specifier 1 is initially set to the value `%1[fmt.inter.name]%1f[fmt.inter.name]`. The name format specifiers 0 and 1 are used by the formatting instructions `%N` and `%n`. Thus you should be careful when redefining them. To help you having an eye on these two name format specifiers BIBTOOL issues a warning when they are modified.

The resource `new.format.type` can be used to assign values to those name format specifiers:

```
new.format.type = {17=" %f%v%i" }
```

⁴To add an e is the German convention for umlaut characters.

⁵The exact number can be changed in the configuration file before compilation. The default is 128.

This instruction sets the name format specifier number 17 to the given value. This value is a string of characters to be used directly. There is only one construct which is not used literally. This construct is started by a % sign optionally followed by a + or a - and a number. Next comes one of the letters f, v, l, or j. Finally there are three optional arguments enclosed in brackets.

Thus the general form looks as follows:

```
%sign len letter[mid] [pre] [post]
```

The letter f denotes all first names. The letter l denotes all last names. The letter v denotes all words in the von part. The letter j denotes all words in the junior part.

If *sign* is + then the words are translated to upper case. If *sign* is - then the words are translated to lower case. If no sign is given then no conversion is performed.

The number *len* can be used to specify the number of characters to be used. Each word is truncated to at most *len* characters if *len* is greater than 0. Otherwise no truncation is performed. Thus a value of 0 acts like ∞ .

If [*mid*] is given then this string is used between several words of the given part. If none is given then the empty string is used.

If [*pre*] is given then this string is used before the given part, but only if the part is not empty. If none is given then the empty string is used.

If [*post*] is given then this string is used after the given part, but only if the part is not empty. If none is given then the empty string is used.

Now we can come to an example. Suppose the name field contains the value **Cervantes Saavedra, Miguel de**⁶. This name has two last names, one first name and one word in the von part.

We want to apply the following name format specifier

```
%1f[.] [] [.]%1v[.] [] [.]%3l[-]%1j
```

This means we want to use abbreviation of first name, von and junior part to one letter and of three letters of the last name. Thus we will get the result **M.d.Cer-Saa**.

Note that the name specifier does not take care to include only allowed letters into a key. Thus watch out and avoid special characters as whitespace and comma.

A.9.9 Example

To end this section we should have a look at a complete example of key generation specification. For this purpose we define a rule according to which the keys should be generated:

1. If a field named **bibkey** is present then the value of this field should be used.
2. If the type of the entry is a book then the authors/editors are used followed by the year separated by a colon.

⁶This is the author of “Don Quixote”

3. If the type of the entry is an article in a journal (**article**) then the author, the journal, the number, and the year should be used. Author and journal should be separated by a colon, The journal should be abbreviated with the initials and separated from number and year by a period.
4. If the type of the entry is a volume of conference proceedings (**proceedings**) then the editor, the first 5 initials of the title and the year should be used. The editor should be followed by a colon and the year preceded by a period.
5. If the type of the entry is a contribution in conference proceedings then the author, the initials of the book title and the year should be used.
6. Otherwise the first three letters of the type, the author and the year should be used. If no author is given then the initials of the title should be used instead—but at most 6 characters.

The names should include up to two names abbreviated to four letters and should be translated to lower case. If an information is missing then the respective part together with the following separator should be omitted.

The disambiguation should be done by appending upper case letters without a preceding string. If everything else fails three question marks should be inserted as key.

To implement this scheme we write the following specification into a resource file:

```
key.expand.macros = on
key.base = upper
key.number.separator = {}
key.format =
{
  %s(bibkey)
#
  %0w(@book)
  { %-2.4n(author): # %-2.4n(editor): # }
  { %4d(year)      # }
#
  %0w(@article)
  { %-2.4n(author): # }
  { %-.1W(journal). # }
  { %4d(year)      # }
#
  %0w(@proceedings)
  { %-2.4n(editor): # }
  { %-.1W(title).   # %-.1W(booktitle). # }
  { %4d(year)      # }
#
  %0w(@inproceedings)
  { %-2.4n(author): # }
  { %-.1W(booktitle). # }
  { %4d(year)      # }
#
```

```

    %3s($type)-
    { %-2.4n(author):
    # %-6.1W(title).
    }
    {%4d(year) # }
#
    %3s($type)-
    %4d(year)
# ???
}

```

Since each part has been explained before we just need some overall remarks. I prefer to use the backtracking-based disjunctions instead of nested if-then-else constructs because they save some braces. They can be read as a switch statement, or even better as a `cond` statement in Lisp. This means they describe cases. The first successful case terminates the evaluation of the whole cascade.

The constructions like `%0w(@book)` are used to distinguish the different types. This construction does not produce any output. It just succeeds or fails depending on the type of the current entry. The `%0w` could also be replaced by other specifiers which serve the same purpose.

The constructions like `{%4d(year) # }` always succeed. The hash sign (`#`) catches the failure and inserts the second alternative—which happens to be empty—if the requested field does not exist.

Summary

	<code>tex.define{macro=text}</code>	Expand the \TeX macro <i>macro</i> to <i>text</i> .
	<code>tex.define{macro[n]=text}</code>	Expand the \TeX macro with arguments.
	<code>new.format.type{n=spec}</code>	Define a new way to format names.

A.10 Field Manipulation

A.10.1 Adding or Deleting Fields

Certain fields can be added or deleted. This feature can be used to update time stamps. For this purpose it is important to know that deletion is done before addition. It is also important to know that the newly added entries are not rewritten (see next section) even so rewrite rules are applicable.

Two resources are provided to accomplish adding and deleting fields: `add.field` and `delete.field`

```
add.field = {field=value}
```

This instruction replaces the contents of the field *field* by *value* in each entry. If this field does not exist already then it is added first. The additions are applied in the sequence they are given.

value can contain formatting instructions already introduced in the section A.9.2 about “Formatting Fields” on page 32.

Suppose a time stamp is stored in the field `time`. With these resources the update of a time-stamp can be achieved using the resource instructions

```
add.field = {time=" June 13, 1997" }
```

If you want to update all time fields to contain the current date the following instruction can be used. It makes use the pseudo fields (see page 38).

```
add.field = {time="%s($mon) %s($day), %s($year)" }
```

If you want to strip the month to three leading letters and the year to two trailing digits this can be achieved with the following instruction:

```
add.field = {time="%3s($mon) %s($day), %2d($year)" }
```

The following instruction deletes all fields named *field*:

```
delete.field = {field}
```

A.10.2 Field Rewriting

To modify the value of fields can be used to optimize or normalize the appearance of a BIB_T_E_X data base. The powerful facility of regular expression matching is used for this purpose as we have already seen in section 1.2.3.

The resource `rewrite.rule` can be used to specify rewrite rules. The general form is as follows:

```
rewrite.rule { field1 ... fieldn # pattern # replacement_text }
```

field₁ ... field_n is a list of field names. The rewrite rule is only applied to those fields which have one of those names. If no field name is given then the rewrite rule is applied to all fields.

```
rewrite.rule { pattern # replacement_text }
```

Next there is the separator `'#'`. This separator is optional. It can also be the equality sign `'='`.

pattern is a regular expression enclosed in double quotes (`"`). This pattern is matched against substrings of the field value — including the delimiters. If a match is found then the matching substring is replaced by the replacement text or the field deleted if no replacement text is given.

replacement_text is the string to be inserted for the matching substring of the field value. The backslash `'\'` is used as escape character. `'\n'` is replaced by the *n*th matching group of *pattern*. *n* is a single digit (1-9). Otherwise The character following the backslash is inserted.⁷ Thus it is possible to have double quotes inside the replacement text.

Other specials are

`\$` which is replaced by the key of the current entry.

`\@` which is replaced by the type of the current entry.

⁷Future releases may use backslash followed by letters for special purposes. It is not safe to rely on escaping letters.

If no replacement text is given then the whole field is deleted. In fact the instruction `delete.field` is only an alias for a corresponding rewrite rule with an empty replacement text. This behaviour is illustrated in the following abstract examples:

```
rewrite.rule { field # pattern }
rewrite.rule { pattern }
```

More concrete, the rewrite rule

```
rewrite.rule { time # "{}$" }
```

deletes the time field if the value of the field is empty and enclosed in curly braces. This is checked with the anchored regular expression `^{}$`. The hat `^` matches the beginning of the value and the dollar `$` matches its end. Since nothing is in between—except the field delimiters—the rule is applied only to time fields with empty contents.

This can be generalized to the following rewrite rule which deletes all empty fields using the same mechanism and just omitting the specification of a field name:

```
rewrite.rule { "{}$" }
```

Note that for a similar kind of rule for double quotes as field delimiters you need to quote these characters with backslashes:

```
rewrite.rule { "{}$" }
```

The replacement text may contain field formatting instructions as described in section A.9.2 on page 32. These field formatting instructions are replaced by their respective values. Thus we could exploit again the time stamp example from above. The following rewrite rule will update an existing timestamp without adding one if none is present:

```
rewrite.rule { time ".*" = "%3s($mon) %s($day), %2d($year)" }
```

The pattern `.*` matches any sequence of arbitrary characters. Thus the old contents of the field is a match. In this example the value is not reused in the replacement text. Thus the old contents is completely replaced by the new one.

Usually the matching is done case insensitive. This means that any upper case letter matches its lower counterpart and vice versa. This behavior is controlled by the boolean resource `rewrite.case.sensitive` which is on by default. Changing this variable influences only rewrite rules specified later.

```
rewrite.case.sensitive = off
```

A problem occurs e.g. when a string is replaced by a string containing the original one. To avoid infinite recursion in such cases the numeric resource `rewrite.limit` controls the number of applications of each rewrite rule. If the number given in `rewrite.limit` is not negative and this limit is exceeded then a warning is printed and further applications of this rule are stopped. A negative value of the resource `rewrite.limit` indicates that no limitation should be used.

Next we will investigate some concrete examples. Note that in these examples the character `'␣'` denotes a single space. It is used to highlight places where spaces have to be used which would be hard to recognize otherwise.

- Empty entries are composed of delimiters — either double quotes or curly braces which enclose an arbitrary number of spaces. If we want to delete empty entries we can use the following two rules.

```
rewrite.rule {"^\_*$" }
```

```
rewrite.rule {"^{\_*$}" }
```

The caret '^' denotes the beginning of the whole string and the dollar is its end. The star is an operator which says that an arbitrary number of the preceding regular expression — i.e. the space — can occur at this point.

- Ranges of pages should usually be composed of numbers separated by an n-dash (--). The next example shows how the pages field can be normalized. Spaces are deleted and a single minus sign is replaced by a double minus.

```
rewrite.rule {pages # "\([0-9]+\)_*_*\([0-9]+\)" = "\1--\2" }
```

A.10.3 Field Ordering

Fields can be reordered within an entry. This feature is controlled by the presence of a specification for the order to use. The order is specified with the resource `sort.order`. The general form is as follows:

```
sort.order { entry = field1 # field2 # ... }
```

entry is the name of an entry like `book`. The *fields* are an arbitrary number of field names like `author`. This specification says that *field1* should precede *field2* etc. Fields which are not in this list are arranged after the specified ones. They are left in the same order as they appear in the entry.

Another possibility is to specify the entry `*`. Such a sorting order is applicable to any kind of entry. If no specific sort order is found then this general order is used if one has been specified.

Any sorting order is added to a list of sorting orders if it has not been defined before. If a sorting order is specified again, the old one is simply overwritten.

Consider the following part of a resource file:

```
sort.order = { * = author # title }
```

```
sort.order = { misc = author # title # howpublished # year # month # note }
```

This means that the `author` field goes before the `title` field in any entry type. For the `misc` entries additional specifications are made.

The library `sort fld.rsc` contains a sample sorting order for the standard entry types.

Summary

	<code>add.field{field=value}</code>	Add a new field to each entry.
	<code>delete.field{field}</code>	Delete the named field from all entries.
	<code>rewrite.case.sensitive=off</code>	Turn off the case comparison during field rewriting.
	<code>rewrite.rule{fields#pattern#text}</code>	Replace in all given fields the pattern by the replacement text.
	<code>sort.order={entry=f#...#f}</code>	Specify a preference order for fields in a given entry.

A.11 Semantic Checks

Semantic checks can be enabled in addition to the syntactic checks performed during parsing.

A.11.1 Finding Double Entries

When merging several bibliographic data bases a common problem is the occurrence of double entries in the resulting data base. When searching for double entries several problems arise. Which entries should be considered equal and what should happen to double entries. The first question is answered as follows. Two entries are considered equal if their sort key is identical and they are adjacent in the final output. The first condition of identical sort keys allows the user to specify which criteria should be used when comparing entries. This can be achieved with the resource `sort.format` (see section A.5). The second condition can easily be achieved by additionally turn on sorting when requesting checking of doubles.

It remains the question what to do with the doubles. Usually it is not desirable to keep double entries in one data base, so only one entry found is kept. The others are printed as comments, i.e. the initial '@' is replaced by '###'. Thus all information is still present but inactive in the `BIBTEX` file. However, further processing with `BIBTOOL` will remove these entries if `pass.comments` is off, which is the default.

Sometimes it is not desirable to include deleted entries in the output – not even as comments. Then the default behaviour can be changed with the boolean resource `print.deleted.entries`. If this resource is off then deleted entries are suppressed completely.

The prefix for deleted entries is stored in the resource `print.deleted.prefix` which defaults to '###'. Thus it can be redefined. However note that you should avoid using a string ending in an at sign @ since this would undo the effect of deleting an entry.

The boolean resource `check.double.delete` can be used to delete double entries completely. For this purpose it has to be turned off as in:

```
check.double.delete = ON
```

The resource `check.double` can be used to turn on the checking of doubles. This feature is turned off initially.

```
check.double = ON
```

Checking of doubles can also be turned on with the command line option `-d`:

```
bibttool -d
```

A.11.2 Regular Expression Checks

The regular expressions (see section A.6) which are used to rewrite fields (see section A.10.2) can also be used to perform semantic checks on fields. For this purpose the resource `check.rule` is provided. The syntax of `check.rule` is the same as for `rewrite.rule`.

```
check.rule { field # pattern # message }
```

Again *field* and *message* is optional. The separator `#` can also be written as equality sign (`=`) or omitted.

Each field is processed as follows. Each `check.rule` is tried in turn until one rule is found where *field* (if given) is identical to the field name and *pattern* matches a substring of the field value. If such a rule is found then the *message* is written to the error stream. If no message is given then nothing is printed and the processing of the current field is ended.

message is treated like the replacement text in `rewrite.rule`, Thus the special character combinations described in section A.10.2 are expanded.

Usually the matching is not done case sensitive. This means that any upper case letter matches its lower counterpart and vice versa. This behavior is controlled by the boolean resource `check.case.sensitive` which is ON by default. Changing this variable influences only rewrite rules as described in section A.10.2.

```
check.case.sensitive = off
```

Consider the following example. We want to check that the year field contains only years from 1800 to 1999. Additionally we want to allow two digit abbreviations.

```
check.rule { year "^[\"]{1}[89][0-9][0-9][\"]$" }
```

```
check.rule { year "^[\"]{0}[0-9][0-9][\"]$" }
```

```
check.rule { year "" "\@ \$: Year has to be a suitable number" }
```

The first rule matches any number starting with 1 followed by 8 or 9 and finally two digits. The whole number may be enclosed in double quotes or curly braces.⁸ The hat at the beginning and the dollar at the end force that the pattern matches against the whole field value only.

The next rule covers years consisting of two digits. The first two rules produce no error message but end the search for further matches. Thus if something suitable is found then one of the first two rules finds it.

Otherwise we have to produce an error message. This is done with the third rule. The empty pattern matches against any value of the year field. This rule is only applied if the preceding rules do not match. In this case we print an error message. `\@` is replaced by the current type and `\$` by the current key.

⁸In fact the regular expression allows also strings starting with a quote and ending in a curly brace. But this syntactical nonsense is ruled out by the parser already.

Summary

	<code>check.case.sensitive=off</code>	Perform semantic checks case sensitive.
<code>-d</code>	<code>check.double=on</code>	Find and mark or delete entries with identical sort keys.
	<code>check.double.delete=on</code>	Delete double entries instead of deactivating them.
	<code>check.rule{field#pattern#msg}</code>	If the value of field matches pattern then print the given message.

A.12 Strings — also called Macros

Strings in `BIBTEX` files play an important role when managing large bibliographic data bases. Thus they deserve special treatment. If the resource `macro.file` is defined then the macros are written to this file. The argument is a file name as in

```
macro.file = {macro/file/name}
```

Note that the reverse operation to string export namely the import of strings does not deserve a special treatment. You can simply give the macro file as one of the input files—preferably before any input file that makes use of one of the macros contained therein. The boolean resource `print.all.strings` indicates if all macros defined in the `BIBTEX` file should be printed or only those macros actually used.

```
print.all.strings = ON
```

The appearance of string names is controlled by the resource `symbol.type` (see 21). Strings can be expanded when printing entries. This feature of `BIBTOOL` is controlled by the resource `expand.macros` as in

```
expand.macros = ON
```

The effect is that all known strings in normal entries are replaced by their values. If the values are not defined at the time of expansion then the macro name remains untouched. As a side effect string concatenations are simplified. Imagine the following `BIBTEX` file.

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

@Book{ almanac-66,
       title = 1967 # WGA,
       month = "1~" # jan
}
```

If `BIBTOOL` is applied with `expand.macros` turned on this results in the following output — if the default settings are used for every other resource.

```
@STRING{wga   = " World Gnus Almanac" }

@Book{
  almanac-66,
  title   = {1967 World Gnus Almanac},
  month   = {1~} # jan
}
```

The macro `WGA` has been expanded and merged with `1967`. Note that the string `jan` has not been expanded since the value should be defined in a `BIBTEX` style file (`.bst`).

When macros are expanded the delimiters of entries is normalized, i.e. only one style is used. In this example braces have been used. The alternative would be to use double quotes. This behavior is controlled by the resource `print.braces`. If this resource is on then braces are used otherwise double quotes are taken. It can be changed like in

```
print.braces = OFF
```

The delimiters of the whole entry are recommended to be braces. For compatibility with Scribe it is also allowed that parentheses are used for those delimiters. This behavior can be achieved with the boolean resource `print.parentheses`. Initially this resource is off. It can be set like in the following instruction:

```
print.parentheses = ON
```

Summary

	<code>macro.file={file}</code>	Write the macro definitions to the file <i>file</i> .
	<code>print.all.strings=off</code>	Print only those macro definitions which are used instead of all.
	<code>expand.macros=on</code>	Turn on macro (string) expansion in fields.
	<code>print.braces=off</code>	Switch to the use of quotes for expanded macros instead of braces.
	<code>print.parentheses=on</code>	Enclose the whole entry in parentheses instead of braces.

A.13 Statistics

Some information can be obtained at the end of a `BIBTOOL` run. The number of `BIBTEX` items read and written is printed. To enable this feature the resources `count.all` and `count.used` are provided.

```
count.all = ON
```

```
count.used = ON
```

`count.all` indicates that all known types of `BIBTEX` items should be listed. `count.used` forces only those types of `BIBTEX` items to be listed which have been found in the input files.

Summary

	<code>count.all=on</code>	Print statistics about all known entry types.
	<code>count.used=on</code>	Print statistics about the used entry types only.

B

Limitations

B.1 Limits of BIBTOOL

BIBTOOL has been written with dynamic memory management wherever possible. Thus BIBTOOL should be limited by the memory available only.

B.2 Bugs and Problems

Problems currently known are the following ones. They are not considered to be bugs.

- The referencing feature of `BIBTEX` is not supported. `\cite` macros can be contained in fields (e.g. notes). Such things can be confused.
- The memory management uses dynamic memory. This memory is reused but not returned to the operating system. Thus BIBTOOL may run out of memory even if a more elaborated memory management may find free memory. This is a design decision and I don't think that I will change it.
- The `TEX` reading apparatus is only imitated to a certain limit. But this should be enough for most applications to produce satisfactory results.
- In several modules ASCII encoding is assumed. I do not know to which extend this influences the functionality since I don't have access to non-ASCII machines.
- Macro expansion uses a dynamic array which can turn out to be too short. This will be corrected as soon as I have an example where this bug shows up.

The distribution of BIBTOOL also contains a file named `ToDo`. If you are interested in more detailed descriptions of possible problems, limitations, and ideas for improvements in further releases then you can have a look at the contents of this file.

C

Sample Resource Files

Sample resource files are included in the distribution of BIBTOOL in the directory Lib. Only some of them are reproduced in this section.

C.1 The Default Settings

```
bibtex.env.name      = "BIBINPUTS"
check.case.sensitive = on
check.double         = off
check.double.delete  = off
count.all            = off
count.used           = off
crossref.limit       = 32
default.key          = "***key*"
dir.file.separator   = "/"
env.separator        = ";"
expand.macros        = on
fmt.et.al            = ".ea"
fmt.inter.name       = "-"
fmt.name.name        = "."
fmt.name.pre         = "."
fmt.name.title       = ":"
fmt.title.title      = "-"
ignored.word         = "a"
ignored.word         = "an"
ignored.word         = "the"
ignored.word         = "le"
ignored.word         = "les"
ignored.word         = "la"
ignored.word         = "un"
ignored.word         = "une"
ignored.word         = "el"
ignored.word         = "il"
ignored.word         = "der"
ignored.word         = "die"
ignored.word         = "das"
ignored.word         = "ein"
```

```

ignored.word          = "eine"
key.base              = lower
key.expand.macros     = on
key.format            = short
key.generation        = off
key.number.separator  = "*"
new.entry.type        = "Article"
new.entry.type        = "Book"
new.entry.type        = "Booklet"
new.entry.type        = "Conference"
new.entry.type        = "InBook"
new.entry.type        = "InCollection"
new.entry.type        = "InProceedings"
new.entry.type        = "Manual"
new.entry.type        = "MastersThesis"
new.entry.type        = "Misc"
new.entry.type        = "PhDThesis"
new.entry.type        = "Proceedings"
new.entry.type        = "TechReport"
new.entry.type        = "Unpublished"
preserve.keys         = off
preserve.key.case     = off
print.align           = 18
print.align.string    = 18
print.align.preamble  = 11
print.align.comment   = 10
print.align.key       = 18
print.braces          = on
print.comma.at.end    = on
print.all.strings     = on
print.deleted.prefix  = "###"
print.deleted.entries = on
print.equal.right     = on
print.indent          = 2
print.line.length     = 77
print.newline         = 1
print.parentheses     = off
print.use.tab         = on
print.wide.equal      = off
rewrite.case.sensitive = on
rewrite.limit         = 512
quiet                 = off
select.case.sensitive = off
select.field          = "$key"
sort                  = off
sort.format           = "$key"
sort.macros           = on
sort.reverse          = off
symbol.type           = lower
verbose               = off

```

C.2 Useful Translations

The resource file `tex_def` translates international characters into plain text representations. Especially the German umlaut sequences are translated. E.g. the letter `Ä` which is written as `{\ "A}` in a `BIBTEX` file is translated to `Ae`.¹

¹Note that the notation of `german.sty` is not understood by `BIBTEX` nor by `BIBTOOL`.

Additionally some logos are defined.

```
tex.define {\ "[1]=#1e}
tex.define {\ ss=ss}
tex.define {\ AE=AE}
tex.define {\ OE=OE}
tex.define {\ aa=aa}
tex.define {\ AA=AA}
tex.define {\ o=o}
tex.define {\ O=O}
tex.define {\ l=l}
tex.define {\ L=L}
tex.define {\ TeX=TeX}
tex.define {\ LaTeX=LaTeX}
tex.define {\ LaTeXe=LaTeX2e}
tex.define {\ BibTeX=BibTeX}
tex.define {\ AMSTeX=AMSTeX}
```

C.3 Other Resource Files

The distribution contains additional resource files. Some of them are sketched here. Others may be contained in the distribution as well. Look in the appropriate directory.

`iso2tex`

define rewrite rules to translate ISO 8859-1 characters into $\text{BIB}_{\text{T}}\text{E}_{\text{X}}$ compatible sequences.

`iso_def`

define macro equivalents for ISO 8859-1 characters into $\text{T}_{\text{E}}\text{X}$ compatible sequences.

`sort_fld`

defines a sort order for the common $\text{BIB}_{\text{T}}\text{E}_{\text{X}}$ entry types.

`check_y`

contains a sample for semantic checks. The year field is checked to be a suitable number.

`month`

tries to introduce $\text{BIB}_{\text{T}}\text{E}_{\text{X}}$ strings for month names. Provisions are made to preserve other information contained in the month field.

`opt`

copies with `OPT` prefixes as introduced e.g. by `bibtex-mode`.

Bibliography

- [GMS94] Michel Goosens, Frank Mittelbach, and Alexander Samarin. *The L^AT_EX companion*. Addison-Wesley Publishing Company, 1994.
- [Knu89] Donald E. Knuth. *The T_EXbook*. Addison-Wesley Publishing Company, 15th edition, 1989.
- [Lam94] Leslie Lamport. *L^AT_EX: A Document Preparation System*. Addison-Wesley Publishing Company, 2nd edition, 1994.
- [Pat88a] Oren Patashnik. *BIB_TE_Xing*, 1988.
- [Pat88b] Oren Patashnik. *Designing BIB_TE_X Styles*, 1988.

Index

- 7, 10, 11, 16, 26
- add.field 45, 46, 49
- BIBINPUTS 17
- bibtex.env.name 17
- bibtex.search.path 17
- BIBTOOL 14
- BIBTOOLRSC 14
- check.case.sensitive 50, 51
- check.double 49, 51
- check.double.delete 49, 51
- check.rule 50, 51
- count.all 52
- count.used 52
- crossref.limit 38
- d 49, 51
- default.key 27, 29, 31, 39
- delete.field 45–47, 49
- digit 29, 30
- dir.file.separator 17
- empty 28, 29
- env.separator 17
- expand.macros 51, 52
- extract.file 25, 26
- extract.regex 26
- extract.regexp 26
- F 28, 31
- f 8, 26, 31
- fmt.et.al 30, 31, 39
- fmt.inter.name 30, 31, 39
- fmt.name.name 30, 31, 39
- fmt.name.pre 30, 31, 39
- fmt.name.title 27, 30, 31, 39
- fmt.title.title 30, 31, 39
- h 13, 16
- HOME 14
- i 16, 17
- ignored.word 35, 37
- input 16
- input.file 17
- K 8
- k 8
- key.base 27, 29–31
- key.expand.macros 30, 31
- key.format 26, 29, 31
- key.generation 28, 31
- key.number.separator 29–31
- long 27, 28, 30
- lower 29, 30
- macro.file 51, 52
- name 14, 15
- new.entry.type 19, 21
- new.field.type 20, 21
- new.format.type 42, 45
- new.long 28
- new.short 28
- o 10, 18
- off 15, 49
- off 15
- on 15, 47
- output.file 18
- pass.comments 19, 21, 49
- preserve.key.case 21, 29, 31
- preserve.keys 28, 29, 31
- print 15, 16
- print.align 19–21
- print.align.comment 19, 21
- print.align.key 19–21
- print.align.preamble 19
- print.align.string 19–21
- print.all.strings 25, 51, 52
- print.braces 52
- print.comma.at.end 20, 21
- print.deleted.entries 49
- print.deleted.prefix 49
- print.equal.right 20
- print.indent 19–21

print.line.length 19–21
print.newline 20
print.parentheses 52
print.print.newline 21
print.use.tab 20, 21
print.wide.equal 20, 21

-q 18
quiet 18

-R 14, 16
-r 11, 13, 15, 16
resource 15, 16
resource.search.path 14, 16
rewrite.case.sensitive 47, 49
rewrite.limit 47
rewrite.rule 46–50

-S 7, 22
-s 7, 22
select 25, 26
select.case.sensitive 25, 26
select.fields 26
short 27, 28, 30
sort 22
sort.format 22, 49
sort.macros 22
sort.order 48, 49
sort.reverse 22
symbol.type 21, 51

t 15
tex.define 41, 42, 45
true 15

upper 29, 30

-v 18
verbose 18

-X 10, 25, 26
-x 10, 25, 26

yes 15