Bib2x Templates

Alexander Feder
i@xandi.eu

June 5, 2006

Contents

1 Bib2x-Templates

When it comes to lay total control into the hands of the user to define how the result of a conversion shall look like, there is no way around templates. Further, if these templates allow a basic set of operations like loops and constraints, and a set of commands to get and modify properties almost arbitrary output can be generated. *„Almost"* because it is restricted to files that can be represented by plain text. Thus, a binary file cannot serve as a template. Due to restrictions of Lex, the character-set of the input is further restricted to Utf-8. Please note that you can define the output to contain Utf-16 characters.

1.1 Basic Structure

Anything enclosed by two `%%` will be processed as an operation or command. Some commands are followed by a list of constraints and conditions, which are defined between a pair of parentheses. Everything else can normally be considered as part of the output-file, and will be kept unchanged. Every whitespace and every newline is recognized, every special character apart from `%` can be used. `%` needs to be escaped using `\%`.

1.2 Defintions

Bib2x understands a few commands, that it can process. In many markup-languages these commands can be converted very easily by just enclosing the contained text with an appropriate pair of tags provided by the markup-language. The user can then decide, how this may look like – it does not necessarily have to make any sense.

For instance, the LaTeX-command `\emph{important text}` could be defined to look like

```
--->>> important text <<<---
```

or

```
<span style="color:red"><b>important text</b></span>
```

in the target-file.

This definition is done using `%%def%%` followed by the index of the enclosing which can be looked up in `latexconv.h`. (This won't be necessary in future versions of Bib2x.) After that, between a pair of quotation-marks the desired string can be set. Please note that this time, the quotation-mark needs to be escaped using `\"`, `%` can be contained without being escaped. For example:

```
%%def%% 0001 "<span style=\"color:red\"><b>"
%%def%% 0002 "</b></span>"
```

defines the `\emph`-conversion given as an example above. If the markup-language or format-definition used does not need enclosings but rather a single command, an empty closing-string can be defined.

1.3 Looping, Grouping and Reducing

Before the first line of the template is being processed, all BibTeXentries are stored in memory as a set waiting to be filtered, grouped and processed. See figure 1 on page 5 for an *artist's impression* of the stages of operation. There are three template commands that can operate on this set.

- `%%GRP%%` to GROUP. The outcome can be understood as multiple sets, that are looped through. Further simply called GROUP. `%%GRP%%` marks the begin of a body that has to be closed using `%%PRG%%`. See section 1.3.1 on page 6 for what is valid inside the GROUP's body.
- `%%FOR%%` to loop through the given set – every single member of the set is processed here. Further simply called FOR. Everything between `%%FOR%%` and its closing tag `%%ROF%%` is the loop's body – see section 1.3.2 on page 7 for detailed explanation.
- `%%RED%%` to REDUCE. The outcome is again a single set. Further simply called RED.

These three commands share the same syntax for describing conditions and constraints. These are given inside the enclosing pair of parentheses as follows:

```
%%CMD%% ( Sort: Condition; Constraint; Constraint; ...; )
```

SORT

This defines the direction how the sort- or group-operation shall be performed. The two options are:

- `inc` for ascending order
- `dec` for descending order

CONDITION

There can be at most one condition, which defines what the criteria for sorting or grouping may be.

There are two types of conditions:

- The name of the tag to be used as key.

- The word `@entry` to specify, that grouping or sorting should follow the BibTeX-entry-type. (@article, @book, etc.)

A condition can be understood as a constraint without arguments. If there is no condition specified, the first constraint will be considered as the command's condition.

Constraints

Conditions follow a simple syntax:

tag:nr operator {"word","more words",...}

- `tag` is the tag to be operated on.

- `:nr` is optional and can specify which component of the tag is considered in the operation. Possible values are:

  `0` : operation will perform on a concatenation of all substrings.
  `x` : each substring is operated on separately.
  `k` : $k \in \mathbb{N}$ – operation will only perform on k^{th} substring.

- `operator` can be one of the following:

  `=` : tag contains at least one of the specified words.
  `!=` : tag does not contain any of the specified words.

Note: the set of words enclosed by braces can be understood as logical OR whereas subsequent constraints can be understood as logical AND.

For example:

```
1 %%FOR%% ( dec: author; author = { "alexander" };
2           author!={"bob","john"}; )
3 %%FOR%% ( inc: category:2={"selfmade","nonsense"};
4           title!={"interesting"}; )
```

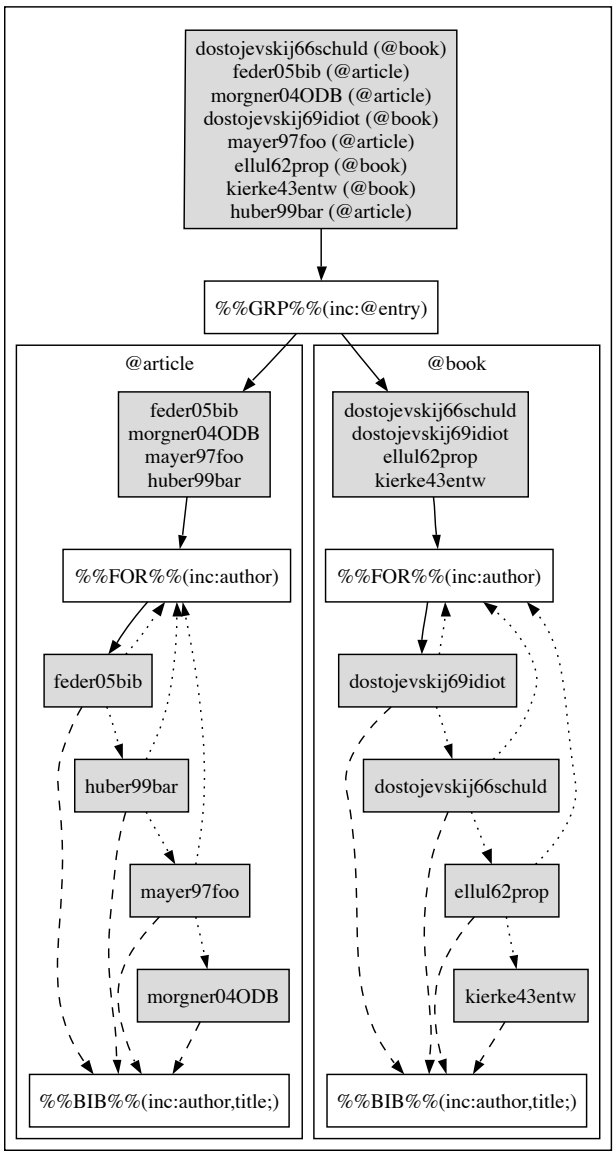

Figure 1: An attempt to visualize how a template is processed with a set of entries. The box above represents the unfiltered starting set. It is then processed by %%GRP%% with the condition to group by entries. Thus it will loop two times, as only articles and books are stored in the set. Inside these loops, a %%FOR%% processes the set, with the condition to sort by author thus starting with the author whose name comes first in the alphabet. Each entry will then be processed by a %%BIB%% which extracts the tags, starting with author followed by title; then, all remaining tags are processed in ascending order.

1.3.1 Group

Group splits the previous set into subsets which then are separately processed according to the body of the Group. The following commands are valid inside a Group:

- %%name%% returns the name of the current subset, which results from the condition grouped by. If grouped, e.g., by author, %%name%% will return a name (or names), if grouped by year %%name%% will return the subset's corresponding year.

- %%for%% to process through all entries of the resulting subset.

- %%grp%% to construct arbitrarily nested Groups, which will lead to further subsets.

Following example shows nested Groups:

```
<html><body>%%GRP%%  (inc: author)<div style="...">
Works by <b>%%name%%</b> were published in
<ul>%%GRP%%(inc:year)<li style="color:blue">%%name%%</li>
%%PRG%%</ul></div><br>
%%PRG%%</body></html>
```

Note that any whitespace will be ignored between the command and its conditions, as well as between the conditions. After the closing parenthesis though, everything is regarded as content. This example will result in something that looks like this:

```
<html><body><div style="...">
Works by <b>Alexander Feder</b> were published in
<ul><li style="color:blue">2004</li>
<li style="color:blue">2005</li>
</ul></div><br>
...
```

1.3.2 For

For iterates through the current set of entries, and processes each entry according to the body of the For. Following commands are valid inside the body of For:

- `%%$odbid%%` returns the numeric Odb-Id of the current entry.
- `%%$bibkey%%` returns the BibTeXcitation key.
- `%%$bibtype%%` returns the BibTeXentry type (@book, @inproceedings, etc.)
- `%%anytag%%`, where anytag is a placeholder for the tag to be returned, for instance, `%%author%%` for the author(s). This results in a concatenation of all substrings. If a specific substring is needed, `%%BIB%%`'s `%%step%%`-command can be used.
- `%%BIB%%` enables conditioned access to all tags without prior knowledge of their existence. See section 1.4.

1.3.3 Reduce

Reduce can be used to further filter through the set of entries using constraints. The constraints are to be applied in the same way described for Group and For. Reduce is used to completely get rid of entries in the specific hierarchy, which makes it possible to let multiple Fors or Groups inside the same hierarchy use the same data set.

1.4 Processing a single entry with %%BIB%%

It often is desirable to process tags previously unknown, and tags expected but not existing. Further, having the ability of specifying the order in which these tags shall be processed and if or under which conditions tags may be skipped or uniquely included.

For this purpose `%%BIB%%` can be used. It iterates through all tags and, by default, returns them all. This can be conditioned and filtered. The syntax of `%%BIB%%` is different to the syntax of `%%FOR%%`, but admittedly confusingly similar:

```
%%BIB%% ( Sort: Sequence; Constraint; Constraint; ...; )
```

Sequence is a comma-separated list of tags. These tags can have a prefix:

`!` : excludes a tag. No operation will be performed on this tag.

`#` : makes the tag unique. No further tags will be included unless also specified using `#`, not explicitly included tags will no longer be processed, once a single tag has been prefixed with `#`.

Tags supplied with no prefix define the order in which they will be processed. Unspecified tags will be processed nonetheless afterwards, except if excluded later.

Constraints

See section 1.3 on page 3 for the syntax of constraints.

1.4.1 Body

The body of a `%%BIB%%` is enclosed by a pair of `%%BIB%%`-tokens and can be divided into four functional different parts, although at least one is mandatory. The parts are defined by one of four letters described below, followed by an `=` and an arbitrarily long string – enclosed by a pair of quotation-marks – which may include any character except unescaped quotation-marks. The following letters can be used to identify a part

`M` : Missing – this string will be processed in case a *specified* tag has not been found.

`E` : Empty – in case the specified tag has been found but is empty.

`N` : Nonempty – for specified tags having content.

`G` : General – for *non*-specified tags having content.

The following commands are valid inside the string of the parts:

- `%%name%%` – returns the name of the current tag.
- `%%content%%` – returns a concatenation of the current tag's substrings.
- `%%step:#-#%%` – for getting a range of the current tag's content, detailed explanation of Step follows.
- `%%MOD_XX%%` – for modifying a returned string or substring. See Modifications.

1.4.2 Step

The syntax for Step is as follows:

```
%%step:#-#%% ( "Prefix"; "Middle"; "Suffix" )
```

where `#` stands for a numeric value *or* the letter `x`, which is either begin or end of all substrings. (for instance, `0-x` gives the concatenation of all substrings). The prefix is put at the beginning; every pair of substrings is separated with *,,Middle"*. To the last substring the suffix will be added.

For example:

```
%%BIB%%(inc:!author,!title,note;)
G="%%name%%: %%step:0-x%%("\"";"\" # \"";"\",")"
M="%%name%%: !! MISSING !!"
E="%%name%%: ?? EMPTY ??"
N="%%name%%: \"%%content%%\""
%%BIB%%
```

The following statements are equal:

```
%%step:0-x%%("<b>"; ""; "</b>")
<b>%%content%%</b>
```

1.4.3 Modifications

Bib2x allows for modifications to be made onto strings returned inside the body of a %%BIB%%. %%MOD_XX%% takes another command as parameter:

%%MOD_XX%% (%%CMD%%)

XX stands for the type of modification, of which currently the following forms exist:

- LC : LowerCase – everything (except those parts embraced, see section **??** on page **??**) is converted to lower-case letters.
- LF : Lowercase, Firstupper – everything is converted to lower-case letters except the very first one.
- UC : UpperCase – everything is converted to upper-case letters.
- UB : Uppercase Begin – every word starts with an upper-case letter.

For instance:

```
%%MOD_LC%% ("A very interesting Text about {BiBTeX}")
%%MOD_LF%% ("A very interesting Text about {BiBTeX}")
%%MOD_UC%% ("A very interesting Text about {BiBTeX}")
%%MOD_UB%% ("A very interesting Text about {BiBTeX}")
```

results in

```
a very interesting text about BiBTeX
A very interesting text about BiBTeX
A VERY INTERESTING TEXT ABOUT BiBTeX
A Very Interesting Text About BiBTeX
```

2 Examples

In this section, some examples of templates and their corresponding results will be demonstrated. It is intended to give an idea of what can be done using Bib2x and give practical insight into the usage of the template language. These templates are further included in the Bib2x-distribution.

2.1 Html

Html, especially in combination with Css is a great way to easily achieve good looking results very quickly while the structure is simple and flexible. Following example defines a list of entries grouped by author. As can be seen, header information will stay untouched, only the sections inside of loops are dynamically generated depending on the entries.

```
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/transitional.dtd">
<html>
<head>
<TITLE>xandi's BibTeX collection</TITLE>
<link rel="stylesheet" href="bib.css" media="screen"
  type="text/css" />
<link rel="stylesheet" href="bib.css" media="print"
  type="text/css" />
<link rel="stylesheet" href="bib.css" media="projection"
  type="text/css" />
</head>

<body>
<div id="BibTOC">
%%FOR%%(dec:author)
     <a href="#%%$bibkey%%">%%$bibkey%%</a><br>
%%ROF%%
<br></div><br>

%%GRP%%(inc:author)
<h1 class="myheading">%%name%%</h1>
%%FOR%%(inc:title)
<div class="BibEntry">
     <a class="EntryGoto" id="%%$bibkey%%"></a>
<span class="EntryType">%%$bibtype%%: </span>
<span class="EntryKey">%%$bibkey%%</span><br>


<table class="EntryTable">
<tr class="TagRow">
```

```
33      <td class="NameCollumn">
34           <div class="EntryTableInfo">Tag</div>
35      </td>
36      <td class="ContentColumn">
37           <div class="EntryTableInfo">Content</div>
38      </td>
39 </tr>
40
41 %%BIB%%(inc:author,title,year;)
42 N = "<tr class=\"TagRow\">
43      <td class=\"%%name%%\" class=\"NameCollumn\">
44           <div class=\"NameText\">%%name%%</div></td>
45      <td class=\"%%name%%\" class=\"ContentColumn\">
46           <span class=\"ContentText\">
47                <span class=\"Text_word\">%%content%%</span>
48           </span>
49      </td></tr>
50 "
51 G = "<tr class=\"TagRow\">
52      <td class=\"%%name%%\" class=\"NameCollumn\">
53           <div class=\"NameText\">%%name%%</div></td>
54      <td class=\"%%name%%\" class=\"ContentColumn\">
55           <span class=\"ContentText\">
56                <span class=\"Text_word\">%%content%%</span>
57           </span>
58      </td></tr>"span></td>
59          </tr>
60 "
61 %%BIB%%
62 </table></div><br><br>
63
64 %%ROF%%
65
66 %%PRG%%
67 <br>
```

Calling BIB2x

```
1 $ bib2x -f test.bib -t html.template > test.html
```

will result in a pretty-printed HTML-file which can be gazed at on page 13 in figure 2.

- Line 16 to line 20 define the TOC[1] which can be seen in figure 2 in the upper-left-hand corner. Position and color is defined using the cascading style-sheet which

[1] Table of Contents

is listed on page 18 in section 2.4.

- Between line 22 and 66, the GROUP is defined. In figure 2 a group named *Alexander Feder* containing two tables can be seen.
- Each table inside this group is defined between Line 24 and 64 using FOR.
- The table's rows are defined in the lines 41 to 61, using BIB. Here, two parts come to use, one defining NONEMPTY expected tags, which should be author, title and year. If they happen to be missing, nothing will happen, as nothing for that case is defined (a M-part would be required for that). The other defined part is GENERIC, which will be used for all other tags, not specified explicitly.

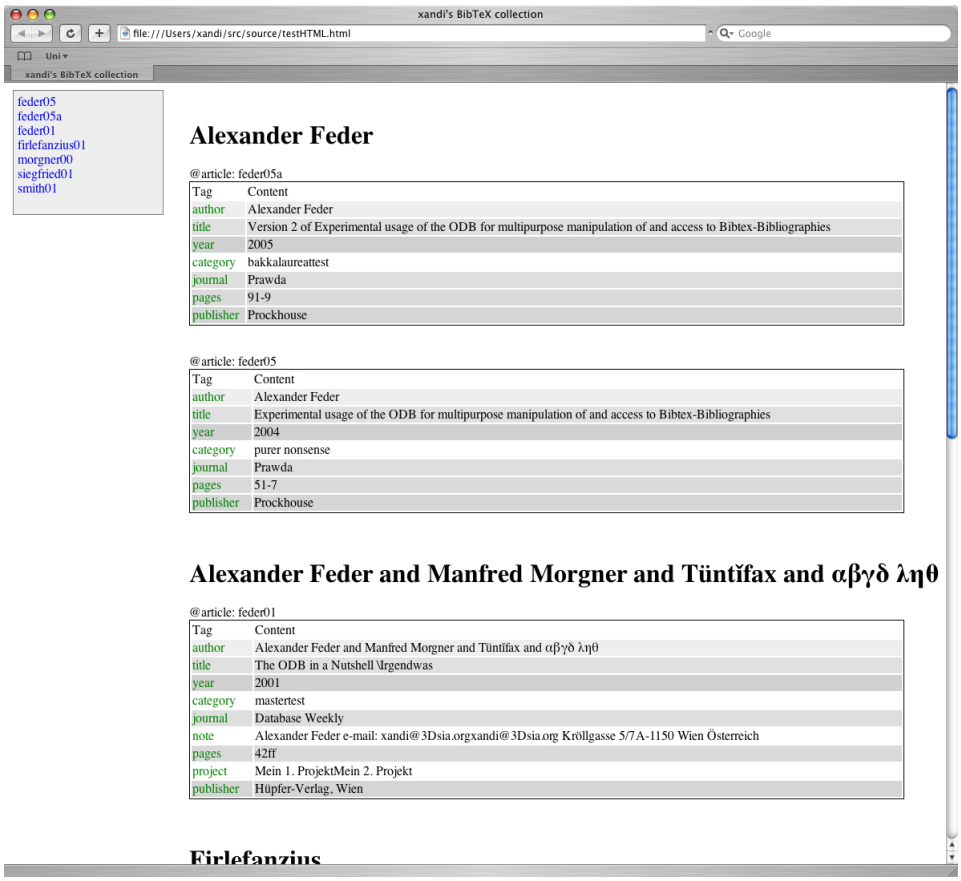

Figure 2: Output of the Html-Template

2.2 Rtf

Following example has been created primarily as a proof of concept, to show that any human readable file format can be used. Please note the handling of special characters. Bib2x does not yet support this kind of notation, thus Utf-characters cannot be displayed properly yet. Here, works of authors are grouped together, and are further grouped by year. This demonstrates the ability to created nested groups.

```
1  {\rtf1\mac\ansicpg10000
2
3  %%GRP%% ( inc : author )
4
5  \cf0 Haupt-Kategorie: %%name%%\
6  %%GRP%%(inc:year)
7  Unter-Kategorie: %%name%%\
8  %%FOR%%(inc:title)
9  %%$bibtype%%: %%$bibkey%%\
10 \pard\ql\qnatural\pardirnatural
11 \ls1\ilvl0\cf0 %%BIB%%(inc:author;)
12 G = "{\\listtext \\'a5 }%%MOD_UB%%(%%name%%): %%content%%\\
13 "
14 M = ""
15 E = ""
16 N = "{\\listtext \\'a5 }%%MOD_UB%%(%%name%%): %%content%%\\
17 "
18 %%BIB%%\pard\ql\qnatural\pardirnatural
19 \cf0 \
20 %%ROF%%\
21 %%PRG%%
22 %%PRG%%}
```

Calling Bib2x

```
1  $ bib2x -f test.bib -t rtf.template > test.rtf
```

will result in a Rtf-file which can be seen on page 15 in figure 3.

- Line 3 starts the first, line 6 the second (nested) group.
- Between line 11 and 18 the %%BIB%% defines all four parts. Note, that it makes no difference to describe an empty part using two quotations-marks with nothing in between or not defining the part at all - in both cases, tags fitting the criteria covered by that part will not form the resulting target file.
- Line 12 shows the usage of the modification-command %%MOD_UB%% which is explained in section 1.4.3 on page 9.

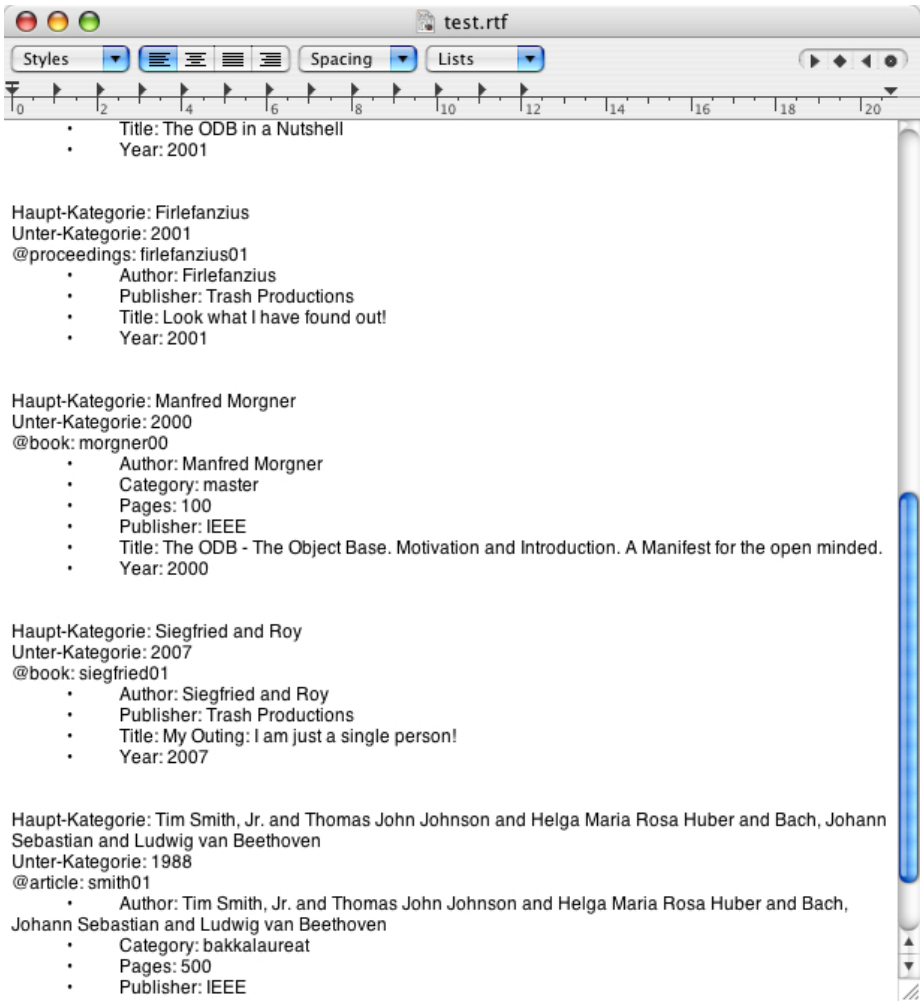

Figure 3: Output of the Rtf-Template

2.3 BibTeX

This example is intended to show that Bib2x can also be used to clean and prettyprint BibTeX files, e.g. add structure and comments to ease editing by a human. Further, it can be used to apply some modifications to it.

```
1  \%\%\% BibTeX-File
2  \%\%\% generated using Bib2x
3
4  @string { missing, "Missing!" }
5
6  %%GRP%%(inc:author;year!={"2001"};)
7  \%\%\%\%\%\%\%\%\%\%\% ... \%\%\%\%\%\%\%\%\%\%\%\%\%\%\%\%\%
8  \%\%\% Works by %%name%%
9  \%\%\%\%\%\%\%\%\%\%\% ... \%\%\%\%\%\%\%\%\%\%\%\%\%\%\%\%\%
10 %%FOR%%(inc:title)
11 %%$bibtype%%{%%$bibkey%%,
12 %%BIB%%(inc:author,title,year,!category;)
13 G = "  %%name%%  = \"%%content%%\",
14 "
15 N = "  %%name%%  = \"%%content%%\",
16 "
17 E = "  %%name%%  = \"\",
18 "
19 M = "  %%name%%  = missing,
20 "
21 %%BIB%%%%BIB%%(inc:#category;)
22 N = "  %%name%%  = %%step:0-x%%("\"";"\" # \"";"\",")
23 "
24 E = "  %%name%%  = \"\",
25 "
26 %%BIB%%}
27 %%ROF%%%%PRG%%
```

Calling Bib2x

```
1  $ bib2x -f test.bib -t bib.template > converted.bib
```

will result in a BibTeX-file which can be seen on page 17 in figure 4.

- Line 7 to 9 demonstrate BibTeX-comments, which result in a clean structure that can be seen in figure 4 on page 17. Note that `%` needs to be escaped by `\%`.
- Line 12 shows the inclusion and exclusion of certain tags. *category* is excluded to be processed between line 21 and 26 explicitly, indicated by using `#` as prefix.
- Line 19 shows the usage of the Missing-Part. Missing tags will be marked with the string defined in line 4.

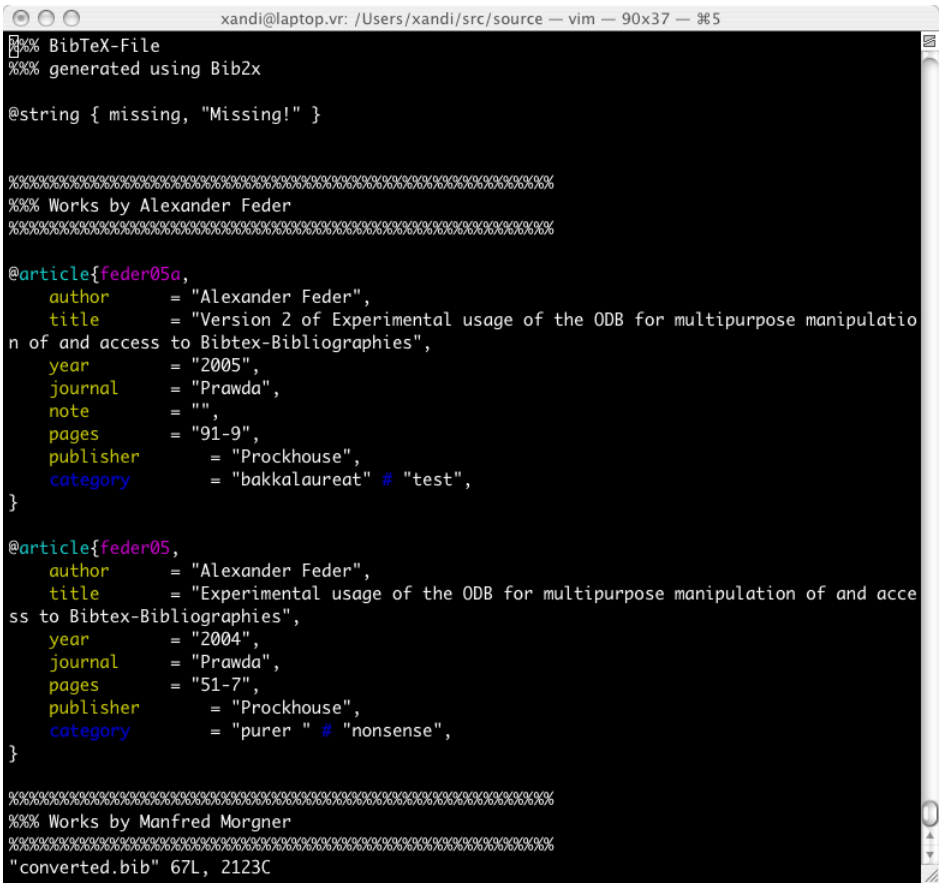

Figure 4: Output of the Bib-Template

2.4 CSS

What follows is the Css used in section 2.1 on page 11.

```
body { height: 95%; width: 95%;
    background:white; }
#BibTOC { position: absolute;
    background: #eeeeee;
    top: 1%; left: 1%; width: 15%;
    border: thin solid gray;
    text-align: left; padding: 5px; }
h1.myheading { position: relative; left: 20%; }
table.EntryTable { position: relative;
    left: 20%; width: 80%;
    border:thin solid black;
    border-spacing:10px
    font: 15px Impact, Helvetica Narrow, sans-serif; }
span.EntryType { position: relative; left: 20%; }
span.EntryKey { font: bold;
    position: relative; left: 20%; }
td.author { background: #eeeeee; }
td.title { background: #e0e0e0; }
td.journal { background: #dedede; }
td.pages { background: #dadada; }
td.publisher { background: #d5d5d5; }
td.year { background: #d0d0d0; }
td.url { background: #cdcdcd; }
div.NameText { font: bold; color: green; }
div.NameTextMissing { font: bold;color: red; }
div.ContentText { font: bold; }
span.Text_math{ color: gray; }
span.Text_missing{color: red;}
span.Text_path{color: blue;}
span.Text_latex{color: orange;}
span.Text_accent{color: magenta;}
span.Explizit{color: green;}
```