The `bibexport.sh` script

Nicolas Markey

2009/10/11

Abstract

`bibexport.sh` is a small shell script, relying on BibTEX, that extracts entries of one or several `.bib` file(s). It will expand abbreviations and cross-references, except standard month and journal abbreviations. The output is indented as neatly as possible, yielding a readable `.bib` file even if the original one was not.

1 Exporting `.bib` files

1.1 Why and how?

BibTEX aims at allowing for the use of one single `.bib` file, containing many entries, from which BibTEX extracts only the `\cite`d ones. When sending a document to someone else, this requires either to send the whole file, or to extract some entries from the `.bib` file.

BibTEX also has a mechanism for using abbreviations and cross-references. When extracting entries of a large `.bib` file, it can be interesting to develop those abbreviations, in order to get a clean, self-contained `.bib` file. Also, it may be useful to develop cross-references in a `.bib` file, independently of any document.

`bibexport` can either extract entries that are cited in a document, or all the entries of one or several `.bib` files. It will always develop cross-references and abreviations, except standard abbreviations for months or some journals, that are defined in standard BibTEX styles. This script uses BibTEX. This has both pros and cons:

+ it is very simple. Basicaly, the script simply calls BibTEX, and the `.bst` file just outputs the name and the content of each field.

+ since it uses BibTEX, we are sure that it will handle everything "properly", *i.e.* in the same way as they will be handled when cited in a LaTEX document;

= BibTEX has some strict limitations (especially "no more than 78 consecutive non-space characters") that we must be aware of. On the other hand, any such problem occuring within the script would also occur when compiling a document;

– abbreviations and cross-references will *always* be developped. It could be argued that this is also a positive point, but having the choice would have been better.

– Many people seem to find BibTEX's internal language clumsy, and thus the script could be difficult to adapt to special needs. However, this is *really* not that difficult, as will be explained later on. And please, don't hesitate to ask for improvements.

1.2 Related scripts

Several other tools exist for achieving this task:

- `aux2bib`, written by Ralf Treinen, relies on `bib2bib`, which is a CAML program for selecting some entries in one or several `.bib` files. It does not expand anything, but includes all the necessary definitions and entries.
- `bibextract.sh`, by Nelson Beebe. This script uses AWK for extracting some entries out of a `.bib` file. It is said to be noncompliant with cross-references.
- subset.bst, by David Kotz. export.bst develops the same ideas (but I discovered that only later on). subset.bst does not handle `@preamble`, neither does it "protect" standard abbreviations.
- Probalby some others...

1.3 Some examples

- extracting `\cite`d references of a document, also including cross-references:

  `bibexport -o <`*result*`>.bib <`*file*`>.aux`

- extracting `\cite`d references of a document, without crossrefs, and using a special `.bst` file:

  `bibexport -b <`*style*`>.bst -o <`*result*`>.bib <`*file*`>.aux`

- export all the entries of two `.bib` files (including crossrefed entries):

  `bibexport -a -o <`*result*`>.bib <`*file1*`>.bib <`*file2*`>.bib`

- export all the entries of two `.bib` files (without crossrefs):

  `bibexport -a -n -o <`*result*`>.bib <`*file1*`>.bib <`*file2*`>.bib`

  In fact, the only difference between this and the previous one is that `crossref` field will be filtered out at the end of the script.

- export all the entries of two `.bib` files, using an extra file containing cross-referenced entries (which should not be included):

```
bibexport -a -e <crossref>.bib -n -o <result>.bib \
              <file1>.bib <file2>.bib
```

1.4 Exporting extra fields

By default, `bibexport.sh` exports only "standard" fields (those defined and used in plain.bst), as well as a few others. It is very easy to modify it in order to export other fields: it suffices to modify export.bst as follows:

- in the `ENTRY` list, add the name of the field you would like to export. Notice that `ENTRY` takes three space-separated lists as arguments; you must add extra fields in the first argument (actually, the last two are empty).
- in the function `entry.export.extra`, add a line of the form

```
"myfield" myfield field.export
```

  where `myfield` is the name of the extra field you want to export.

Acknowledgements

I thank Éric Colin de Verdière and Richard Mathar for suggesting several improvements or corrections.

2 The code

2.1 The shell script

2.1.1 Initialization

`usage()` We first define how the script should be used:

```
1 ⟨*script⟩
2 function usage()
3 {
4 echo "bibexport: a tool to extract BibTeX entries out of .bib files.
5 usage: $0 [-h|v] [-n] [-b bst] [-a [-e file]...] [-o file] file...
6   -a, --all              export the entire .bib files
7   -b, --bst              specifies the .bst style file [default: export.bst]
8   -c, --crossref         include entries that are crossref'd  [default: yes]
9   -e file, --extra file  extra .bib files to be used (for crossrefs)
10  -n, --no-crossref      don't include crossref'd entries      [default: no]
11  -o file                write output to file       [default: bibexport.bib]
12  -p, --preamble         write a preamble at beginning of output
13  -t, --terse            operate silently
14  -h, --help             print this message and exit
```

```
  -v, --version          print version number and exit";
exit 0;
}
⟨/script⟩
```

VERSION
VDATE
CREF
FILE
EXT
EXTRA
EXTRABIB
SPACE
BST
TERSE
NOBANNER
ARGS

We define the default value of some variables:

- `$VERSION`: the version number,
- `$VDATE`: the release date,
- `$CREF`: the value of `-min-crossrefs`,
- `$FILE`: the input file(s),
- `$EXT`: the extension (`.aux` or `.bib`) of input files,
- `$EXTRA`: list of possible extra `.bib` files without extension,
- `$EXTRABIB`: list of possible extra `.bib` files with extension,
- `$SPACE`: file name separator (can be ␣, comma or empty),
- `$BST`: the `.bst` file to be used,
- `$TERSE`: tells BibTEX to run silently,
- `$NOBANNER`: don't print the initial comment,
- `$ARGS`: the list of aruments passed to `bibexport.sh`.

```
⟨*script⟩
VERSION="2.20";
VDATE="2009/10/11";
CROSSREF="1";
FILE="";
EXT=".aux";
EXTRA="";
EXTRABIB="";
SPACE=" ";
BST="export";
TERSE="";
NOBANNER="true";
ARGS=$@;
⟨/script⟩
```

2.1.2 Handling arguments

If no argument have been supplied, we call `usage()`.

```
⟨*script⟩
if [ $# -eq 0 ]; then
  usage;
fi
⟨/script⟩
```

Otherwise, we enter a `while`-loop for handling the whole list of arguments:

```
⟨*script⟩
while [ $# != 0 ]; do
        case $1 in
⟨/script⟩
```

- `-a` or `--all`: export all the bibliography. This means that we input `.bib` files.

```
⟨*script⟩
                -a|--all)
                  EXT=""; SPACE="";
                  shift ;;
⟨/script⟩
```

- `-b` or `--bst`: specifies the style file. It seems that BibTEX does not like the `./style.bst` syntax, and we have to handle that case separately.

```
⟨*script⟩
            -b|--bst)
                if [ "`dirname $2`" = "." ]; then
                    DOLLARTWO="`basename $2 .bst`";
                else
                    DOLLARTWO="`dirname $2`/`basename $2 .bst`";
                fi
                BST="${DOLLARTWO}";
                shift 2;;
⟨/script⟩
```

- `-e` or `--extra`: when we want to export all the entries of a `.bib` file, we can specify an extra `.bib` file that would contain entries that we don't want to export, but that are needed, *e.g.* for crossrefs.

```
⟨*script⟩
            -e|--extra)
                if [ "`dirname $2`" = "." ]; then
                    DOLLARTWO="`basename $2 .bib`";
                else
                    DOLLARTWO="`dirname $2`/`basename $2 .bib`";
                fi
                EXTRA="${EXTRA}${DOLLARTWO},";
                EXTRABIB="${EXTRABIB},${DOLLARTWO}.bib";
                shift 2;;
⟨/script⟩
```

- `-o` or `--output`: the name of the output file.

```
⟨*script⟩
            -o|--output-file)
                if [ "`dirname $2`" = "." ]; then
                    DOLLARTWO="`basename $2 .bib`";
```

```
                    else
                        DOLLARTWO="`dirname $2`/`basename $2 .bib`";
                    fi
                    OUTPUT="${DOLLARTWO}.bib";
                    shift 2 ;;
⟨/script⟩
```

- `-c` or `--crossref` (or others): this options means that we want crossrefs to be included. Note that for any entry, field inheritage will be performed.

```
⟨*script⟩
            -c|--crossref|--crossrefs|--with-crossref|--with-crossrefs)
                CREF="1" ;
                shift ;;
⟨/script⟩
```

- `-n` or `--no-crossref`: don't include crossref'ed entries.

```
⟨*script⟩
            -n|--no-crossref|--without-crossref)
                CREF="20000" ;
                shift ;;
⟨/script⟩
```

- `-v` or `--version` for version number:

```
⟨*script⟩
            -v|--version)
                echo "This is bibexport v${VERSION} (released ${VDATE})"; exit 0;;
⟨/script⟩
```

- `-p` or `--preamble` for inserting some informations at the beginning of the output file:

```
⟨*script⟩
            -p|--preamble)
                NOBANNER="false";
                shift ;;
⟨/script⟩
```

- `-t` or `--terse` for asking BibTeX to run silently:

```
⟨*script⟩
            -t|--terse)
                TERSE=" -terse ";
                shift ;;
⟨/script⟩
```

- other dash-options are erroneous (except `-h`, but...):

```
⟨*script⟩
            -*)
                usage;;
⟨/script⟩
```

- there should only remain file names: we add those names to the list of files.

```
⟨*script⟩
                *)
                    if [ "`dirname $1`" = "." ]; then
                        DOLLARONE="`basename $1 ${EXT}`";
                    else
                        DOLLARONE="`dirname $1`/`basename $1 ${EXT}`";
                    fi
                    FILE="${FILE}${SPACE}${DOLLARONE}${EXT}";
                    if [ -z "${SPACE}" ]; then
                        SPACE=",";
                    fi;
                    shift;;
⟨/script⟩
```

That's all folks:

```
⟨*script⟩
        esac
done
⟨/script⟩
```

2.1.3 The core of the script

We first set the name of the result and intermediary files:

```
⟨*script⟩
FINALFILE=${OUTPUT};
if [ ! "${FINALFILE}" ]; then
    FINALFILE="bibexport.bib";
fi
TMPFILE="bibexp.`date +%s`";
⟨/script⟩
```

We then create the `.aux` file for the main run of BibTEX. Note that this could call BibTEX, with the expkeys.bst file, in the case where we want to export all entries of a `.bib` file but not crossrefs. Note how, in that case, we trick BibTEXfor inputing extra files twice: we include then with their short name first (with no extension), and then with the full name. We *need* to do that, since `string` abbreviations must be defined first, while crossrefs must occur after having been referenced.

```
⟨*script⟩
if [ -z "${EXT}" ]; then
    if [ -z "${EXTRA}" ]; then
        cat > ${TMPFILE}.aux <<EOF
\citation{*}
\bibdata{${FILE}}
\bibstyle{${BST}}
EOF
    else
```

```
        cat > ${TMPFILE}.aux <<EOF
\citation{*}
\bibdata{${FILE}}
\bibstyle{expkeys}
EOF
        bibtex -min-crossrefs=${CREF} -terse ${TMPFILE};
        mv -f ${TMPFILE}.bbl ${TMPFILE}.aux;
        cat >> ${TMPFILE}.aux <<EOF
\bibdata{${EXTRA}${FILE}${EXTRABIB}}
\bibstyle{${BST}}
EOF
    fi
else
    cat ${FILE} | sed -e "s/bibstyle{.*}/bibstyle{${BST}}/" > ${TMPFILE}.aux;
fi
```
⟨/script⟩

This was the hard part. We now call BibTEX, clean and rename the output file, and remove intermediary files:

```
⟨*script⟩
bibtex -min-crossrefs=${CREF} ${TERSE} ${TMPFILE}
echo "" > ${FINALFILE}
if [ ! "${NOBANNER}" = "true" ]; then
    echo -ne "@comment{generated using bibexport:\n" >> ${FINALFILE};
    echo -ne "  creation date:\t`date +\"%c\"`\n" >> ${FINALFILE};
    echo -ne "  command:\t\t$0 ${ARGS}\n" >> ${FINALFILE};
    echo -ne "  bibexport-version:\tv${VERSION} (${VDATE})\n" >> ${FINALFILE};
    echo -ne "  bibexport-maintainer:\tmarkey(at)lsv.ens-cachan.fr\n" >> ${FINALFILE};
    sed -ie "s/}/)/g" ${FINALFILE};
    echo -ne "}\n\n\n" >> ${FINALFILE};
fi
if [ ! "x${CREF}" = "x1" ]; then
    sed -r -e \
        "/^ *[cC][rR][oO][sS][sS][rR][eE][fF] *= *[^,]+,?$/d" \
        ${TMPFILE}.bbl >> ${FINALFILE};
else
    cat ${TMPFILE}.bbl >> ${FINALFILE};
fi
rm -f ${TMPFILE}.bbl ${TMPFILE}.aux ${TMPFILE}.blg
```
⟨/script⟩

2.2 The **expkeys.bst** file

The only role of that file is to export the list of entries to be exported. It is used when we export all the entries of `.bib` files, except those of *extra* `.bib` files. Thus:

```
⟨*expkeys⟩
ENTRY{}{}{}
READ
FUNCTION{export.key}
```

```
{
  "\citation{" cite$ "}" * * write$ newline$
}
ITERATE{export.key}
```
⟨/expkeys⟩

2.3 The export.bst file

2.3.1 Some configuration values

left.width
right.width
url.right.width
left.short.width
right.short.width
left.delim
right.delim

We define here the indentation values, and the field delimiters. *short* width are used for `@preamble`.

⟨∗export⟩
```
FUNCTION{left.width}{#18}
FUNCTION{right.width}{#55}
FUNCTION{url.right.width}{#61}
FUNCTION{left.short.width}{#10} %% for @preamble
FUNCTION{right.long.width}{#63}
FUNCTION{left.delim}{"{"}
FUNCTION{right.delim}{"}"}
%FUNCTION{left.delim}{quote$}
%FUNCTION{right.delim}{quote$}
```
⟨/export⟩

2.3.2 Entries

We use standard entries here. Of course, more entries could be added for special `.bib` files. Those extra entries will also have to be added in the main exporting function.

ENTRY

⟨∗export⟩
```
ENTRY{
% Standard fields:
    address
    author
    booktitle
    chapter
    edition
    editor
    howpublished
    institution
    journal
    key
    month
    note
    number
    organization
    pages
```

```
    publisher
    school
    series
    title
    type
    volume
    year
% Special (but still somewhat standard) fields (natbib, germbib, ...):
    abstract
    doi
    eid
    isbn
    issn
    language
    url
}{}{}
```
⟨/export⟩

2.3.3 Basic functions

No comment.

or
and
not

```
⟨*export⟩
FUNCTION{not}
{
    {#0}
    {#1}
  if$
}
FUNCTION{and}
{
    'skip$
    {pop$ #0}
  if$
}
FUNCTION{or}
{
    {pop$ #1}
    'skip$
  if$
}
⟨/export⟩
```

2.3.4 Splitting strings

We design functions for splitting strings, so that the final `.bib` file will be cleanly indented.

space.complete
split.string
split.url
split name

```
⟨*export⟩
INTEGERS{left.length right.length}
STRINGS{ s t }
FUNCTION{space.complete}
{
  'left.length :=
  duplicate$ text.length$ left.length swap$ -
  {duplicate$ #0 >}
    {
      swap$ " " * swap$ #1 -
    }
  while$
  pop$
}
FUNCTION{split.string}
{
  'right.length :=
  duplicate$ right.length #1 + #1 substring$ "" =
    {""}
    {
      's :=
      right.length
      {duplicate$ duplicate$ s swap$ #1 substring$ " " = not and}
        {#1 -}
      while$
      duplicate$ #2 <
        {
          pop$ "    " s * ""
        }
        {
          duplicate$ s swap$ #1 swap$ substring$
          swap$
          s swap$ global.max$ substring$
        }
      if$
    }
  if$
}
FUNCTION{split.url}
{
  'right.length :=
  duplicate$ right.length #1 + #1 substring$ "" =
    {""}
    {
      's :=
      right.length
      {duplicate$ duplicate$ s swap$ #1 substring$ "/" = not and}
        {#1 -}
      while$
      duplicate$ #2 <
```

```
        {
          pop$ "    " s * ""
        }
        {
          duplicate$ s swap$ #1 swap$ substring$
          swap$ #1 +
          s swap$ global.max$ substring$
        }
      if$
    }
  if$
}
FUNCTION{split.name}
{
  'right.length :=
  duplicate$ right.length #1 + #1 substring$ "" =
    {""}
    {
      's :=
      right.length
      {duplicate$ duplicate$ s swap$ #5 substring$ " and " = not and}
        {#1 -}
      while$
      duplicate$ #2 <
        {
          pop$ "  " s * ""
        }
        {
          #4 + duplicate$ s swap$ #1 swap$ substring$
          swap$
          s swap$ global.max$ substring$
        }
      if$
    }
  if$
}
⟨/export⟩
```

2.3.5 Exporting fields

Here, we have four exporting functions, since we also have to deal with abbreviations:

field.export
abbrv.export
name.export
url.export

```
⟨*export⟩
FUNCTION{field.export}
{
  duplicate$ missing$
    'skip$
    {
```

```
      left.delim swap$ * right.delim *
      swap$
      "  " swap$ * " = " * left.width space.complete
      swap$ "," *
      {duplicate$ "" = not}
        {
          right.width split.string 't :=
          *
          write$ newline$
          "" left.width space.complete t
        }
      while$
    }
  if$
  pop$ pop$
}
FUNCTION{abbrv.export}
{
  duplicate$ missing$
    'skip$
    {
      swap$
      "  " swap$ * " = " * left.width space.complete
      swap$ "," *
      {duplicate$ "" = not}
        {
          right.width split.string 't :=
          *
          write$ newline$
          "" left.width space.complete t
        }
      while$
    }
  if$
  pop$ pop$
}
FUNCTION{name.export}
{
  duplicate$ missing$
    'skip$
    {
      left.delim swap$ * right.delim *
      swap$
      "  " swap$ * " = " * left.width space.complete
      swap$ "," *
      {duplicate$ "" = not}
        {
          right.width split.name 't :=
          *
          write$ newline$
```

```
394           "" left.width space.complete t
395         }
396       while$
397     }
398   if$
399   pop$ pop$
400 }
401 FUNCTION{url.export}
402 {
403   duplicate$ missing$
404     'skip$
405     {
406       left.delim swap$ * right.delim *
407       swap$
408       "  " swap$ * " = " * left.width space.complete
409       swap$ "," *
410       {duplicate$ "" = not}
411         {
412           url.right.width split.url 't :=
413           *
414           write$ newline$
415           "" left.width space.complete t
416         }
417       while$
418     }
419   if$
420   pop$ pop$
421 }
422 ⟨/export⟩
```

2.3.6 Handling abbreviations

Abbreviations are difficult to deal with if we wish to still use them, since BibTEXwill expand them before we can do anything. All we can do is to define them in a special way, in order to be able to get back to the abbreviations later on. This is precisely what we do:

jan-dec
acmcs-tcs
remove.exports.from.months
remove.export.from.journal

```
423 ⟨*export⟩
424 MACRO{jan}{"export-jan"}
425 MACRO{feb}{"export-feb"}
426 MACRO{mar}{"export-mar"}
427 MACRO{apr}{"export-apr"}
428 MACRO{may}{"export-may"}
429 MACRO{jun}{"export-jun"}
430 MACRO{jul}{"export-jul"}
431 MACRO{aug}{"export-aug"}
432 MACRO{sep}{"export-sep"}
433 MACRO{oct}{"export-oct"}
434 MACRO{nov}{"export-nov"}
```

```
MACRO{dec}{"export-dec"}
MACRO{acmcs}{"export-acmcs"}
MACRO{acta}{"export-acta"}
MACRO{cacm}{"export-cacm"}
MACRO{ibmjrd}{"export-ibmjrd"}
MACRO{ibmsj}{"export-ibmsj"}
MACRO{ieeese}{"export-ieeese"}
MACRO{ieeetc}{"export-ieeetc"}
MACRO{ieeetcad}{"export-ieeetcad"}
MACRO{ipl}{"export-ipl"}
MACRO{jacm}{"export-jacm"}
MACRO{jcss}{"export-jcss"}
MACRO{scp}{"export-scp"}
MACRO{sicomp}{"export-sicomp"}
MACRO{tocs}{"export-tocs"}
MACRO{tods}{"export-tods"}
MACRO{tog}{"export-tog"}
MACRO{toms}{"export-toms"}
MACRO{toois}{"export-poois"}
MACRO{toplas}{"export-toplas"}
MACRO{tcs}{"export-tcs"}
INTEGERS{ intxt }
FUNCTION{remove.exports.from.months}
{
  #0 'intxt :=
  duplicate$ missing$
    'skip$
    {'t :=
    ""
    {t #1 #1 substring$ "" = not}
      {
      t #1 #7 substring$ "export-" =
        {intxt
           {right.delim * #0 'intxt :=}
           'skip$
         if$
         duplicate$ "" =
           'skip$
           {" # " *}
         if$
         t #8 #3 substring$ *
         t #11 global.max$ substring$ 't :=}
        {intxt
           'skip$
           {duplicate$ "" =
              {}
              {" # " *}
            if$
            left.delim * #1 'intxt :=}
         if$
```

```
485         t #1 #1 substring$ *
486         t #2 global.max$ substring$ 't :=}
487       if$
488       }
489     while$
490     intxt
491       {right.delim *}
492       'skip$
493     if$
494     }
495   if$
496 }
497 FUNCTION{remove.export.from.journals}
498 {
499   duplicate$ missing$
500     'skip$
501     {
502       duplicate$ #1 #7 substring$ "export-" =
503         {#8 global.max$ substring$}
504         {left.delim swap$
505          right.delim * *}
506       if$
507     }
508   if$
509 }
510 ⟨/export⟩
```

2.3.7 Now, we export...

We gather everything. This is were special fields must be added for being exported:

entry.export.standard
entry.export.extra
entry.export
export

```
511 ⟨*export⟩
512 FUNCTION{entry.export.standard}
513 {
514   "address" address field.export
515   "author" author name.export
516   "booktitle" booktitle field.export
517   "chapter" chapter field.export
518   "crossref" crossref field.export
519   "edition" edition field.export
520   "editor" editor name.export
521   "howpublished" howpublished field.export
522   "institution" institution field.export
523   "journal" journal remove.export.from.journals abbrv.export
524   "key" key field.export
525   "month" month remove.exports.from.months abbrv.export
526   "note" note field.export
527   "number" number field.export
528   "organization" organization field.export
```

```
  "pages" pages field.export
  "publisher" publisher field.export
  "school" school field.export
  "series" series field.export
  "type" type field.export
  "title" title field.export
  "volume" volume field.export
  "year" year field.export
}
FUNCTION{entry.export.extra}
{
  "abstract" abstract field.export
  "doi" doi field.export
  "eid" eid field.export
  "isbn" isbn field.export
  "issn" issn field.export
  "language" language field.export
  "url" url url.export
}
FUNCTION{entry.export}
{
  entry.export.standard
  entry.export.extra
}
FUNCTION{export}
{
  "@" type$ * "{" * cite$ * "," * write$ newline$
  entry.export
  "}" write$ newline$ newline$
}
```
⟨/export⟩

2.3.8 Miscellanea

We also have to handle preamble, and to define functions for each entry type (we won't use them but otherwise, BibTEXwould complain).

preamble
header
entries.headers
article-unpublished

⟨*export⟩

```
FUNCTION{preamble}
{
preamble$ duplicate$ "" =
  'pop$
  {
    ",--------------------." write$ newline$
    "|     PREAMBLE      |" write$ newline$
    "`--------------------'" write$ newline$ newline$
    "@preamble{ "  swap$
    quote$ swap$ * quote$ *
    {duplicate$ "" = not}
```

```
      {
        right.long.width split.string 't :=
        *
        write$ newline$
        "" left.short.width space.complete t
      }
    while$
    "}" write$ newline$ newline$
    pop$ pop$
  }
if$
}
FUNCTION{header}
{
%"** This file has been automatically generated by bibexport **"
%write$ newline$
%"** See   http://www.lsv.ens-cachan.fr/~markey/bibla.php    **"
%write$ newline$
%"** for more informations about bibexport.                  **"
%write$ newline$
newline$
}
FUNCTION{entries.header}
{
preamble$ "" =
  'skip$
  {
    ",--------------------." write$ newline$
    "|  BIBTEX ENTRIES   |" write$ newline$
    "`--------------------'" write$ newline$ newline$
  }
if$
}
FUNCTION{article}{export}
FUNCTION{book}{export}
FUNCTION{booklet}{export}
FUNCTION{conference}{export}
FUNCTION{habthesis}{export}
FUNCTION{inbook}{export}
FUNCTION{incollection}{export}
FUNCTION{inproceedings}{export}
FUNCTION{journals}{export}
FUNCTION{manual}{export}
FUNCTION{mastersthesis}{export}
FUNCTION{misc}{export}
FUNCTION{phdthesis}{export}
FUNCTION{proceedings}{export}
FUNCTION{techreport}{export}
FUNCTION{unpublished}{export}
```
⟨/export⟩

2.3.9 Main program

We now can execute and iterate those functions:

```
⟨*export⟩
READ
EXECUTE{header}
EXECUTE{preamble}
EXECUTE{entries.header}
ITERATE{export}
⟨/export⟩
```