

LB_BTeX

Lua による Bib_BTeX の実装です.

1 使い方

TeX Live または W32TeX をインストールしておいてください. 拡張子が lua であるファイルを全て kpathsea が探せる場所においてください. 例えば TeX Live の標準設定では \$TEXMF/scripts 以下に置くことができます. また,

- UNIX: lbibtex.lua へのリンクを適当な bin ディレクトリに作る.
- Windows (TeX Live): bin/win32/runscript.exe を bin/win32/lbibtex.exe としてコピーする.
- W32TeX: bin/win32/runscr.exe を bin/win32/lbibtex.exe としてコピーする.

とします. または, 代わりに lbibtex (UNIX) または lbibtex.bat (Windows) を PATH の通っている場所におくことでも実行が可能になります.

```
$ bibtex sample
```

としていた代わりに

```
$ lbibtex sample
```

とします. つまり, sample.tex を処理するには

```
$ latex sample.tex
```

```
$ lbibtex sample
```

```
$ latex sample.tex
```

```
$ latex sample.tex
```

とします. 文字コードは (現在のところ) UTF-8 に限定されています.

2 データベースについて

通常の.bibを読むことができます. 典型的には次のようになっています.

```
@article{reference,  
  author = "Last, First",  
  title = {Some title},  
}
```

これは reference という名前のついた article に関する情報です. 著者名とタイトルが定義されています. 本マニュアルでは,

- 各々のデータを「エントリー」
- article を「エントリータイプ」
- reference を「エントリーキー」
- 著者名などの情報を「フィールド」
- author = "Last, First"における author を「キー」"Last, First"を「値」

と呼ぶことにします。エントリータイプ、エントリーキー、またフィールドのキーは大文字小文字を無視して処理されます。

また次のようなデータ

```
@string{str = "some string"}
@article{reference
    title = "Title and " # str
}
```

に対しては、文字列の連結と置換が行われます。たとえばこの例では reference 内の title に対する値が “Title and some string” と置き換えられます。このような置き換えのルール（今の場合は str を “some string” に置き換える）をマクロと呼ぶことにします。マクロはこのようなデータベース内の string エントリーを使っても定義できますし、スタイルファイル内で定義することもできます。

正確には次のような EBNF で定義されたファイルを読みます。

```
Database = (Ignored '@' Entry)*
Ignored = [^@]*
Entry = Preamble|Comment|String|Data
Comment = "comment" ('{' Name '}'|'(' Name ')')
Preamble = "preamble" ('{' Name '}'|'(' Name ')')
String = "string" ('{' Fields '}'|'(' Fields ')')
Data = Entry_Type ('{' Entry_Key ',' Fields '}'|'(' Entry_Key ',' Fields ')')
Entry_Type = [^{(]*
Entry_Key = Name
Fields = Field? (',' Field)* ','?
Field = Key '=' Value
Key = Name
Value = Name
Name = ([^{}"]+ | '"'[^"]*" | '{' Name '}')*
```

- Ignored は無視されます。通常空白と改行のみを含みます。（ただし、この部分をコメントとして用いることも可能です。）
- Comment はコメントです。無視されます。
- Preamble はそのまま bbl に書き出されることが想定されています。
- String はマクロを定義します。これは後述の「文字列連結機能」にて用いられます。
- Data が文献情報を表します。

また文字列の連結および置換は次のように振る舞います.

```
Value = EachString ('#' EachString)*
EachString = ([^#{ }"]* | '"' [^"]* '"' | '{' EachString '}')*
```

各々の EachString には以下の処理が施されます.

- 前後の空白は全て無視されます.
- EachString と同じ文字列がマクロとして定義されていた場合, 対応する文字列に変更されます.
- 最後に, 最初及び最後の "{}" は削除されます.

3 スタイルファイルの書き方

Lua 言語によりスタイルを記述することができます. kpathsea から見える場所に `lbt-<style>_bst.lua` として保存してください. (<style> はスタイル名.) 標準の plain, alpha, abbrev, unsrt に対応するファイルは既に用意されています.

スタイルファイルの中身は, Lua スクリプトファイルです. グローバル変数 BibTeX を通じ, 各種設定などを行います. 多くの場合, 次のような流れになるでしょう.

- (1) BibTeX.blockseparator に, テンプレート設定で使うセパレータを設定する.
- (2) BibTeX.templates と BibTeX.formatters に実際に thebibliography 環境として出力する内容のテンプレートを設定する.
- (3) BibTeX.crossref にクロスリファレンスの設定を行う.
- (4) BibTeX.sorting にソートの設定をする.
- (5) BibTeX.label にラベル出力の設定をする.
- (6) BibTeX:outputthebibliography() で出力を行う.

順番に見ていきます.

3.1 テンプレート設定

BibTeX.templates, BibTeX.formatters および BibTeX.blockseparator を通じて設定を行います. たとえば, エントリータイプ article に対しては, 著者, タイトル, ジャーナル, 年をカンマ区切りで出し, 最後にピリオドをつける場合は次のようにします.

```
local Functions = require "lbt-funcs"
```

```
BibTeX.blockseparator = {"", "\u0020", "."}
BibTeX.templates["article"] = "[%<author>:<\emph{|\$<title>|}>:<journal>:<year>]"
function BibTeX.formatters:author(c)
  if c.fields["author"] == nil then return nil end
  local a = Functions.split_names(c.fields["author"])
  if #a <= 2 then
    return Functions.make_name_list(a, {"ff~}{vv~}{ll}{,jjj}", {"", "\u0020", "\u0026\u0020"}, "\u0026et~al.")
  else
    return Functions.make_name_list(a, {"ff~}{vv~}{ll}{,jjj}", {"", "\u0020", "\u0026\u0020"}, "\u0026et~al.")
  end
end
```

```
end
end
```

BibTeX.templates に実際に出力される内容を設定します。次のような書式で指定します。

- “[A:B:C...:X]” は「ブロック」を表します。各ブロックには「セパレータ」“<sep>”と「終端文字列」“<last>”が設定されており、“A<sep>B<sep>C...<sep>X<last>”というように出力されます。ただし、たとえば“B”が空文字列の場合は、“A<sep>C...<sep>X<last>”というように出力されます。なお、このセパレータや終端文字列では、“.”が連続しないように処理がされます。ブロックはネストが可能です。
- “\$<A>”はフィールド A の出力を行います。A がフィールドにない場合は空文字列になります。また“\$<A|B|...|X>”と続けることもできて、この場合は A,B,...,X の中で最初に定義されているものが出力されます。
- “<A|B|C>”は、B が空文字列ならば空文字列に、そうでないならば“ABC”という文字列になります。ネストが可能です。
- 特殊文字は“%”でエスケープできます。

ブロックのセパレータと終端文字列は BibTeX.blockseparator で設定します。中身は配列で、

```
BibTeX.blockseparator = {
  {<ネストレベル 1 のセパレータ>,<ネストレベル 1 の終端文字列>},
  {<ネストレベル 2 のセパレータ>,<ネストレベル 2 の終端文字列>},
  ...
}
```

という形です。

“\$<A|B|...|X>”で出力される各種フィールドの出力は BibTeX.formatters により整形されます。その実体は関数で、キー name のフィールドの整形を行う関数は

```
function BibTeX.formatters:name(c)
-- 本体
end
```

という形で定義します。戻り値は文字列です。引数 c には

- c.key にはエントリーキー
- c.type にはエントリータイプ
- c.fields[name] にはキーが name のフィールドの中身

が入っています。より詳しくは節 5 を参照してください。上の author の例ではモジュール lbt-funcs の提供する関数を使っています。節 4 を参照してください。

BibTeX.formatters の名前は実際のフィールド名である必要はありません。たとえば

```
BibTeX.templates["article"] = "$<author_editor>:$<title>"
function BibTeX.formatters:author_editor(c)
  if c.fields["author"] == nil then return c.fields["editor"]
  else return c.fields["author"]
end
```

とすると、“\$<author_editor>”は「author が定義されていれば author フィールドに、そうでなければ editor フィールド」という扱いになります。（つまり“\$<author|editor>”と同等。）

少し発展的な内容です。

- ブロックの定義において、“[A:@S<sep>B:C]”とすると、B の前のセパレータを“sep”に変更できます。
- “\$<A|(B)|C|...|X>”とすると、B はフィールド名ではなく、テンプレートして解釈されます。たとえば、“\$<author|(<edited by |\$<editor>|.)>”とすると、
 - author が定義されていれば author フィールドそのまま。
 - author が定義されていなく、editor が定義されていれば“edited by <editor フィールド>.”
 - author も editor も定義されていなければ空文字列が出力されます。

- formatters にも templates のような書式が使えます。たとえば上の BibTeX.formatters:author_editor の例は

```
BibTeX.formatters.author_editor = "$<author|editor>"
```

と書くこともできます。なお、ここでの“\$<A>”によるフィールド名の参照は、必ずフィールドの内容そのままとして解釈され、formatters による整形は行われません。

- formatters の関数の戻り値は原則文字列ですが、文字列の配列を返すこともできます。これはブロックとして扱われます。たとえば

```
BibTeX.templates["article"] = "$<author>:$<title_journal_year>"
function BibTeX.formatters:title_journal_year(c)
  return {c.fields["title"],c.fields["journal"],c.fields["year"]}
end
```

と

```
BibTeX.templates["article"] = "$<author>:$<title>:$<journal>:$<year>"
```

は等価です。

3.2 クロスリファレンス

クロスリファレンスの設定は BibTeX.corsref に対して行います。例としては次のようになります。

```
BibTeX.crossref.templates["article"] = "$<author>:$<title>:\\cite{$<crossref>}"
```

これにより、corsref フィールドが定義されている article に対しては、その出力が上で指定されたものになります。なお、formatters や blockseparator は BibTeX.formatters や BibTeX.blockseparator がそのまま使われます。また、BibTeX.crossref.templates["article"] が定義されていない場合は BibTeX.formatters["article"] が使われます。

3.2.1 クロスリファレンスの遺伝

クロスリファレンスが行われると、親エントリーから子エントリーへとフィールドのコピーが行われます。デフォルトでは、そのままのコピーが行われますが、この挙動は制御することができます。たとえば

```
BibTeX.crossref.inherit["article"]["book"] = {
  {"title","booktitle"},
  {"author","editor"},"editor"},
```

```
    {"A", "B"}, {"C", "D"}
}
```

とすると、親：article、子：book というクロスリファレンスに対して

- title は booktitle にコピー
- author と editor は editor にコピー
- A,B は C,D の両方にコピー

が行われます。各々の項目に空文字列 “” を指定すると、それは「全部」を表します。たとえば

```
BibTeX.crossref.inherit[""][""] = {
    {"title", "booktitle"}
}
```

は全てのエントリータイプに対して、title を booktitle へとコピーします。個別の指定は、“” による全ての指定より優先されます。たとえば

```
BibTeX.crossref.inherit[""][""] = {
    {"title", "booktitle"}
}
BibTeX.crossref.inherit["article"][""] = {
    {"title", "subtitle"}
}
```

という指定は、article からの場合に限り title を subtitle に、それ以外は title を booktitle にコピーします。

3.2.2 その他の設定

子エントリーに既にフィールドが存在している場合に上書きするかどうかは、`BibTeX.crossref.override` で制御します。簡単な方法は

```
BibTeX.crossref.override = true
```

とすることです。これで全てのフィールドが上書きされます。（なお、デフォルトは false です。）inherit と同様個別の定義を行うこともできます。たとえば

```
BibTeX.crossref.override["article"]["book"] = {
    {"author", "editor"}, {"bookeditor"}, true
}
```

は親：article の author か editor フィールドが子：book の bookeditor フィールドにコピーされる場合に上書きを許すことを意味します。inherit と同様 “” は全ての項目を表します。

その他以下の項目が設定できます。

- `BibTeX.crossref.mincrossrefs`：ここに設定されているだけのクロスリファレンスがあれば、エントリーが現在の参考文献一覧に追加されます。デフォルト 2。
- `BibTeX.crossref.reference_key_name`：クロスリファレンスを表すフィールドのキー名です。デフォルト “crossref”。

3.3 ソート

ソートに関する設定は、`BibTeX.sorting` で行います。

```
BibTeX.sorting.targets = {"name","title","year"}
```

とすると、「名前」「タイトル」「年」の順番で比較されます。タイトルと年については、ほぼフィールド名そのまま^{*1}で比較されます。名前については、デフォルトでは

- book, inbook: author/editor/key
- proceedings: editor/organization/key
- manual: author/organization/key
- その他: author/key

のうち定義されている最初のものになります。BibTeX.sorting.targets には上の “name” とフィールド名その他, “entry_key” (エントリーキー), “label” (ラベル) が指定できます。

実際に比較する値は、BibTeX.sorting.formatters で設定可能です。

```
function BibTeX.sorting.formatters.name(c)
....
end
```

とすると、上の name に対応する定義を上書きすることができます。

比較するための関数は、

- 一致しているか否かを返す BibTeX.sorting.equal
- < であるかを返す BibTeX.sorting.less than

で設定できます。デフォルトでは

```
local function purify(s)
    return s:gsub("\\[a-zA-Z]*", ""):gsub("[_-/:-@%[-{~-]", "")
end
function BibTeX.sorting.less than(a,b)
    return unicode.utf8.lower(purify(a)) < unicode.utf8.lower(purify(b))
end
function BibTeX.sorting.equal(a,b)
    return unicode.utf8.lower(purify(a)) == unicode.utf8.lower(purify(b))
end
```

と定義されています。

3.4 ラベル

thebibliography 環境における

```
\bibitem[label]{key} ....
```

の label の部分をラベルと呼ぶことにします。デフォルトでは、著者などから自動的に生成されます。ただし、shorthand フィールドがある場合には、その値が使われます。ラベルの生成を押さえる（標準スタイルの「plain」に対応）には

```
BibTeX.label.make = nil
```

^{*1} タイトルの頭文字の A, An, The は取り除かれる。

とします.

より細かく設定する場合は, `BibTeX.label.templates` と `BibTeX.label.formatters` を設定します. 設定の方法はテンプレート (項 3.1) と同様です. なお, 同じラベル名が生成された場合, デフォルトでは末尾に `a,b,c,...` が追加されます.

3.5 出力

最後に

```
BibTeX:outputthebibliography()
```

とすることで, `.bbl` ファイルが出力されます.

4 関数

有用そうな関数群やオリジナルの `BibTeX` に存在していた関数が, モジュール `lbt-funcs` で定義されています.

```
local Functions = require "lbt-funcs"
x = Functions.text_prefix(str,num)
```

のように使ってください.

4.1 `stable_sort(list,comp)`

配列 `list` に対して, 安定なソートを行います. `comp` は比較関数です. 省略された場合は標準演算子 `<` が使われます.

4.2 `text_prefix(str,num)`

`str` の先頭 `num` バイトを返します. ただし, 文字を途中で切ることはなく, またコントロールシーケンス等や引数はバイト数に加算されません. たとえば,

```
text_prefix("a あい",2)
text_prefix("あいう",5)
```

はそれぞれ `"a あ"`, `"あい"` を返します.*2

4.3 `text_length(str)`

`str` のバイト数を返しますが, コントロールシーケンス等や引数は加算されません.

4.4 `string_split(str,func)`

検索関数 `func` により `str` を分割して返します. 戻り値は二つの配列で, 一つ目の配列には分割された文字列, 二つ目の配列には分割文字列が入ります. たとえば

*2 内部コードは UTF-8 なので, `"あ"` や `"い"` は 3byte です. この扱いはどうするか考え中…….


```
string_split("aXbYc",function(s) return s:find("[XY]") end)
```

は

```
{"a","b","c"},{"X","Y"}
```

を返します.

4.5 change_case(str,format)

大文字小文字の変換を行います. ただし, 中括弧の中は処理されません. format は “t”, “u”, “l” のどれかで,

- “u”, “l” はそれぞれ大文字, 小文字への変換を表す.
- “t” は小文字への変換を行うが, 一文字目及び “: *” で表される文字の次の文字は変換されない.

4.6 split_names(names[,seps])

複数名の名前からなる文字列から, 各人の名前の入った配列を得ます. 人と人との区切りを配列 seps で与えることができます. (配列中のいずれかにマッチした部分で区切られる.) seps のデフォルトは {"[aA][nN][dD]"} です.

4.7 get_name_parts(names)

名前から first name, last name, von part, jr part の四つの部分を抽出します. 戻り値は {first = <first part>, last = <last part>, von = <von part>, jr = <jr part>}

で, 各々の部分は

```
{parts = <array of name>, seps = <separator of names>}
```

です. 例えば von-von Last Last, First, jr に対しては, 次のように返ります.

```
{
  first = {parts = {"First"}, seps = {}},
  last = {parts = {"Last","Last"}, seps = {" "}},
  von = {parts = {"von", "von"}, seps = {"-"}},
  jr = {pars = {"jr"}, seps = {}}
}
```

この関数は, 次のルールに基づき名前を分解します.

- (1) “[,~\t%-]+” に該当するパターン^{*3}で区切り, 配列を生成する.
- (2) 1 で区切られた際に用いられた区切り文字のうち, 最初の一文字がカンマ「,」のものの数を数える. この数に基づき, 次の三つのパターンのどれかと見なす.
 - (a) カンマがない: First von Last のパターン. 頭から見て von と見なされるパターンの前までが First, 後ろから見て von と見なされるパターンの後ろまでが Last. von がない場合は 1 で区切ら

^{*3} Lua の意味でのパターン

れたうちの最後の一つのみが Last. (ただし, 区切り文字が“-”のものはまとめて考える. 例えば “First Last Last” の Last は “Last” であるが, “First Last-Last” ならば “Last-Last” である.)

(b) カンマが一つ: von Last, First のパターン. von Last から Last を抜き出す処理は (a) と同じ.

(c) カンマが二つ: von Last, Jr, First のパターン. von Last から Last を抜き出す処理は (a) と同じ.

(3) 2 における「von と見なされるパターン」とは, (基本的には)*⁴最初に現れたアルファベットが小文字であるもののことである.

4.8 forat_name_by_parts(nameparts,format)

format にて指定された書式に基づき, 名前の整形を行います. nameparts は get_name_parts で得られる戻り値と同じかたちで与えます. format で与える書式は次の形です.

```
<str1>{<before1><name1><after1>}<str2>{<before2><name2><after2>}...
```

- <str1>はそのまま出力される.
- <name1>は “l”, “ll”, “f”, “ff”, “v”, “vv”, “j”, “jj” の何れか. Last name, First name, von part, jr part に対応し, 二つ続いているものは名前全体を, そうでないものは短縮形を表す.
- <before1>はそのまま出力される. ただし<name1>に対応する部分がない場合, 出力されない.
- <after1>は{<sep1><after1_>か<after1_> (中括弧なし) の何れかである. <sep1>は<name1>の各部分をつなぐ文字として使われ, <after1_>は次の部分とのつなぐ文字として使われる. <sep1>が省略された場合や, “~” であった場合は, 空白 (“ ” か “~”) が状況に応じて使われる. もし常に “~” を出力したい場合は, “~~” を指定する.
- <str2>等も同様.

4.9 format_name(name,format)

BIBTEX の format.names\$ と似た関数です. 中身は

```
return forat_name_by_parts(get_name_parts(name),format)
```

です.

4.10 make_name_list(namearray, format, separray[, etalstr])

複数人の名前の配列から文字列を作ります. separray の長さを k, namearray の長さを n とすると,

```
namearray[1]separray[1]namearray[2]separray[2] ....
namearray[n - k + 1]separray[2] ...
namearray[n - 1]separray[k]namearray[n]
```

という文字列を生成します. (実際には改行無し.) ただし, namearray の各項は format に従い整形され (書

*4 実際には中括弧内や, コントロールシーケンスで定義されたアクセントなども考慮に入れる.

式は `format_name_by_parts` と同様), またもし `namearray[n]` が “others” の場合は, `namearray[n]` は `etalstr` に置き換えられます. デフォルトでは `etalstr` は空文字列です.

4.11 `remove_TeX_cs(s)`

`s` から \TeX のコントロールシーケンスを取り除いた文字列を得ます.

5 文献データ

文献データは以下のようなテーブルに格納されています. 変数名を `Citation` とします.

`Citation.type`

エントリータイプ

`Citation.key`

エントリーキー

`Citation.fields`

フィールドが格納されているテーブル. マクロなどが施された結果が帰る.

`function Citation:clone()`

自分の複製を作ります.

`function Citation:set_field(key,cite,key1)`

文献データ `cite` のキー “key1” のフィールドを “key” に設定します.

`function Citation:get_raw_field(key)`

キー “key” のフィールドの生の値 (マクロなど適用前) を返します.

6 変数 BibTeX

変数 `BibTeX` には, 現在の \LaTeX の状態が格納されています.

6.1 各種状態

`BibTeX.style`

スタイル名.

`BibTeX.cites`

引用されている文献一覧からなる配列. 各々の中身は節 5 の通り.

`BibTeX.db`

読み込まれたデータベースを表すテーブル. エントリーキー “key” には

`BibTeX.db["key"]`

でアクセスできる. 各々の中身は節 5 の通り.

`BibTeX.aux`

aux ファイル名.

`BibTeX.aux_contents`

aux ファイル名の中身. aux の各行の

```
\somecs{arg1}[arg2](arg3)
```

という行から

```
{somecs = {  
  {arg = "arg1", open = "{", close = "}"},  
  {arg = "arg2", open = "[", close = "]"},  
  {arg = "arg3", open = "(", close = ")}  
}}
```

というテーブルが生成されて、ここに格納されている. 括弧は上記の “{ }”, “[]”, “()” が認識され, 対応がとれているものとして扱われる.

6.2 関数

`BibTeX:output(str)`

bb1 への出力を行う.

`BibTeX:outputline(str)`

bb1 への一行出力を行う.

`BibTeX:outputthebibliography()`

項 6.3 に従い bb1 への thebibliography 環境の出力を行う.

`BibTeX:warning(str)`

文字列 `str` を警告として出力する. 出力は標準出力および `blg` に対して行われる.

`BibTeX:error(str,exit_code)`

文字列 `str` をエラーとして出力し, 終了コード `exit_code` でプログラムを終了する. 出力は標準エラー出力および `blg` に対して行われる.

`BibTeX:log(str)`

`blg` に `str` を出力する.

`BibTeX:message(str)`

標準出力に `str` を出力する.

6.3 出力設定

`BibTeX.templates`

出力される `\bibitem` のフォーマットを指定する. 書式は項 3.1 に基づく.

`BibTeX.formatters`

`BibTeX.templates`, `BibTeX.crossref.templates` で使われる整形用の関数. 書式は項 3.1 に基づく.

`BibTeX.blockseparator`

`BibTeX.templates`, `BibTeX.crossref.templates` におけるブロックの区切り文字.

`BibTeX.crossref`

クロスリファレンスの遺伝を設定する.

`BibTeX.crossref.templates`

クロスリファレンスが定義されている場合に使われるフォーマット. 値が定義されていない場合, `BibTeX.templates` が使われる.

`BibTeX.sorting.targets`

ソートの際に使われるフィールドキーを並べた配列. (正確には, テーブル `BibTeX.sorting.formatters` のキーを指定する. つまり, “key” を指定すると, 関数 `BibTeX.sorting.formatters:key` が呼び出された結果が使われる.)

`BibTeX.sorting.lessthan`, `BibTeX.sorting.equal`

ソートのための比較関数.

`BibTeX.sorting.formatters`

ソート時のフィールドの整形関数からなるテーブル. 比較の際に `self` と文献情報 (節 5) が渡されて実行される.

`BibTeX.sorting.label:make`

`self` と文献情報 (節 5) を受け取り, ラベル名を返す関数を設定する. ラベルを作らない場合は `nil` を設定する.

`self.label:add_suffix`

同一のラベル名があった場合に, ラベル名を変更する処理をおこなう関数を設定する. デフォルトでは末尾に `a,b,c,...` を付加する. 文献情報からなる配列 (ソート済み) を受け, やはり配列を返す.

`self.label:modify_citations`

出力直前に実行される関数. 最後の段階で文献情報を調整することができる. 文献情報からなる配列 (ソート済み) を受け, やはり配列を返す.

7 デバッグ

次のようにしておくと, デバッグに有用な情報がはき出されたりする……かもしれません.

```
local lbtdebug = require "lbt-debug"
lbtdebug.debugmode = true -- デバッグモード ON
```

-- 以下スタイルファイルの処理