BIBTEX

THE CONTEXT WAY

Contents

Introduction

This manual describes how MkIV handles bibliographies. Support in ConTEXt started in MkIIfor bibTEX, using a module written by Taco Hoekwater. Later his code was adapted to MkIV, but because users demanded more, I decided that reimplementing made more sense than patching. In particular, through the use of Lua, the bibTEX data files can be easily directly parsed, thus liberating ConTEXt from the dependency on an external bibTEX executable. The CritEd project (by Thomas Schmitz, Alan Braslau, Luigi Scarso and myself) was a good reason to undertake this rewrite. As part that project users were invited to come up with ideas about extensions. Not all of them are (yet) honored, but the rewrite makes more functionality possible.

This manual is dedicated to Taco Hoekwater who in a previous century implemented the first bibTEX module and saw it morf into a TEX–Lua hybrid in this century. The fact that there was support for bibliographies made it possible for users to use ConTEXt in an academic environment, dominated by bibliographic databases encoded in the bibTEX format.

Hans Hagen
PRAGMA ADE
Hasselt NL

1 The database

The bibTEX format is rather popular in the TEX community and even with its shortcomings it will stay around for a while. Many publication websites can export and many tools are available to work with this database format. It is rather simple and looks a bit like Lua tables. Unfortunately the content can be polluted with non-standardized TEX commands which complicates pre- or postprocessing outside TEX. In that sense a bibTEX database is often not coded neutrally. Some limitations, like the use of commands to encode accented characters root in the ascii world and can be bypassed by using utf instead (as handled somewhat in LATEX through extensions such as `bibtex8`).

The normal way to deal with a bibliography is to refer to entries using a unique tag or key. When a list of entries is typeset, this reference can be used for linking purposes. The typeset list can be processed and sorted using the `bibtex` program that converts the database into something more TEX friendly (a `.bbl` file). I never used the program myself (nor bibliographies) so I will not go into too much detail here, if only because all I say can be wrong.

In ConTEXt we no longer use the `bibtex` program: we just use database files and deal with the necessary manipulations directly in ConTEXt. One or more such databases can be used and combined with additional entries defined within the document. We can have several such datasets active at the same time.

A bibTEX file looks like this:

```
@Article{sometag,
    author  = "An Author and Another One",
    title   = "A hopefully meaningful title",
    journal = maps,
    volume  = "25",
    number  = "2",
    pages   = "5--9",
    month   = mar,
    year    = "2013",
    ISSN    = "1234-5678",
}
```

Normally a value is given between quotes (or curly brackets) but single words are also OK (there is no real benefit in not using quotes, so we advise to always use them). There can be many more fields and instead of strings one can use predefined shortcuts. The title for example quite often contains TEX macros. Some fields, like `pages` have funny characters such as the endash (typically as `--`) so we have a mixture of data and typesetting directives. If you are covering non–english references, you often need characters that are not in the ascii subset but ConTEXt is quite happy with utf. If your database file uses old-fashioned TEX accent commands then these will be internally converted automatically to utf. Commands (macros) are converted to an indirect call, which is quite robust.

The bibTEX files are loaded in memory as Lua table but can be converted to xml so that we can access them in a more flexible way, but that is a subject for specialists.

In the old MkII setup we have two kinds of entries: the ones that come from the bibTEX run and user supplied ones. We no longer rely on bibTEX output but we do still support the user supplied definitions. These were in fact prepared in a way that suits the processing of bibTEX generated entries. The next variant reflects the ConTEXt recoding of the old bibTEX output.

```
\startpublication[k=Hagen:Second,t=article,a={Hans Hagen},y=2013,s=HH01]
    \artauthor[]{Hans}[H.]{}{Hagen}
    \arttitle{Who knows more?}
    \journal{MyJournal}
    \pubyear{2013}
    \month{8}
    \volume{1}
    \issue{3}
    \issn{1234-5678}
    \pages{123--126}
\stoppublication
```

The split `\artauthor` fields are collapsed into a single `author` field as we deal with the splitting later when it gets parsed in Lua. The `\artauthor` syntax is only kept around for backward compatibility with the previous use of bibTEX.

In the new setup we support these variants as well:

```
\startpublication[k=Hagen:Third,t=article]
    \author{Hans Hagen}
    \title{Who knows who?}
    ...
\stoppublication
```

and

```
\startpublication[tag=Hagen:Third,category=article]
    \author{Hans Hagen}
    \title{Who knows who?}
    ...
\stoppublication
```

and

```
\startpublication
    \tag{Hagen:Third}
    \category{article}
    \author{Hans Hagen}
    \title{Who knows who?}
    ...
\stoppublication
```

Because internally the entries are Lua tables, we also support loading of Lua based definitions:

```
return {
    ["Hagen:First"] = {
        author   = "Hans Hagen",
        category = "article",
        issn     = "1234-5678",
        issue    = "3",
        journal  = "MyJournal",
        month    = "8",
        pages    = "123--126",
        tag      = "Hagen:First",
        title    = "Who knows nothing?",
        volume   = "1",
        year     = "2013",
    },
}
```

Files set up like this can be loaded too. The following xml input is rather close to this, and is also accepted as input.

```
<?xml version="2.0" standalone="yes" ?>
<bibtex>
    <entry tag="Hagen:First" category="article">
        <field name="author">Hans Hagen</field>
        <field name="category">article</field>
        <field name="issn">1234-5678</field>
        <field name="issue">3</field>
        <field name="journal">MyJournal</field>
        <field name="month">8</field>
        <field name="pages">123--126</field>
        <field name="tag">Hagen:First</field>
        <field name="title">Who knows nothing?</field>
        <field name="volume">1</field>
        <field name="year">2013</field>
    </entry>
</bibtex>
```

Todo: Add some remarks about loading EndNote and RIS formats, but first we need to complete the tag mapping (on Alan's plate).

So the user has a rather wide choice of formatting style for bibliography database files.

You can load more data than you actually need. Only entries that are referred to explicitly through the `\cite` and `\nocite` commands will be shown in lists. We will cover these details later.

2 Commands in entries

One unfortunate aspect commonly found in bibTEX files is that they often contain TEX commands. Even worse is that there is no standard on what these commands can be and what they mean, at least not formally, as bibTEX is a program intended to be used with many variants of TEX style: plain, LATEX, and others. This means that we need to define our use of these typesetting commands. However, in most cases, they are just abbreviations or font switches and these are often known. Therefore, ConTEXt will try to resolve them before reporting an issue. In the log file there is a list of commands that has been seen in the loaded databases. For instance, loading `tugboat.bib` gives a long list of commands of which we show a small set here:

```
publications > start used btx commands

publications > standard CONTEXT 1 known
publications > standard ConTeXt 4 known
publications > standard TeXLive 3 KNOWN
publications > standard eTeX    1 known
publications > standard hbox    6 known
publications > standard sltt    1 unknown

publications > stop used btxcommands
```

You can define unknown commands, or overload existing definitions in the following way:

```
\definebtxcommand\TUB {TUGboat}
\definebtxcommand\sltt{\tt}
\definebtxcommand\<#1>{\type{#1}}
```

Unknown commands do not stall processing, but their names are then typeset in a monospaced font so they probably stand out for proofreading. You can access the commands with `\btxcommand{...}`, as in:

```
commands like \btxcommand{MySpecialCommand} are handled in an indirect way
```

As this is an undefined command we get: “commands like MySpecialCommand are handled in an indirect way”.

??

3 Datasets

Normally in a document you will use only one bibliographic database, whether or not distributed over multiple files. Nevertheless we support multiple databases as well which is why we talk of datasets instead. A dataset is loaded with the `\usebtxdataset` command. Although currently it is not necessary to define a (default) dataset you can best do this because in the future we might provide more options. Here are some examples:

```
\definebtxdataset[standard]

\usebtxdataset[standard][tugboat.bib]
\usebtxdataset[standard][mtx-bibtex-output.xml]
\usebtxdataset[standard][test-001-btx-standard.lua]
```

These three suffixes are understood by the loader. Here the dataset has the name `standard` and the three database files are merged, where later entries having the same tag overload previous ones. Definitions in the document source (coded in TEX speak) are also added, and they are saved for successive runs. This means that if you load and define entries, they will be known at a next run beforehand, so that references to them are independent of when loading and definitions take place.

```
\setupbtxdataset [.1.] [.2.]
                 OPTIONAL OPTIONAL
1  IDENTIFIER
2  ...
```

```
\definebtxdataset [.1.] [.2.]
                          OPTIONAL
1  IDENTIFIER
2  inherits from \setupbtxdataset
```

```
\usebtxdataset [.1.] [.2.]
1  IDENTIFIER
2  FILE
```

In this document we use some example databases, so let's load one of them now:

```
\definebtxdataset[example]

\usebtxdataset[example][mkiv-publications.bib]
```

You can ask for an overview of entries in a dataset with:

```
\showbtxdatasetfields[example]
```

this gives:

```
tag       category  fields
demo-001  book      author index title year
demo-002  book      crossref index year
demo-003  book      author comment index title year
demo-004  book      author comment index title year
demo-005  book      author doi index language pages serial title url year
demo-006  book      author comment index title year
```

You can set the current active dataset with

```
\setbtxdataset[standard]
```

but most publication-related commands accept optional arguments that denote the dataset and references to entries can be prefixed with a dataset identifier.. More about that later.

Sometimes you want to check a database. One way of doing that is the following:

```
\definebtxdataset[check]

\usebtxdataset[check][mkiv-publications-check.bib]

\showbtxdatasetcompleteness[check]
```

The database like like this:

```
@BOOK{test1,
  title = {Title 1},
  author = {Author 1}
  publisher = {Thomas and Alan},
  year = {2015},
  editor = {Thomas Schmitz and Alan Braslau},
  volume = {1},
}

@BOOK{test2,
  title = {Title 2},
  author = {Author 2}
  crossref = {test1},
  volume = {2},
}

@BOOK{test3,
  title = {Title 3},
  author = {Author 3}
}
```

The completeness check shows (with green field names) the required fields and when one is missing this is indicated in red. In blue we show what gets inherited.

```
tag               test1
```

author	Author 1
editor	Thomas Schmitz and Alan Braslau
title	Title 1
publisher	Thomas and Alan
year	2015
volume	1

tag	test2	=> test1
author	Author 2	
editor	Thomas Schmitz and Alan Braslau	
title	Title 2	
publisher	Thomas and Alan	
year	2015	
volume	2	

tag	test3
author	Author 3
title	Title 3
publisher	[missing]
year	[missing]

4 Renderings

A list of publications can be rendered at any place in the document. A database can be much larger than needed for a document. The same is true for the fields that make up an entry. Here is the list of fields that are currently handled, but of course there can be additional ones:

`abstract`, `address`, `annotate`, `assignee`, `author`, `bibnumber`, `booktitle`, `chapter`, `comment`, `country`, `day`, `dayfiled`, `doi`, `edition`, `editor`, `eprint`, `howpublished`, `institution`, `isbn`, `issn`, `journal`, `key`, `keyword`, `keywords`, `language`, `lastchecked`, `month`, `monthfiled`, `names`, `nationality`, `note`, `notes`, `number`, `organization`, `pages`, `publisher`, `revision`, `school`, `series`, `size`, `title`, `type`, `url`, `volume`, `year`, `yearfiled`

If you want to see what publications are in the database, the easiest way is to ask for a complete list:

```
\definebtxrendering
  [example]
  [dataset=example,
   method=local,
   alternative=apa]
\placelistofpublications % \placebtxrendering
  [example]
  [criterium=all]
```

This gives:

1 Hagen, H. and Otten, T. (1996). *Typesetting education documents*.

2 Scarso, L. (2021). *Designing high speed trains properly!*.

3 author (year). *title*. pages p.

4 Hagen, H. (2013). *bibTEX, the ConTEXt way*.

5 van Marle Hans Hagen, K. and Otten, T. (2014). *Why do we always have lack of time?*.

The rendering itself is somewhat complex to set up because we have not only many different standards but also many fields that can be set up. This means that there are several commands involved. Often there is a prescribed style to render bibliographic descriptions, for example `apa`. A rendering is setup and defined with:

```
\setupbtxrendering [.1.] [.2=.]
                    OPTIONAL
1  IDENTIFIER

2  alternative  =  TEXT = apa
   dataset      =  TEXT = standard
   setups       =  TEXT = btx:rendering:apa
   method       =  local global none force = global
   sorttype     =  short reference dataset default
   criterium    =  TEXT
   refcommand   =  TEXT = authoryears
   numbering    =  yes cite = yes
   width        =  DIMENSION auto = auto
   distance     =  DIMENSION = 1.5\emwidth
   continue     =  yes no = no
   repeat       =  yes no = no
```

```
\definebtxrendering [.1.] [.2.] [.3=.]
                             OPTIONAL  OPTIONAL
1  IDENTIFIER

2  IDENTIFIER

3  inherits from \setupbtxrendering
```

And a list of such descriptions is generated with:

```
\placebtxrendering
```

A dataset can have all kind of entries:

Each has its own rendering variant. To keep things simple we have their settings separated. However, these settings are shared for all rendering alternatives. In practice this is seldom a problem in a publication as only one rendering alternative will be active. If this be not sufficient, you can always group local settings in a setup and hook that into the specific rendering.

```
\setupbtxlistvariant [.1.] [.2.]
                     OPTIONAL
1  IDENTIFIER

2  separator           =  TEXT
   namesep             =  TEXT = ,
   lastnamesep         =  TEXT =  and
   finalnamesep        =  TEXT =  and
   firstnamesep        =  TEXT =
   juniorsep           =  TEXT =
   vonsep              =  TEXT =
   surnamesep          =  TEXT = ,
   surnamejuniorsep    =  TEXT
   juniorjuniorsep     =  TEXT
   surnamefirstnamesep =  TEXT = ,
   surnameinitialsep   =  TEXT = ,
   etallimit           =  TEXT = 5
   etaldisplay         =  TEXT = 5
   etaltext            =  TEXT =  others
   monthconversion     =  number month month:mnem = number
   authorconversion    =  name normal inverted normalshort invertedshort = normal
```

```
\definebtxlistvariant [...]

*  IDENTIFIER
```

Examples of list variants are:

```
setupbtxlistvariant : artauthor

no specific settings

setupbtxlistvariant : author

no specific settings

setupbtxlistvariant : editor

no specific settings
```

The exact rendering of list entries is determined by the `alternative` key and defaults to `apa` which uses definitions from `publ-imp-apa.mkiv`. If you look at that file you will see that each category has its own setup. You may also notice that additional tests are needed to make sure that empty fields don't trigger separators and such.

There are a couple of accessors and helpers to get the job done. When you want to fetch a field from the current entry you use `\btxfield`. In most cases you want to make sure this field has a value, for instance because you don't want fences or punctuation that belongs to a field.

```
\btxdoif {title} {
    \bold{\btxfield{title}},
}
```

There are three test macros:

```
\btxdoifelse{fieldname}{action when found}{action when not found}
\btxdoif    {fieldname}{action when found}
\btxdoifnot {fieldname}                   {action when not found}
```

An extra conditional is available for testing interactivity:

```
\btxdoifelseinteraction{action when true}{action when false}
```

In addition there is also a conditional `\btxinteractive` which is more efficient, although in practice efficiency is not so important here.

There are three commands to flush data:

`\btxfield`	fetch a explicit field (e.g. `year`)
`\btxdetail`	fetch a derived field (e.g. `short`)
`\btxflush`	fetch a derived or explicit field

Normally you can use `\btxfield` or `\btxflush` as derived fields just like analyzed author fields are flushed in a special way. There is experimental support for so called manipulators. You can for instance say this:

```
\btxflush{lowercase->title}
```

A sequence of manipulators is applied to fetched field, where a sequence is one or more manipulators:

```
\btxflush{stripperiod->uppercase->title}
```

Some actions are recognized (built-in) but you can also use actions from other namespaces, like in:

```
\btxflush{converters.Word -> title}
```

Watch how we can use spaces around the `->` which is nicer for wrapped around usage. Eventually, we might put some more function in the default namespace.

You can improve readability by using setups, for instance:

```
\btxdoifelse {author} {
    \btxsetup{btx:apa:author:yes}
} {
    \btxsetup{btx:apa:author:nop}
}
```

Keep in mind that normally you don't need to mess with definitions like this because standard rendering styles are provided. These styles use a few helpers that inject symbols but also take care of leading and trailing spaces:

`\btxspace`	before after
`\btxperiod`	before. after
`\btxcomma`	before, after

\btxlparent	before (after
\btxrparent	before) after
\btxlbracket	before [after
\btxrbracket	before] after

So, the previous example setup can be rewritten as:

```
\btxdoif {title} {
    \bold{\btxfield{title}}
    \btxcomma
}
```

There is a special command for rendering a (combination) of authors:

```
\btxflushauthor{author}
\btxflushauthor{editor}
\btxflushauthor[inverted]{editor}
```

Instead of the last one you can also use:

```
\btxflushauthorinverted{editor}
```

You can use a (configurable) default or pass directives: Valid directives are

conversion	**rendering**
inverted	the Frog jr, Kermit
invertedshort	the Frog jr, K
normal	Kermit, the Frog, jr
normalshort	K, the Frog, jr

The list itself is not a list in the sense of a regular ConTEXt structure related list. We do use the list mechanism to keep track of used entries but that is mostly because we can then reuse filtering mechanisms. The actual rendering of a reference and entry runs on top of so called constructions (other examples of constructions are descriptions, enumerations and notes).

```
\setupbtxlist [.*=.]

*  alternative  =  TEXT = left
   style        =  TEXT
   color        =  TEXT
   headstyle    =  TEXT
   headcolor    =  TEXT
   width        =  DIMENSION = 4\emwidth
   distance     =  DIMENSION = \emwidth
   hang         =  NUMBER
   align        =
   headalign    =
   margin       =  cd:yes cd:no = no
   before       =  COMMAND = \blank
   inbetween    =  COMMAND = \blank
   after        =  COMMAND = \blank
   display      =  cd:yes cd:no = yes
   command      =  COMMAND
```

You need to be aware what command is used to achieve the desired result. For instance, in order to put parentheses around a number reference you say:

```
\setupbtxlistvariant
  [num]
  [left=(,
   right=)]
```

If you want automated width calculations, the following does the trick:

```
\setupbtxrendering
  [standard]
  [width=auto]
```

but if you want to control it yourself you say something:

```
\setupbtxrendering
  [width=none]

\setupbtxlist
  [standard]
  [width=3cm,
   distance=\emwidth,
   color=red,
   headcolor=blue,
   headalign=flushright]
```

In most cases the defaults will work out fine.

Normally the references are numbered using one conunter for the whole document. If you want each list to have its own number, then you can set the `continue` parameter:

```
\setupbtxrendering[continue=no]
```

In a simlar fashion you can influence if references are included only once of in each list:

```
\setupbtxrendering[repeat=yes]
```

5 Citations

Citations are references to bibliographic entries that normally show up in lists someplace in the document: at the end of a chapter, in an appendix, at the end of an article, etc. We discussed the rendering of these lists in the previous chapter. A citation is normally pretty short as its main purpose is to refer uniquely to a more detailed description. But, there are several ways to refer, which is why the citation subsystem is configurable and extensible. Just look at the following commands:

```
\cite[author][example::demo-003]
\cite[authoryear][example::demo-003]
\cite[authoryears][example::demo-003]
\cite[author][example::demo-003,demo-004]
\cite[authoryear][example::demo-003,demo-004]
\cite[authoryears][example::demo-003,demo-004]
\cite[author][example::demo-004,demo-003]
\cite[authoryear][example::demo-004,demo-003]
\cite[authoryears][example::demo-004,demo-003]
```

(Hagen and Otten)
(Hagen and Otten, 1996)
(Hagen and Otten, 1996)
(Hagen and Otten and Scarso)
(Hagen and Otten, 1996 and Scarso, 2021)
(Hagen and Otten, 1996 and Scarso, 2021)
(Scarso and Hagen and Otten)
(Scarso, 2021 and Hagen and Otten, 1996)
(Scarso, 2021 and Hagen and Otten, 1996)

The first argument is optional.

```
\cite [...] [...]
       OPTIONAL
1  IDENTIFIER

2  IDENTIFIER
```

You can tune the way a citation shows up:

```
\setupbtxcitevariant[author]     [sorttype=author,color=darkyellow]
\setupbtxcitevariant[authoryear] [sorttype=author,color=darkyellow]
\setupbtxcitevariant[authoryears][sorttype=author,color=darkyellow]

\cite[author][example::demo-004,demo-003]
\cite[authoryear][example::demo-004,demo-003]
\cite[authoryears][example::demo-004,demo-003]
```

Here we sort the authors and color the citation:

(Scarso and Hagen and Otten)

(Scarso, 2021 and Hagen and Otten, 1996)
(Scarso, 2021 and Hagen and Otten, 1996)

For reasons of backward compatibility the `\cite` command is a bit picky about spaces between the two arguments, of which the first is optional. This is a consequence of allowing its use with the key specified between curly brackets as is the traditional practice. (We do encourage users to adopt the more coherent ConTEXt syntax by using square brackets for keywords and reserving curly brackets to regroup text to be typeset.)

The `\citation` command is synonymous but is more flexible with respect to spacing of its arguments:

```
\citation[author]     [example::demo-004,demo-003]
\citation[authoryear] [example::demo-004,demo-003]
\citation[authoryears][example::demo-004,demo-003]
```

There is a whole bunch of cite options and more can be easily defined.

key	**rendering**
`author`	(author)
`authornum`	(author [3])
`authoryear`	(author, year)
`authoryears`	(author, year)
`doi`	[doi]
`key`	[demo-005]
`none`	
`num`	[3]
`page`	pages
`serial`	[5]
`short`	[<demo-005>]
`type`	[book]
`url`	[url]
`year`	(year)

Because we are dealing with database input and because we generally need to manipulate entries, much of the work is delegated to Lua. This makes it easier to maintain and extend the code. Of course TEX still does the rendering. The typographic details are controlled by parameters but not all are used in all variants. As with most ConTEXt commands, it starts out with a general setup command:

```
\setupbtxcitevariant [.1.] [.2.]
                          OPTIONAL
1  IDENTIFIER

2  alternative       =  TEXT = num
   setups            =  TEXT = btx:cite:num
   interaction       =  TEXT = start
   andtext           =  TEXT =  and
   otherstext        =  TEXT =  others
   compress          =  TEXT = no
   putsep            =  TEXT
   lastputsep        =  TEXT
   inbetween         =  TEXT =
   right             =  TEXT
   middle            =  TEXT
   left              =  TEXT
   monthconversion   =  TEXT
   authorconversion  =  name normal inverted normalshort invertedshort = name
```

On top of that we can define instances that inherit either from a given parent or from the topmost setup.

```
\definebtxcitevariant [.1.] [.2.] [.3.]
                                OPTIONAL  OPTIONAL
1  IDENTIFIER

2  IDENTIFIER

3  inherits from \setupbtxvariant
```

But, specific variants can have them overloaded:

```
setupbtxcitevariant : author

right   )
middle  ,
left    (

setupbtxcitevariant : authornum

right   )
middle  ,
left    (

setupbtxcitevariant : authoryear

compress   yes
inbetween  ,
right      )
middle     ,
left       (
```

```
setupbtxcitevariant : authoryears

compress   yes
inbetween  ,
right      )
middle     ,
left       (

setupbtxcitevariant : doi

right   ]
middle  ,
left    [

setupbtxcitevariant : key

right   ]
middle  ,
left    [

setupbtxcitevariant : none

no specific settings

setupbtxcitevariant : num

compress  yes
right     ]
middle    ,
left      [

setupbtxcitevariant : page

middle  ,

setupbtxcitevariant : serial

right   ]
middle  ,
left    [

setupbtxcitevariant : short

right   ]
middle  ,
left    [
```

```
setupbtxcitevariant : type

right   ]
middle  ,
left    [

setupbtxcitevariant : url

right   ]
middle  ,
left    [

setupbtxcitevariant : year

right   )
middle  ,
left    (
```

A citation variant is defined in several steps and if you really want to know the dirty details, you should look into the `publ-imp-*.mkiv` files. Here we stick to the concept.

```
\startsetups btx:cite:author
    \btxcitevariant{author}
\stopsetups
```

You can overload such setups if needed, but that only makes sense when you cannot configure the rendering with parameters. The `\btxcitevariant` command is one of the build in accessors and it calls out to Lua where more complex manipulation takes place if needed. If no manipulation is known, the field with the same name (if found) will be flushed. A command like `\btxcitevariant` assumes that a dataset and specific tag has been set. This is normally done in the wrapper macros, like `\cite`. For special purposes you can use these commands

```
\setbtxdataset[example]
\setbtxentry[hh2013]
```

But don't expect too much support for such low level rendering control.

Unless you use `criterium=all` only publications that are cited will end up in the lists. You can force a citation into a list using `\usecitation`, for example:

```
\usecitation[example::demo-004,demo-003]
```

This command has two synonyms: `\nocite` and `\nocitation` so you can choose whatever fits you best.

```
\nocite [*...]

*  IDENTIFIER
```

6 The LUA view

Because we manage data at the Lua end it is tempting to access it there for other purposes. This is fine as long as you keep in mind that aspects of the implementation may change over time, although this is unlikely once the modules become stable.

The entries are collected in datasets and each set has a unique name. In this document we have the set named `example`. A dataset table has several fields, and probably the one of most interest is the `luadata` field. Each entry in this table describes a publication:

```
t={
 ["author"]="Hans Hagen",
 ["category"]="book",
 ["index"]=1,
 ["tag"]="demo-001",
 ["title"]="\\btxcmd{BIBTEX}, the \\btxcmd{CONTEXT}\\ way",
 ["year"]="2013",
}
```

This is `publications.datasets.example.luadata["demo-001"]`. There can be a companion entry in the parallel `details` table.

```
t={
 ["author"]={
  {
   ["firstnames"]={ "Hans" },
   ["initials"]={ "H" },
   ["original"]="Hans Hagen",
   ["surnames"]={ "Hagen" },
   ["vons"]={},
  },
 },
 ["short"]="Hag13",
}
```

These details are accessed as `publications.datasets.example.details["demo-001"]` and by using a separate table we can overload fields in the original entry without losing the original.

You can loop over the entries using regular Lua code combined with MkIV helpers:

```
local dataset = publications.datasets.example

context.starttabulate { "|l|l|l|" }
for tag, entry in table.sortedhash(dataset.luadata) do
    local detail = dataset.details[tag] or { }
    context.NC() context.type(tag)
    context.NC() context(detail.short)
    context.NC() context(entry.title)
```

```
    context.NC() context.NR()
end
context.stoptabulate()
```

This results in:

demo-001 Hag13 bibTEX, the ConTEXt way
demo-002 Hag14 bibTEX, the ConTEXt way
demo-003 HO96 Typesetting education documents
demo-004 Sca21 Designing high speed trains properly!
demo-005 aut00 title
demo-006 HO14 Why do we always have lack of time?

You can manipulate a dataset after loading. Of course this assumes that you know what kind of content you have and what you need for rendering. As example we load a small dataset.

```
\definebtxdataset[drumming]
\usebtxdataset[drumming][mkiv-publications.lua]
```

Because we're going to do some Lua, we could also have loaded the dataset with:

```
publications.load("drumming","mkiv-publications.lua","lua")
```

The dataset has three entries:

```
return {

    ["GH0001"] = {

        category  = "book",
        title     = "Rhythmic Illusions",
        subtitle  = "for drums",
        author    = "Gavin Harrison",
        publisher = "Warner",
        isbn      = "1576236870",
        year      = "1996",
        comment   = "plus cd",
    },

    ["GH0002"] = { -- no reference in brittisch library

        category  = "book",
        title     = "Rhythmic Perspectives",
        subtitle  = "a multidimensional study of rhythmic composition",
        author    = "Gavin Harrison",
        publisher = "Alfred Publishing Co., Inc",
        year      = "1999",
        comment   = "plus cd",
    },
```

```
    ["GH0003"] = {

        category  = "book",
        title     = "Rhythmic Designs",
        subtitle  = "a study of practical creativity",
        author    = "Gavin Harrison and Terry Branham",
        publisher = "Hudson",
        year      = "2010",
        comment   = "plus dvd",

    },

}
```

As you can see, we can have a subtitle. We will combine the title and subtitle into one:

```
\startluacode
for tag, entry in next, publications.datasets.drumming.luadata do
    if entry.subtitle then
        if entry.title then
            entry.title = entry.title .. ", " .. entry.subtitle
        else
            entry.title = entry.subtitle
        end
        entry.subtitle = nil
        logs.report("btx","combining title and subtitle of entry tagged %a",tag)
    end
end
\stopluacode
```

We can now typeset the entries with:

```
\definebtxrendering[drumming][dataset=drumming,method=dataset]
\placebtxrendering[drumming]
```

Because we just want to show the entries, and have no citations that force them to be shown, we have to the `method` to `dataset`.[1]

1 Harrison, G. (1996). *Rhythmic Illusions, for drums*. Warner.

2 Harrison, G. (1999). *Rhythmic Perspectives, a multidimensional study of rhythmic composition*. Alfred Publishing Co., Inc.

3 Harrison, G. and Branham, T. (2010). *Rhythmic Designs, a study of practical creativity*. Hudson.

[1] Gavin Harrison is in my opinion one of the most creative, diverse and interesting drummers of our time. It's also fascinating to watch him play and a welcome distraction from writing code and manuals.

7 The XML view

The luadata table can be converted into an xml representation. This is a follow up on earlier experiments with an xml-only approach. I decided in the end to stick to a Lua approach and provide some simple xml support in addition.

Once a dataset is accessible as xml tree, you can use the regular \xml... commands. We start with loading a dataset, in this case from just one file.

```
\usebtxdataset[tugboat][tugboat.bib]
```

The dataset has to be converted to xml:

```
\convertbtxdatasettoxml[tugboat]
```

The tree is now accessible by its root reference btx:tugboat. If we want simple field access we can use a few setups:

```
\startxmlsetups btx:initialize
    \xmlsetsetup{#1}{bibtex|entry|field}{btx:*}
    \xmlmain{#1}
\stopxmlsetups

\startxmlsetups btx:field
    \xmlflushcontext{#1}
\stopxmlsetups

\xmlsetup{btx:tugboat}{btx:initialize}
```

The two setups are predefined in the core already, but you might want to change them. They are applied in for instance:

```
\starttabulate[|||]
    \NC \type {tag}   \NC \xmlfirst {btx:tugboat}
        {/bibtex/entry[string.find(@tag,'Hagen')]/attribute('tag')}
    \NC \NR
    \NC \type {title} \NC \xmlfirst {btx:tugboat}
        {/bibtex/entry[string.find(@tag,'Hagen')]/field[@name='title']}
    \NC \NR
\stoptabulate
```

tag
title

```
\startxmlsetups btx:demo
    \xmlcommand
        {#1}
        {/bibtex/entry[string.find(@tag,'Hagen')][1]}{btx:table}
\stopxmlsetups

\startxmlsetups btx:table
```

```
\starttabulate[|||]
    \NC \type {tag}   \NC \xmlatt{#1}{tag} \NC \NR
    \NC \type {title} \NC \xmlfirst{#1}{/field[@name='title']} \NC \NR
\stoptabulate
\stopxmlsetups

\xmlsetup{btx:tugboat}{btx:demo}
```

Here is another example:

```
\startxmlsetups btx:row
    \NC \xmlatt{#1}{tag}
    \NC \xmlfirst{#1}{/field[@name='title']}
    \NC \NR
\stopxmlsetups

\startxmlsetups btx:demo
    \xmlfilter {#1} {
        /bibtex
        /entry[@category='article']
        /field[@name='author' and (find(text(),'Knuth') or find(text(),'DEK'))]
        /../command(btx:row)
    }
\stopxmlsetups

\starttabulate[|||]
    \xmlsetup{btx:tugboat}{btx:demo}
\stoptabulate
```

A more extensive example is the following. Of course this assumes that you know what xml support mechanisms and macros are available.

```
\startxmlsetups btx:getkeys
    \xmladdsortentry{btx}{#1}{\xmlfilter{#1}{/field[@name='author']/text()}}
    \xmladdsortentry{btx}{#1}{\xmlfilter{#1}{/field[@name='year'  ]/text()}}
    \xmladdsortentry{btx}{#1}{\xmlatt{#1}{tag}}
\stopxmlsetups

\startxmlsetups btx:sorter
    \xmlresetsorter{btx}
  % \xmlfilter{#1}{entry/command(btx:getkeys)}
    \xmlfilter{#1}{
        /bibtex
        /entry[@category='article']
        /field[@name='author' and find(text(),'Knuth')]
        /../command(btx:getkeys)}
    \xmlsortentries{btx}
    \starttabulate[|||||]
        \xmlflushsorter{btx}{btx:entry:flush}
    \stoptabulate
```

```
\stopxmlsetups

\startxmlsetups btx:entry:flush
    \NC \xmlfilter{#1}{/field[@name='year'  ]/context()}
    \NC \xmlatt{#1}{tag}
    \NC \xmlfilter{#1}{/field[@name='author']/context()}
    \NC \NR
\stopxmlsetups

\xmlsetup{btx:tugboat}{btx:sorter}
```

The original data is stored in a Lua table, hashed by tag. Starting with Lua 5.2 each run of Lua gets a different ordering of such a hash. In older versions, when you looped over a hash, the order was undefined, but the same as long as you used the same binary. This had the advantage that successive runs, something we often have in document processing gave consistent results. In today's Lua we need to do much more sorting of hashes before we loop, especially when we save multi–pass data. It is for this reason that the xml tree is sorted by hash key by default. That way lookups (especially the first of a set) give consistent outcomes.

8 Standards

The rendering of bibliographic entries is often standardized and prescribed by the publisher. If you submit an article to a journal, normally it will be reformatted (or even re-keyed) and the rendering will happen at the publishers end. In that case it may not matter how entries were rendered when writing the publication, because the publisher will do it his or her way. This means that most users probably will stick to the standard apa rules and for them we provide some configuration. Because we use setups it is easy to overload specifics. If you really want to tweak, best look in the files that deal with it.

Many standards exist and support for other renderings may be added to the core. Interested users are invited to develop and to test alternate standard renderings according to their needs.

Todo: maybe a list of categories and fields.

9 Cleaning up

Although the bibTEX format is reasonably well defined, in practice there are many ways to organize the data. For instance, one can use predefined string constants that get used (either or not combined with other strings) later on. A string can be enclosed in curly braces or double quotes. The strings can contain TEX commands but these are not standardized. The databases often have somewhat complex ways to deal with special characters and the use of braces in their definition is also not normalized.

The most complex to deal with are the fields that contain names of people. At some point it might be needed to split a combination of names into individual ones that then get split into title, first name, optional inbetweens, surname(s) and additional: `Prof. Dr. Alfred B. C. von Kwik Kwak Jr. II and P. Q. Olet` is just one example of this. The convention seems to be not to use commas but `and` to separate names (often each name will be specified as lastname, firstname).

We don't see it as challenge nor as a duty to support all kinds of messy definitions. Of course we try to be somewhat tolerant, but you will be sure to get better results if you use nicely setup, consistent databases.

Todo: maybe some examples of bad.

10 Transition

In the original bibliography support module usage was as follows (example taken from the contextgarden wiki):

```
% engine=pdftex

\usemodule[bib]
\usemodule[bibltx]

\setupbibtex
  [database=xampl]

\setuppublications
  [numbering=yes]

\starttext
    As \cite [article-full] already indicated, bibtex is a \LATEX||centric
    program.

    \completepublications
\stoptext
```

For MkIV the modules were partly rewritten and ended up in the core so the two commands were no longer needed. The overhead associated with the automatic loading of the bibliography macros can be neglected these days, so standardized modules such as `bib` are all being moved to the core and do not need to be explicitly loaded.

The first `\setupbibtex` command in this example is needed to bootstrap the process: it tells what database has to be processed by bibTEX between runs. The second `\setuppublications` command is optional. Each citation (tagged with `\cite`) ends up in the list of publications.

In the new approach we no longer use bibTEXso we don't need to setup bibTEX. Instead we define dataset(s). We also no longer set up publications with one command, but have split that up in rendering-, list-, and cite-variants. The basic `\cite` command remains. The above example becomes:

```
\definebtxdataset
  [document]

\usebtxdataset
  [document]
  [mybibfile.bib]

\definebtxrendering
  [document]

\setupbtxrendering
  [document]
```

```
  [numbering=yes]

\starttext
    As \cite [article-full] already indicated, bibtex is a \LATEX||centric
    program.

    \completebtxrendering[document]
\stoptext
```

So, we have a few more commands to set up things. If you intend to use just a single dataset and rendering, the above preamble can be simplified to:

```
\usebtxdataset
  [mybibfile.bib]

\setupbtxrendering
  [numbering=yes]
```

But keep in mind that compared to the old MkII derived method we have moved some of the options to the rendering, list and cite setup variants.

Another difference is now the use of lists. When you define a rendering, you also define a list. However, all entries are collected in a common list tagged `btx`. Although you will normally configure a rendering you can still set some properties of lists, but in that case you need to prefix the list identifier. In the case of the above example this is `btx:document`.

11 MLBIBTEX

Todo: how to plug in MLbibTEX for sorting and other advanced operations.

12 Extensions

As TEX and Lua are both open and accessible in ConTEXt it is possible to extend the functionality of the bibliography related code. For instance, you can add extra loaders.

```
function publications.loaders.myformat(dataset,filename)
    local t = { }
    -- Load data from 'filename' and convert it to a Lua table 't' with
    -- the key as hash entry and fields conforming the luadata table
    -- format.
    loaders.lua(dataset,t)
end
```

This then permits loading a database (into a dataset) with the command:

```
\usebtxdataset[standard][myfile.myformat]
```

The `myformat` suffix is recognized automatically. If you want to use another suffix, you can do this:

```
\usebtxdataset[standard][myformat::myfile.txt]
```

13 Searching

Finding the right key in a database can be a pain. On the other hand, asking for a wildcard also makes no sense. Nevertheless we provide a mechanism for matching a query. For this we load a small bibliographic database:

```
\usebtxdataset[graph][mkiv-publications-graph.bib]
```

We could switch to this base using:

```
\setbtxdataset[graph]
```

but instead we will use a prefix. For instance, if we have this in our source:

```
searching give a few hits, so we get: \cite [ graph :: match ( author:cleveland and
year:1993 ) ].
```

We will get: “searching give a few hits, so we get: [1 and 2].”. Of course this assumes that we also typeset a list of referred to references, so let’s do that:

```
\definebtxrendering[graph][dataset=graph]
\placebtxrendering[graph][criterium=chapter]
```

We get:

1 Cleveland, W. S. (1993). *Visualizing Data*. Summit, New Jersey, Hobart Press.

2 Cleveland, W. S. (1993). A Model for Studying Display Methods of StatisticalGraphics (with discussion). *Journal of Computational and Statistical Graphics*, *2*, 323–343.

3 Cleveland, W. S. (1985, revised 1994). *The Elements of Graphing Data*. Summit, New Jersey, Hobart Press.

Let’s look in more detail at the `\cite` command. In order to distinguish efficiently between a normal reference and a more clever one, we use the `match` keyword:

The handler is rather tolerant for spaces:

Which is handy if you have long queries that wrap around in the source code. Of course the `dataset::` prefix is optional in which case the current dataset is taken.

A query eventually becomes a Lua expression so you can use helpers to achieve your goal. As a convenience there are some shortcuts to access fields. The following examples demonstrate this:

```
match(author:hagen)
match(author:hagen and author:hoekwater and year:1990-2010)
match(author:"Bogusław Jackowski")
match(author:"Bogusław Jackowski" and (tonumber(field:year) or 0) > 2000)
```

You can use quotes when spaces are involved. Of course you can use other characters that the basic alphabet. Ranges (of numbers) are recognized. String lookups are partial and case insensitive.[2]

```
Wildcards: \cite [graph::match(author:cleve)].
```

We get three entries: “Wildcards: [1–3].”.

[2] At the time of this writing, may 2014, this mechanism is still somewhat experimental.

14 Authors

The most complicated part of the rendering is authors. The way names are made up is quite different and depends on culture, history, country and personal taste. For instance, in the Netherlands you seldom see junior or senior being used, but in the Unites States this is quite common. Then there is the matter of several authors cooperating.

Herw we give some examples of rendering. The authornames are taken from the database, analyzed, split and depending on the demand, reconstructed.

`\citation[alternative=author,authorconversion=...][example::demo-001]`

`name`	(Hagen)
`normal`	(Hans Hagen)
`normalshort`	(H. Hagen)
`inverted`	(Hagen, Hans)
`invertedshort`	(Hagen, H.)

`\citation[alternative=author,authorconversion=...][example::demo-003]`

`name`	(Hagen and Otten)
`normal`	(Hans Hagen and Ton Otten)
`normalshort`	(H. Hagen and T. Otten)
`inverted`	(Hagen, Hans and Otten, Ton)
`invertedshort`	(Hagen, H. and Otten, T.)

`\citation[alternative=author,authorconversion=...][example::demo-003,demo-004]`

`name`	(Hagen and Otten and Scarso)
`normal`	(Hans Hagen and Ton Otten and Luigi Scarso)
`normalshort`	(H. Hagen and T. Otten and L. Scarso)
`inverted`	(Hagen, Hans and Otten, Ton and Scarso, Luigi)
`invertedshort`	(Hagen, H. and Otten, T. and Scarso, L.)

`\citation[alternative=author,authorconversion=...][example::demo-006]`

`name`	(van Marle Hans Hagen and Otten)
`normal`	(Kees van Marle Hans Hagen and Ton Otten)
`normalshort`	(K. van Marle Hans Hagen and T. Otten)
`inverted`	(van Marle Hans Hagen, Kees and Otten, Ton)
`invertedshort`	(van Marle Hans Hagen, K. and Otten, T.)

15 Combining

It is possible to refer to two sources in one go. In that case the list will have one entry for two bibliographic entries.

```
Let's save numbers and refer to Bentley and Tufte with one: \cite [graph ::
Bentley1990 + Tufte1983]!
```

Indeed we get one number only: “Let’s save numbers and refer to Bentley and Tufte with one: [4]!”.

We produce the (local) list with:

```
\setupbtxrendering[graph][continue=yes,separator={; }]
\placebtxrendering[graph][criterium=chapter]
```

which shows the two entries pasted together:

4 Bentley, J. L. and Kernighan, B. W. (1990). In, *Grap—A language for Typesetting Graphs*, pages 109–146 Tenth edition Murray Hill, New Jersey, AT&T Bell Laboratories.; Tufte, E. R. (1983). In, *Visual Display of Quantitative Information* Box 430, Cheshire, Connecticut 06410, Graphics Press.

As demonstration we also specified the separator although that one is already set up by default.

You can combine citations with additional text before and/or after it. This can be done per citation. This feature is of course not that useful, as one can put text before and after a citation anyway.

```
foo bar \citation [before=<<,after=>>] [graph::Cleveland1993] foo bar
```

Gives:

foo bar << [1] >> foo bar

16 Summary

There are a lot of combinations possible and not all of them make sense. Nevertheless we show most of them here. (There will be more.)

alternative=author / compress=no

Cleveland : (Cleveland, Cleveland and Cleveland)
Tufte : (Tufte)
Bentley : (Bentley and Kernighan)
All : (Tufte, Cleveland, Bentley and Kernighan, Cleveland and Cleveland)

alternative=authoryear / compress=no

Cleveland : (Cleveland, 1993, Cleveland, 1985, revised 1994 and Cleveland, 1993)
Tufte : (Tufte, 1983)
Bentley : (Bentley and Kernighan, 1990)
All : (Tufte, 1983, Cleveland, 1993, Bentley and Kernighan, 1990, Cleveland, 1985, revised 1994 and Cleveland, 1993)

alternative=authoryear / compress=yes

Cleveland : (Cleveland, 1985, revised 1994, 1993a and 1993b)
Tufte : (Tufte, 1983)
Bentley : (Bentley and Kernighan, 1990)
All : (Tufte, 1983, Cleveland, 1985, revised 1994, 1993a and 1993b and Bentley and Kernighan, 1990)

alternative=authoryears / compress=no

Cleveland : (Cleveland, 1993, Cleveland, 1985, revised 1994 and Cleveland, 1993)
Tufte : (Tufte, 1983)
Bentley : (Bentley and Kernighan, 1990)
All : (Tufte, 1983, Cleveland, 1993, Bentley and Kernighan, 1990, Cleveland, 1985, revised 1994 and Cleveland, 1993)

alternative=authoryears / compress=yes

Cleveland : (Cleveland, 1985, revised 1994, 1993a and 1993b)
Tufte : (Tufte, 1983)
Bentley : (Bentley and Kernighan, 1990)
All : (Tufte, 1983, Cleveland, 1985, revised 1994, 1993a and 1993b and Bentley and Kernighan, 1990)

alternative=authornum / compress=no

Cleveland : (Cleveland [1], Cleveland [3] and Cleveland [2])

Tufte : (Tufte [5])
Bentley : (Bentley and Kernighan [4])
All : (Tufte [5], Cleveland [1], Bentley and Kernighan [4], Cleveland [3] and Cleveland [2])

alternative=authornum / compress=yes

Cleveland : (Cleveland [1–3])
Tufte : (Tufte [5])
Bentley : (Bentley and Kernighan [4])
All : (Tufte [5], Cleveland [1–3] and Bentley and Kernighan [4])

alternative=year / compress=no

Cleveland : (1993, 1985, revised 1994 and 1993)
Tufte : (1983)
Bentley : (1990)
All : (1983, 1993, 1990, 1985, revised 1994 and 1993)

alternative=year / compress=yes

Cleveland : (1985, revised 1994, 1993 and 1993)
Tufte : (1983)
Bentley : (1990)
All : (1983, 1985, revised 1994, 1990, 1993 and 1993)

alternative=short / compress=no

Cleveland : [<Cleveland1993>, <Cleveland1985> and <Cleveland1993a>]
Tufte : [<Tufte1983>]
Bentley : [<Bentley1990>]
All : [<Tufte1983>, <Cleveland1993>, <Bentley1990>, <Cleveland1985> and <Cleveland1993a>]

alternative=serial / compress=no

Cleveland : [3, 2 and 4]
Tufte : [5]
Bentley : [1]
All : [5, 3, 1, 2 and 4]

alternative=serial / compress=yes

Cleveland : [2–4]
Tufte : [5]
Bentley : [1]
All : [1–5]

alternative=tag / compress=no

Cleveland : [Cleveland1993, Cleveland1985 and Cleveland1993a]
Tufte : [Tufte1983]
Bentley : [Bentley1990]
All : [Tufte1983, Cleveland1993, Bentley1990, Cleveland1985 and Cleveland1993a]

alternative=key / compress=no

Cleveland : [Cleveland1993, Cleveland1985 and Cleveland1993a]
Tufte : [Tufte1983]
Bentley : [Bentley1990]
All : [Tufte1983, Cleveland1993, Bentley1990, Cleveland1985 and Cleveland1993a]

alternative=doi / compress=no

Cleveland : [<Cleveland1993>, <Cleveland1985> and 10.1080/10618600.1993.10474616]
Tufte : [<Tufte1983>]
Bentley : [<Bentley1990>]
All : [<Tufte1983>, <Cleveland1993>, <Bentley1990>, <Cleveland1985> and 10.1080/10618600.1993.10474616]

alternative=url / compress=no

Cleveland : [<Cleveland1993><Cleveland1985>http://www.tandfonline.com/doi/abs/10.1080/10618600.1993.10474616]
Tufte : [<Tufte1983>]
Bentley : [<Bentley1990>]
All : [<Tufte1983><Cleveland1993><Bentley1990><Cleveland1985>http://www.tandfonline.com/doi/abs/10.1080/10618600.1993.10474616]

alternative=type / compress=no

Cleveland : [book, book and article]
Tufte : [book]
Bentley : [incollection]
All : [book, book, incollection, book and article]

alternative=category / compress=no

Cleveland : [book, book and article]
Tufte : [book]
Bentley : [incollection]
All : [book, book, incollection, book and article]

alternative=page / compress=no

Cleveland : <Cleveland1993>, <Cleveland1985> and 323–343

Tufte : <Tufte1983>
Bentley : 109–146
All : <Tufte1983>, <Cleveland1993>, 109–146, <Cleveland1985> and 323–343

alternative=num / compress=no

Cleveland : [1, 3 and 2]
Tufte : [5]
Bentley : [4]
All : [5, 1, 4, 3 and 2]

alternative=num / compress=yes

Cleveland : [1–3]
Tufte : [5]
Bentley : [4]
All : [1–5]

We produce a local list with:

```
\setupbtxrendering[graph][continue=yes]
\placebtxrendering[graph][criterium=chapter]
```

and get a list with (new) entries:

5 Tufte, E. R. (1983). *Visual Display of Quantitative Information*. Box 430, Cheshire, Connecticut 06410, Graphics Press.

17 Notes

The move from external bibTEX processing to internal processing has the advantage that we stay within the same run. In the traditional approach we had roughly the following steps:

- the first run information is collected and written to file
- after that run the bibTEX program converts that file to another one
- successive runs use that data for references and producing lists

In the MkIV approach the bibliographic database is loaded in memory each run and processing also happens each run. On paper this looks less efficient but as Lua is quite fast, in practice performance is much better.

Probably most demanding is the treatment of authors as we have to analyze names, split multiple authors and reassemble firstnames, vons, surnames and juniors. When we sort by author sorting vectors have to be made which also has a penalty. However, in practice the user will not notice a performance degradation. We did some tests with a list of 500.000 authors, sorted them and typeset them as list (producing some 5400 dense pages in a small font and with small margins). This is typical one of these cases where using LuajitTEX saves quite time. On my machine it took just over 100 seconds to get this done. Unfortunately not all operating systems performed equally well: 32 bit versions worked fine, but 64 bit linux either crashed (stalled) the machine or ran out of memory rather fast, while MacOSX and Windows performed fine. In practice you will never run into this, unless you produce massive amounts of bibliographic entries. LuaJIT has some benefits but also some drawbacks.