

# Usage of Lua $\TeX$ module `luaindex` and Lua $\LaTeX$ Package `luaindex` for Generating Indexes

Markus Kohm\*

v0.1b

With Lua $\TeX$  it would not be a problem to call an index processor like `MakeIndex` while running Lua $\TeX$ . So the user would not longer require to call the index processor on his own. But on the other side Lua hat enough power to process the index itself. Package `luaindex` was made to do this. It consists primary of a Lua module: `luaindex.lua`. This provides functions to generate a new index (or several new indexes), add entries to it and print the index. To make the world easier there's an additional  $\LaTeX$  package: `luaindex.sty`.

## Contents

<b>1</b>	<b>Idea</b>	<b>2</b>
<b>2</b>	<b>General Options</b>	<b>2</b>
<b>3</b>	<b>Generating Index Entries</b>	<b>2</b>
<b>4</b>	<b>Print an Index</b>	<b>2</b>
<b>5</b>	<b>Known Issues</b>	<b>2</b>
<b>6</b>	<b>Implementation of Lua Module <code>luaindex.lua</code></b>	<b>3</b>

---

\*komascript@gmx.info

<b>7</b>	<b>Implementation of <math>\LaTeX</math> Package <code>luaindex.sty</code></b>	<b>12</b>
7.1	Package Startup . . . . .	12
7.2	Options . . . . .	14
7.3	Some Usual Index Commands . . . . .	15
7.4	Generation of Indexes and Index Entries . . . . .	15
7.5	Printing an Index . . . . .	19
<b>8</b>	<b>Examples</b>	<b>21</b>

## 1 Idea

We will explain this in a future release.

## 2 General Options

See implementation documentation.

## 3 Generating Index Entries

See implementation documentation.

## 4 Print an Index

See implementation documentation.

## 5 Known Issues

Currently the user documentation is not existing. Please use the implementation documentation and the example instead of. This will be changed in a future release but maybe not at a near future.

Currently there are no attributes to give the different indexes different headings. You may redefine `\indexname` before printing an index to do so. Future releases will do this simply by option.

Currently repeated pre-sort-replaces are not supported. Maybe they will in a future release.

Currently page ranges are not supported. They will in a future release.

Note: This is not even a beta version. It's only a proof of concept. Almost everything may be designed and implemented in a better kind. The author himself is just learning  $\text{Lua}\TeX$ .

Nevertheless you may report bugs and patches to [komascript@gmx.info](mailto:komascript@gmx.info).

## 6 Implementation of Lua Module `luaindex.lua`

First of all we define a new module named `luaindex`. All variables and functions will be local to this module.

```
1 module("luaindex", package.seeall)
```

To handle all indexes we have a variable named `indexes`. This is a table of index tables *associated by the name of the index table*

```
indexes={
  name={
    presortreplaces={
      {[pattern]=replace, ...}, ...
    },
    sortorderbychar={
      [char]=position, ...
    },
    {
      sort="...",
      value="...",
      pages={...},
      subindex={...}
    }
  }
}
```

- Each index table has at least *two elements* associated to `presortreplaces` and `sortorderbychar`.
- There may be additional numerically associated elements, the *index entries*.
  - Each index entry has at least *two elements* associated to `sort` and `value`. Element `sort` is the sort key of the index entry. Element `value` is the print value of the index entry.
  - Each index entry may have an element associated to `pages`. This is a table of print values, that will be used as page number of the entry. It need not to be numeric. This table has numeric associations. Later added pages will be appended to the end of the table.
  - Each index entry may have an element associated to `subindex`. This is an index table too, but do not have elements `presortreplaces` or `sortorderbychar`.

```
2 local indexes = {}
```

`newindex(index name)`

Next we have a function to generate a new *index table* at `indexes`:

```
3 function newindex( indexname )
4   indexes[indexname]={ presortreplaces = {},
5                         sortorderbychar = {} }
6 end
```

The function parameter is the name of the index. This is not really a print name, but a simple association name.

Don't be impressed because of empty initialization of `presortreplaces` and `sortorderbychar`. We will have functions to change this.

First of all, we have a function to add a new sort order.

`sortorder(index name, sort-order)`

```
7 function sortorder( indexname, sortorder )
8   local i, value
```

The first parameter of the function is the name of the index table. If an index table with the given name does not exist,  $\TeX$  should release an error message with some optional help.

```
9   local index = indexes[indexname]
10  if index == nil then
```

```

11     tex.error( "Unknown index `" .. indexname .. "'",
12               { "You've tried to add a new sortorder to an index, but there's
13                 "given name.",
14                 "You should define the index using lua function ",
15                 "`luaindex.newindex(\"" .. indexname .. "\")'",
16                 "before."
17               }
18     )
19   else
20     if type(sortorder) == "string" then

```

The second parameter of the function may be a string. The string simply is an concatenation of the character in the order that should be used to sort the index entries of this index. The index table assoziation `sortorderbychar` is a table. The characters are the assoziation and the wanted sort position is the assoziated value.

```

21     local value
22     i = 1
23     repeat
24       value = unicode.utf8.sub( sortorder, i, i )
25 <debug>     print( i, value )
26     if value then
27       index.sortorderbychar[value] = i
28     end
29     i = i + 1
30     until value == ""
31   else -- should be table

```

The second parameter of the function may also be a table with numerical assoziations.

```

32     for i, value in ipairs( sortorder ) do
33       index.sortorderbychar[value] = i
34     end
35   end
36 end
37 end

```

`presortreplace(index name, pass, pattern, replace)` Second manipulation function is to add presort entries to a presort pass of an index. *pattern* and *replace* are strings. See Lua function `unicode.utf8.sub` for more information about these.

```

38 function presortreplace( indexname, pass, pattern, replace )
39   local n

```

The first parameter of the function is the name if the index table. If an index table with the given name does not exist,  $\TeX$  should release an error message with some optional help.

```

40   local index = indexes[indexname]
41   if index == nil then
42     tex.error( "Unknown index `" .. indexname .. "'",
43               { "You've tried to add a new presort-replace to an index, but tl

```

```

44             "with the given name.",
45             "You should define the index using lua function ",
46             " `luaindex.newindex(\" .. indexname .. "\")'",
47             "before."
48         }
49     )
50 else

```

If the index exists, we have to create replace tables for every pass until the given.

```

51     for n = table.maxn(index.presortreplaces), pass, 1 do
52         if ( index.presortreplaces[n] == nil ) then
53             index.presortreplaces[n] = {}
54         end
55     end

```

Last but not least we have to add a new replace to the pass:

```

56         index.presortreplaces[pass][pattern]=replace
57     end
58 end

```

`local getclass(  
 utf8-char)`

Indexes are normally separated into single letters, all numbers and all other symbols. To do so, we have a new function that returns 1 for all other symbols, 2 for all numbers and 3 for all letters. Whether an UTF-8 character is a letter or not depends on the locale type “collate”. You may set it using `os.setlocale("locale", "collate")`.

```

59 local function getclass( utfc )
60     local i
61     for i in unicode.utf8.gmatch( utfc, "%n" ) do
62 <debug>     print( utfc .. " is a number" )
63         return 2
64     end
65     for i in unicode.utf8.gmatch( utfc, "%a" ) do
66 <debug>     print( utfc .. " is a letter" )
67         return 3
68     end
69 <debug>     print( utfc .. " is a symbol" )
70     return 1
71 end

```

`local do_presortreplaces(  
 utf8-string,  
 replace table)`

Before printing or sorting we may want to replace some strings. We have a table of those. At the string each occurrence of the association should be replaced by the associated value.

```

72 local function do_presortreplaces( srcstr, presortreplace )
73     if presortreplace then
74         local pat, rep
75         for pat, rep in pairs( presortreplace ) do
76             srcstr = unicode.utf8.gsub( srcstr, pat, rep )
77         end
78     end

```

```

79   return srcstr
80 end

```

```

local printsubindex(
  level,
  index,
  presortreplace_zero)

```

Now let's print the index. There aren't much differences in printing an index or a sub-index to an index entry. We only need to know the level of the (sub-) index. level 0 is the main index.

```

81 local function printsubindex( level, index, presortreplace_zero )
82   local i,t,n,p,l
83   local group=""
84   local class=-1

```

We build the T<sub>E</sub>X index item command: `\item`, `\subitem`, `\subsubitem` etc. depending on the level. So `level` is simply the number of `sub` at the index item command.

```

85   local item=""
86   for l = 1, level, 1 do
87     item = item .. "sub"
88   end
89   item = item .. "item "

```

Walk through all index items.

```

90   for i,t in ipairs( index ) do

```

If `level` is 0, we are at the root index. We want to group this Index into numbers, symbols and single letters. To do so, we detect the class of the first character at the sort string and add `\indexgroup` commands if necessary.

```

91     if ( level == 0 ) then
92       local sort=do_presortreplaces( t["sort"], presortreplace_zero )
93       local firstchar=unicode.utf8.upper( unicode.utf8.sub( sort, 1, 1 ) )
94       if ( firstchar ~= group ) then
95         local newclass

```

The character differ, but we have to print the group only if the groups of the characters differ.

```

96         newclass=getclass( firstchar )
97         if ( newclass == 1 and class ~= newclass ) then
98           tex.print( "\\indexgroup{\\symbolsname}" )
99         elseif ( newclass == 3 ) then
100          tex.print( "\\indexgroup{" .. firstchar .. "}" )
101         elseif ( newclass == 2 and class ~= newclass ) then
102          tex.print( "\\indexgroup{\\numbersname}" )
103         end
104         group=firstchar
105         class=newclass
106       end
107     end

```

Now we have to print the index item. We use the `value` to be printed. If one or more pagenumbers are stored, we print them too. If the index entry has a sub index, we call `printsubindex` for this one with increased level.

```

108     tex.sprint( item, t["value"] )
109     if t["pages"] then
110         tex.sprint( "\\indexpagenumbers{" )
111         for n,p in ipairs( t["pages"] ) do
112             tex.sprint( "\\indexpagenumber{" .. p, "}" )
113         end
114         tex.print( "}" )
115     end
116     if t["subindex"] then
117         printsubindex( level+1, t["subindex"], presortreplaces_zero )
118     end
119 end
120 end

```

`printindex(index name)` Printing a whole index is simply the same like printing a sub index, but before printing the index, we have to test, whether the named index exists or not.

```

121 function printindex( indexname )
122     local index=indexes[indexname]
123     if index == nil then
124         tex.error( "Unknown index `" .. indexname .. "'",
125                 { "You've tried to print an index, but there's no index with the",
126                   "given name.",
127                   "You should define the index using lua function ",
128                   "`\luaindex.newindex(\"" .. indexname .. "\')'",
129                   "before."
130                 }
131             )
132     else
133         print( "Index: \"" .. indexname .. "\" with " .. table.maxn( index ) .. " entries" )
134         tex.print( "\\begin{theindex}" )
135         printsubindex(0,indexes[indexname],indexes[indexname].presortreplaces[0])
136         tex.print( "\\end{theindex}" )
137     end
138 end

```

`local getsubclass(utf8-char)`

To sort the index character classes numbers, letters and other are not enough. So we build sub-classes inside these three classes.

```

139 local function getsubclass( utfc )
140     local i
141     Inside letters we want so sort upper case before lower case.
142     for i in unicode.utf8.gmatch( utfc, "%l" ) do
143         return 1
144     end
145     for i in unicode.utf8.gmatch( utfc, "%u" ) do
146         return 2
147     end

```

Inside other symbols we want so sort controls before spaces before punctuations before numbers before unknown.

```

147   for i in unicode.utf8.gmatch( utfc, "%c" ) do
148       return 1
149   end
150   for i in unicode.utf8.gmatch( utfc, "%s" ) do
151       return 2
152   end
153   for i in unicode.utf8.gmatch( utfc, "%p" ) do
154       return 3
155   end
156   for i in unicode.utf8.gmatch( utfc, "%n" ) do
157       return 4
158   end
159   return 10 -- unkown is the biggest sub class
160 end

```

`local do_strcmp(`  
     *first string,*  
     *second string,*  
     *sort order table)*

To compare two UTF8-strings we could simply use the string compare of Lua. But for our purpose this is not enough. So we've added a configurable sort order and now have to compare character by character depending on this sort order.

```

161 local function do_strcmp( first, second, sortorderbychar )
162     local secondtable = string.explode( second, "" )
163     local firstutf
164     local n = 1
165     <debug> print( first .. ", " .. second );
166     for firstutf in string.utfcharacters( first ) do
167         local secondutf = unicode.utf8.sub( second, n, n )
168         n = n + 1;
169         if firstutf then
170             if secondutf ~= "" then
171                 <debug> print( " " .. firstutf .. ", " .. secondutf )
172                 if firstutf ~= secondutf then
173                     local firstn, secondn
174                     if sortorderbychar then
175                         firstn = sortorderbychar[firstutf]
176                         secondn = sortorderbychar[secondutf]
177                     end

```

If both characters were in the sort order table with different index we may return -1, if the index of first was lower than second, and 1, if the index of first was higher than second.

```

178             if firstn and secondn then
179                 <debug> print( " n: " .. firstn .. ", " .. secondn )
180                 if firstn < secondn then
181                     return -1
182                 elseif firstn > secondn then
183                     return 1
184                 end
185             else

```



If one character was not in the sort order table, we compare the classes and if same the sub-classes.

```
186         local firstclass = getclass( firstutf )
187         local secondclass = getclass( secondutf )
188         if firstclass < secondclass then
189             return -1
190         elseif firstclass == secondclass then
191             local firstsubclass = getsubclass( firstutf)
192             local secondsubclass = getsubclass( secondutf )
193             if firstsubclass < secondsubclass then
194                 return -1
195             elseif firstsubclass == secondsubclass then
196                 if firstutf < secondutf then
197                     return -1
198                 else
199                     return 1
200                 end
201             else
202                 return 1
203             end
204         else
205             return 1
206         end
207     end
208 end
209 else
```

If the first string was longer than the second, it is greater.

```
210         return 1
211     end
212 else
```

If the first string was shorter than the second, it is lower.

```
213     if secondutf ~= "" then
214         return -1
215     else
216         return 0 -- This should never happen!
217     end
218 end
219 end
```

If the first string was shorter than the second, it is lower. If not they are same.

```
220 if unicode.utf8.sub( second, n, n ) ~= "" then
221     return -1
222 else
223     return 0
224 end
225 end
```

```
local do_indexcmp(
    first string,
    second string,
    replace tables,
    sort order table)
```

Now we are able to compare the sort value of two index entries. Before

the first compare we do the first pre-sort replace. All other pre-sort replaces will be done only, if the sort entries are not same!

```

226 local function do_indexcmp( firstsort, secondsort,
227                             presortreplaces, sortorderbychar )
228     local pass = 0
229     local ncmp = 0
230     repeat
231         if presortreplaces and presortreplaces[pass] then
232             firstsort = do_presortreplaces( firstsort, presortreplaces[pass] )
233             secondsort = do_presortreplaces( secondsort, presortreplaces[pass] )
234 <debug>         print( "Replace-Pass " .. pass .. ": " .. firstsort .. ", " .. se
235             end
236             pass = pass + 1
237             ncmp = do_strcmp( firstsort, secondsort, sortorderbychar )
238         until ( ncmp ~= 0 ) or ( pass > table.maxn( presortreplaces ) )
239 <*debug>
240         if ncmp < 0 then
241             print( firstsort .. "<" .. secondsort )
242         elseif ncmp == 0 then
243             print ( firstsort .. "=" .. secondsort )
244         else
245             print( firstsort .. ">" .. secondsort )
246         end
247 </debug>
248     return ncmp
249 end

```

*local subinsert(*  
*index table,*  
*replace tables,*  
*sort order table,*  
*page string,*  
*sort value,*  
*print value,*  
*...)*

Inserting a new entry to an index is same like inserting a new entry to a sub-index of an already existing entry. So we have only one local function for this. A new entry consists of a page string, that should be added to the page list of the entry, a sort value, that should be used to find the correct entry and a print value, that should be shown at the index. Entries are only same, if the compare of the sort value is 0 and the print values are same. A new entry may be not only a new entry to the top level but to sub levels. Because of this, there may be several pairs of sort- and print values. We use bisection search to find the insert position.

```

250 local function subinsert( index, presortreplaces, sortorderbychar,
251                             pagestring, sortvalue, outputvalue, ... )
252     local min = 1
253     local max = table.maxn(index)
254     local updown = 0
255
256     local n = math.ceil(( min + max ) / 2)
257     while min <= max do
258         updown = do_indexcmp( sortvalue, index[n].sort,
259                             presortreplaces, sortorderbychar )
260         if updown == 0 then

```

The sort values are compared to be same (after serveral replaces). But only if the print values are (without any replaces) same, we have to use this entry. In this case we add a new sub-entry to this entry and if no new sub entry was given the page string to the page table.

```

261         if outputvalue == index[n].value then
262 <debug>             print( "The entries are same." )
263                 if ( ... ) then
264 <debug>                     print( " Adding subentry to already existing entry" )
265                         if ( index[n].subindex == nil ) then
266                             index[n].subindex = {}
267                         end
268                         subinsert( index[n].subindex, presortreplaces, sortorderbychar,
269                                 pagestring, ... )
270                     else
271 <debug>                         print( " Is the pagestring already at the pages table?" )
272                             local i, p
273                             for i, p in ipairs( index[n].pages ) do
274                                 if pagestring == p then
275 <debug>                                     print( "The pagestring is already at the pages table." )
276 <debug>                                     print( " We have nothing to do." )
277                                 return
278                             end
279 <debug>                                     print( pagestring, "!=", p )
280                         end
281 <debug>                             print( "The pagestring was not at the pages table.",
282 <debug>                                     "Add the new pagestring to the pages table",
283 <debug>                                     "and stop processing." )
284                             table.insert( index[n].pages, pagestring )
285                         end
286                         return
287                     else

```

If the print values are not same, we use sequential search for the position after the last entry with same sort value but different print value. This is the position to use for the new entry.

```

288 <debug>             print( "The entries are not same.",
289 <debug>                 "Search for the last entry, with same sort." )
290                 repeat
291                     n = n + 1
292                     if n <= max then
293                         updown = do_indexcmp( sortvalue, index[min].sort,
294                                             presortreplaces, sortorderbychar )
295                     end
296                     until n > max or updown ~= 0
297                     min = n
298                     max = n-1
299                 end
300                 elseif updown > 0 then
301                     min = n+1

```

```

302     else
303         max = n-1
304     end
305     n = math.ceil(( min + max ) / 2)
306 <debug>     print ( min, max, n )
307     end

    if we have a new sub entry we add this to the new position. If not we
    simply add the new entry with the page table.

308     if ( ... ) then
309 <debug>     print( "Generating new entry without page but subindex" )
310         table.insert( index, n,
311             { sort=sortvalue, value=outputvalue, subindex={ } } )
312 <debug>     print( "Add subindex to new generated entry" )
313         subinsert( index[n].subindex, presortreplaces, sortorderbychar,
314             pagestring, ... )
315     else
316 <debug>     print( "Generating new entry with page" )
317         table.insert( index, n,
318             { sort=sortvalue, value=outputvalue, pages={pagestring} } )
319     end
320 end

```

```

insert(index name,
page string,
sort value,
print value,
...)

```

We've explained before, that inserting a new entry is same like inserting a entry to a sub entry. There's only one tiny difference: the replace tables and sort order are members of the index table.

```

321 function insert( indexname, pagestring, sortvalue, outputvalue, ... )
322     local index=indexes[indexname]
323     subinsert( index, index.presortreplaces, index.sortorderbychar,
324         pagestring, sortvalue, outputvalue, ... )
325 end

```

`removeentries(index name)` Last we will need a function, that only removes all index entries but not presortreplaces or sortorderbychar.

```

326 function removeentries( indexname )
327     local p = indexes[indexname].presortreplaces
328     local s = indexes[indexname].sortorderbychar
329     indexes[indexname]={ presortreplaces = p,
330         sortorderbychar = s }
331 end

```

## 7 Implementation of $\text{\LaTeX}$ Package `luaindex.sty`

The  $\text{\LaTeX}$  package is user's candy but not necessary. You may use `luaindex.lua` directly, but  $\text{\LaTeX}$  users will expect a  $\text{\LaTeX}$  interface.

### 7.1 Package Startup

Lua $\text{\LaTeX}$  must be used to use the package.

```

332 \RequirePackage{ifluatex}
333 \ifluatex\else
334   \PackageError{luaindex}{lualatex needed}{%
335     Package `luaindex' needs LuaTeX.\MessageBreak
336     So you should use `lualatex' to process you document!\MessageBreak
337     See documentation of `luaindex' for further information.}%
338   \expandafter\expandafter\expandafter\csname endinput\endcsname
339 \fi

340 \RequirePackage{luatexbase-compat}[2010/10/10]

341 \RequirePackage{luatexbase-modutils}[2010/10/10]

    We need some LuaTEX primitives:
342 \luatexbase@ensure@primitive{luaescapestring}

    We need some Lua functions:
343 \directlua{%
344   if not tex.error then
345     luatexbase.module_error('luaindex',
346       'undefined function!\string\n%
347       LuaTeX function tex.error() needed but not defined.\string\n%
348       Maybe you are using the wrong version of LuaTeX.')
349   end
350   if not tex.print then
351     luatexbase.module_error('luaindex',
352       'undefined function!\string\n%
353       LuaTeX function tex.print() needed but not defined.\string\n%
354       Maybe you are using the wrong version of LuaTeX.')
355   end
356   if not tex.sprint then
357     luatexbase.module_error('luaindex',
358       'undefined function!\string\n%
359       LuaTeX function tex.sprint() needed but not defined.\string\n%
360       Maybe you are using the wrong version of LuaTeX.')
361   end
362 }

    Load an initialize lua module. We could do this much later, but it is
    very, very important, so we do is as soon as possible.
363 \RequireLuaModule{luaindex}

    With luaindex we use a temporary index file, too. This is necessary,
    because page numbers are only valid while output routine. So usage of a
    temporary index file is a good solution to have correct page numbers. If this
    file exists, we load it simply while \begin{document} and then produce an
    new one. But loading the old one is not simply an \input. Out temporary
    index file is a Lua file, so we use Lua function dofile to load it.
364 \newwrite\@indexfile
365 \AtBeginDocument{%

```

```

366 \IfFileExists{\jobname.ldx}{\directlua{dofile('\jobname.ldx')}}{}%
367 \openout\@indexfile=\jobname.ldx
368 }

```

## 7.2 Options

We use a key-value interface even for options. Because of this we're using KOMA-Script package `scrbase`.

```

369 \RequirePackage{scrbase}
370 \DefineFamily{luaindex}
371 \DefineFamilyMember{luaindex}

```

`sortorder` Support for individual sort order. Sort order is an attribute of the index  
`\luaindex@sortorder` root Lua table. Because of this the option simply saves it and it will be  
 setup later while defining new indexes.

```

372 \newcommand*{\luaindex@sortorder}{}
373 \DefineFamilyKey{luaindex}{sortorder}{%
374 \edef\luaindex@sortorder{#1}%
375 }

```

`locale` If no individual sort order is given, the *collate* locale would cause the sort  
 order. So we add an option make this locale changable. Note, that changing  
 this locale may also affect to other Lua functions!

```

376 \DefineFamilyKey{luaindex}{locale}{%
377 \if@atdocument
378 \expandafter\@firstofone
379 \else
380 \expandafter\AtBeginDocument
381 \fi
382 {%
383 \protected@write\@indexfile{}{%
384 os.setlocale('#1','collate')
385 }%
386 }%
387 }

```

`pageformat` The page format is an attribute of every index entry. But you may define a  
`\luaindex@pageformat` primary page format to be used, if no individual page format will be given.

```

388 \newcommand*{\luaindex@pageformat}{}
389 \DefineFamilyKey{luaindex}{pageformat}{%
390 \def\luaindex@pageformat{#1}%
391 }

```

`singlepass` This option changes the general behavior of `\printindex`. See definition  
 of `\printindex` for more information about.

```

392 \FamilyBoolKey{luaindex}{singlepass}{@luaindexsinglepass}

```

Processing all the options while loading the package.

```
393 \FamilyProcessOptions{luaindex}\relax
```

`\setupluaindex` This is only an convenience command for run time setup of `luaindex` options.

```
394 \newcommand*{\setupluaindex}{\FamilyOptions{luaindex}}
```

### 7.3 Some Usual Index Commands

`\see` `\see` and `\seealso` are common commands used at the page number format. They are defined for compatibility.

`\seename` `\seename` and `\alsoname` are used by `\see` and `\seealso` and needed to be defined also.

```
395 \newcommand*\see[2]{\emph{\seename} #1}
396 \providecommand\seealso[2]{\emph{\alsoname} #1}
397 \providecommand\seename{see}
398 \providecommand*\alsoname{see also}
```

### 7.4 Generation of Indexes and Index Entries

`\newindex` We can handle not only one index but several indexes. To do so, we have to create a new lua index table for each index. Just use

```
\newindex{<index name>}
```

to do so. Additional features may be set up using:

```
\newindex[<index options>]{<index name>}
```

Currently all global options are supported for `<index options>`, but some will be ignored.

```
399 \newcommand*\newindex[2][{}]{%
400   \directlua{luaindex.newindex('\luatexluaescapestring{#2}')}%
401   \begingroup
402     \setupluaindex{#1}%
403     \ifx\luaindex@sortorder\@empty\else
404       \AtBeginDocument{%
405         \protected@write\@indexfile{}{%
406           luaindex.sortorder('\luatexluaescapestring{#2}',
407                               '\luaindex@sortorder')
408         }%
409       \fi
410     \endgroup
411 }
```

You may use `\newindex` at the document preamble only.

```
412 \@onlypreamble\newindex
```

`\luaindex` This command will be used to add a new root level entry to an index:

`\luaindex{⟨index name⟩}[⟨options⟩]{⟨entry⟩}`

`⟨index name⟩` – the name of the index to be used. This has to be the same like you’ve used to create the new index using `\newindex`.

`⟨options⟩` – several options for the index entry. Currently supported are:

`locale=⟨locale specifier⟩` – just calls `\luaindexsetup{⟨locale specifier⟩}`.

Note, that this is a global action!

`pageformat=⟨command⟩` – is a command with at most one argument to format the page number of the index entry. You may, e. g., use `sort=\see{⟨reference⟩}` or `sort=\seealso{⟨reference⟩}` to produce a “see” or “see also” cross reference to `⟨reference⟩` instead of showing a real page number.

`sort=⟨sort entry⟩` – destines the sort position of the index entry. If it is omitted `⟨entry⟩` will be used instead.

`⟨entry⟩` – this will be shown in the index.

Note: An index entry is only same, if `⟨sort entry⟩` is same (after several presort replaces) and `⟨entry⟩` is same. Index entries with same `⟨sort entry⟩` but different `⟨entry⟩` will be placed at the current end of the entries with same `⟨sort entry⟩`.

```
413 \newcommand*\luaindex}[1]{%
414   \@bsphack
415   \begingroup
416     \edef\luaindex@name{#1}%
417     \lua@index
418 }
419 \newcommand*\lua@index}[2][ ]{%
420   \set@display@protect
421   \edef\luaindex@sort{#2}%
422   \define@key{luaindex.setindex}{sort}{\edef\luaindex@sort{##1}}%
423   \define@key{luaindex.setindex}{pageformat}{\def\luaindex@pageformat{##1}}%
424   \define@key{luaindex.setindex}{locale}{\luaindexsetup{locale=#1}}%
425   \setkeys{luaindex.setindex}{#1}%
426   \protected@write\@indexfile{\let\luatexluaescapestring\relax}{%
427     luaindex.insert('\luatexluaescapestring{\luaindex@name}',
428                    '\luatexluaescapestring{\luaindex@pageformat{\thepage}},
429                    '\luatexluaescapestring{\luaindex@sort}',
430                    '\luatexluaescapestring{#2}')
431   }%
432 \endgroup
433 \@esphack
434 }
```



`\luasubindex` Same like `\luaindex` but to produce a sub entry:  
`\lua@subindex`  
`\lua@@subindex` `\luasubindex{⟨index name⟩}[⟨options⟩]{⟨entry⟩}[⟨options⟩]{⟨sub-entry⟩}`

Note, that the `⟨options⟩` for the `⟨sub-entry⟩` only allows a sub-set of the options shown for `\luaindex`. Currently only `sort=⟨sort entry⟩`.

```

435 \newcommand*\luasubindex}[1]{%
436   \@bsphack
437   \begingroup
438     \edef\luaindex@name{#1}%
439     \lua@subindex
440 }
441 \newcommand*\lua@subindex}[2] []{%
442   \set@display@protect
443   \edef\luaindex@sort{#2}%
444   \define@key{luaindex.setindex}{sort}{\edef\luaindex@sort{##1}}%
445   \define@key{luaindex.setindex}{pageformat}{\def\luaindex@pageformat{##1}}%
446   \define@key{luaindex.setindex}{locale}{\luaindexsetup{locale=#1}}%
447   \setkeys{luaindex.setindex}{#1}%
448   \protected@write\@indexfile{\let\luatexluaescapestring\relax}{%
449     luaindex.insert('\luatexluaescapestring\luaindex@name',
450                    '\luatexluaescapestring{\luaindex@pageformat{\thepage}},
451                    '\luatexluaescapestring{\luaindex@sort}',
452                    '\luatexluaescapestring{#2}',
453                    }%
454   \aftergroup\lua@@subindex
455 \endgroup
456 }
457 \newcommand*\lua@@subindex}[2] []{%
458   \begingroup
459   \set@display@protect
460   \edef\luaindex@sort{#2}%
461   \define@key{luaindex.setindex}{sort}{\edef\luaindex@sort{##1}}%
462   \setkeys{luaindex.setindex}{#1}%
463   \protected@write\@indexfile{\let\luatexluaescapestring\relax}{%
464     \@spaces
465     '\luatexluaescapestring{\luaindex@sort}',
466     '\luatexluaescapestring{#2}')
467   }%
468 \endgroup
469 \@esphack
470 }

```

`\luasubsubindex` Same like `\luaindex` but to produce a sub-sub-entry, that is a sub-entry  
`\lua@subsubindex` to a sub-entry:  
`\lua@@@subindex` `\luasubindex{⟨index name⟩}[⟨options⟩]{⟨entry⟩}[⟨options⟩]{⟨sub-entry⟩}`  
`[⟨options⟩]{⟨sub-sub-entry⟩}`

Note, that the *options* for the *sub-entry* and the *sub-sub-entry* only allows a sub-set of the options shown for `\luaindex`. Currently only `sort=<sort entry>`.

```

471 \newcommand*\luasubsubindex}[1]{%
472   \@bsphack
473   \begingroup
474     \edef\luaindex@name{#1}%
475     \lua@subsubindex
476 }
477 \newcommand*\lua@subsubindex}[2][{}]{%
478   \set@display@protect
479   \edef\luaindex@sort{#2}%
480   \define@key{luaindex.setindex}{sort}{\edef\luaindex@sort{##1}}%
481   \define@key{luaindex.setindex}{pageformat}{\def\luaindex@pageformat{##1}}%
482   \define@key{luaindex.setindex}{locale}{%
483     \luaindexsetup{locale=#1}%
484   }
485   \setkeys{luaindex.setindex}{#1}%
486   \protected@write\@indexfile{\let\luatexluaescapestring\relax}{%
487     luaindex.insert('\luatexluaescapestring{\luaindex@name}',
488                    '\luatexluaescapestring{\luaindex@pageformat{\thepage}},
489                    '\luatexluaescapestring{\luaindex@sort}',
490                    '\luatexluaescapestring{#2}'),
491   }%
492   \aftergroup\lua@@@subindex
493 \endgroup
494 }
495 \newcommand*\lua@@@subindex}[2][{}]{%
496   \begingroup
497     \set@display@protect
498     \edef\luaindex@sort{#2}%
499     \define@key{luaindex.setindex}{sort}{\edef\luaindex@sort{##1}}%
500     \setkeys{luaindex.setindex}{#1}%
501     \protected@write\@indexfile{\let\luatexluaescapestring\relax}{%
502       \@spaces
503       '\luatexluaescapestring{\luaindex@sort}',
504       '\luatexluaescapestring{#2}'),
505   }%
506   \aftergroup\lua@@subindex
507 \endgroup
508 }

```

<code>\makeindex</code>	These are defined to increase compatibility to old index packages only.
<code>\index</code>	Command <code>\makeindex</code> simply generates the new index named <code>general</code>
<code>\subindex</code>	and the other commands to add entries to that index. Note, that adding
<code>\subsubindex</code>	a sub-entry or sub-sub-entry is not yet compatible to other index packages. You need to use the command <code>\subindex</code> and <code>\subsubindex</code> instead of something like <code>\index{&lt;entry&gt;!&lt;sub-entry&gt;!&lt;sub-sub-entry&gt;}</code> . Note also,

Table 1: Implications of option `singlepass` to `\printindex`

<code>singlepass=false</code>	<code>singlepass=true</code>
index of previous Lua <sub>A</sub> T <sub>E</sub> X run will be printed	index of current Lua <sub>A</sub> T <sub>E</sub> X run will be printed
start of index depends on the class	start of the index at next page earliest
index entries may be added to an index even after it has been printed	no more index entries may be added to the index after it has been printed

that changing the format of the page number is not compatible with other index packages. You have to use `\index[pageformat=<page format>]{...}` instead of something like `\index{<entry>|<page format>}`.

```

509 \renewcommand*\makeindex}{%
510   \newindex{general}%
511   \renewcommand*\index{\luaindex{general}}%
512   \newcommand*\subindex{\luasubindex{general}}%
513   \newcommand*\subsubindex{\luasubsubindex{general}}%
514 }
```

## 7.5 Printing an Index

We do not only want to create an index, we also need to print it.

`\printindex` With

```
\printindex[<options>]
```

you can print an index. The known options are

`index=<index name>` – print the index with the given name as declared at `\newindex`. If you omit this option, index “general” will be printed.

`singlepass=<boolean value>` – you may switch on and of the single pass feature. For the differences of single pass feature on and off, see table 1

```

515 \newcommand*\printindex}[1] []{%
516   \begingroup
517     \edef\luaindex@name{general}%
518     \define@key{luaindex.setindex}{index}{\edef\luaindex@name{##1}}%
519     \define@key{luaindex.setindex}{singlepass}[true]{%
520       \setupluaindex{singlepass}{##1}%
521     }%
522     \setkeys{luaindex.setindex}{#1}%

```

```

523 \if@luaindexsinglepass
524 \closeout\@indexfile
525 \clearpage
526 \directlua{%
527     luaindex.removeentries('\luatexluaescapestring{\luaindex@name}')
528     dofile('\jobname.ldx')
529 }%
530 \fi
531 \directlua{%
532     luaindex.printindex('\luatexluaescapestring{\luaindex@name}')
533 }%
534 \endgroup
535 }

```

`luaindex.lua` uses several macros while printing the index. First of all it uses the environment `theindex`. But several additional macros will be used:

<code>\indexgroup</code>	Each index is grouped. Index groups are symbols, numbers and each first
<code>\indexspace</code>	letter. Each group starts with <code>\indexgroup{&lt;group&gt;}</code> with group is ei-
<code>\symbolsname</code>	ther <code>\symbolsname</code> , <code>\numbersname</code> or a upper case letter. In difference
<code>\numbersname</code>	to other index processors no automatic <code>\indexspace</code> will be added before
	each group. So we define <code>\indexgroup</code> to add it.

```

536 \providecommand*\indexgroup}[1]{%
537 \indexspace\textbf{#1}\nopagebreak
538 }
539 \providecommand*\indexspace}{%
540 \def\indexspace{\vskip\baselineskip}
541 }
542 \providecommand*\symbolsname{Symbols}
543 \providecommand*\numbersname{Numbers}
544 \AtBeginDocument{%
545 \providecaptionname{english}\symbolsname{Symbols}%
546 \providecaptionname{english}\numbersname{Numbers}%
547 \providecaptionname{german}\symbolsname{Symbole}%
548 \providecaptionname{german}\numbersname{Zahlen}%
549 \providecaptionname{ngerman}\symbolsname{Symbole}%
550 \providecaptionname{ngerman}\numbersname{Zahlen}%
551 \providecaptionname{austrian}\symbolsname{Symbole}%
552 \providecaptionname{austrian}\numbersname{Zahlen}%
553 \providecaptionname{naustrian}\symbolsname{Symbole}%
554 \providecaptionname{naustrian}\numbersname{Zahlen}%
555 \providecaptionname{french}\symbolsname{Symbole}%
556 \providecaptionname{french}\numbersname{Chiffres}%
557 \providecaptionname{spanish}\symbolsname{Simbolos}%
558 \providecaptionname{spanish}\numbersname{N'umeros}%
559 }

```

<code>\indexpagenumbers</code>	The page numbers of an entry are printed all together as argument
<code>\indexpagenumber</code>	
<code>\indexpagenumbersep</code>	
<code>\index@pagenumbersep</code>	

of `\indexpagenumbers{⟨page number⟩}`. Each single page number is printed as argument of `\indexpagenumber{⟨page number⟩}`. So separate the single page numbers `\indexpagenumber` is predefined to add internal macro `\index@pagenumbersep` before the page number. This will add `\indexpagenumbersep` before each page number but the first one.

```
560 \providecommand*\indexpagenumbers}[1]{%
561   \def\index@pagenumbersep{\let\index@pagenumbersep\indexpagenumbersep}%
562   \nobreakspace-- #1}
563 \providecommand*\indexpagenumber}[1]{\index@pagenumbersep #1}
564 \providecommand*\indexpagenumbersep}{, }
```

## 8 Examples

Currently only one example file will be produced:

`luaindex-example` – This should show index entries, index sub-entries, index sub-sub-entries.

```
565   \documentclass{article}
566   \usepackage[ngerman]{babel}
567   \usepackage{blindtext}
568   \usepackage{fontspec}
```

We load package `luaindex` with option `locale=de_DE`. At least at Linux this will add Ä, Ö, Ü, ä, ö, ü, and ß to the letters and even set a valid sort order for those.

We load package `luaindex` with option `singlepass` to produce a valid index with one Lua<sub>La</sub>T<sub>E</sub>X run instead of two or more. But with this printing of the index will produce a new page.

```
569   \usepackage[
570     locale=de_DE,
571     singlepass % Wenn der Index ohnehin eine neue Seite produziert,
572                % dann kann er direkt beim ersten Lauf ein korrektes
573                % Ergebnis liefern.
574   ]{luaindex}
```

We use the compatibility command `\makeindex` to generate the “general” index and the further compatibility commands, e.g., `\index`.

```
575   \makeindex
```

We want `\textbf` to be ignored at the sort:

```
576   \directlua{luaindex.presortreplace('general',0,
577     '\luatexluaescapestring{\string\textbf}\space*\string\{([\string^\string\
```

Now we can start our document. This consist of some text and several index entries.

```

578     \begin{document}
579
580     \blindtext[10]
581     A\index{B ist der zweite Buchstabe}
582     aber\index{aber ist ein Wort}
583     D\index{D ist der vierte Buchstabe}
584     A\index{A ist der erste Buchstabe}
585     A\index{A ist der erste Buchstabe}

```

Now, let's do something different. Let's show that babel shorthands may be used inside index entries:

```

586     C\index{C ist "`der" dritte Buchstabe}
587     X\index{X ist der drittletzte Buchstabe}

```

And macros may also be used but change the sort sequence of the index!

```

588     D\index{\textbf{D} ist der Buchstabe nach C}
589     Y\index{Y ist der \textbf{vorletzte} Buchstabe}
590     Z\index{Z ist der letzte Buchstabe}
591     A\index{Ä ist auch ein Buchstabe}

```

We may change the sort sequence manually by adding the `sort` option. The page number format may also be changed using the `pageformat` option.

```

592     Ä\index[sort={Ä ist aber auch ein Buchstabe},%
593           pageformat=\emph]{Ä ist wirklich auch
594           ein Buchstabe (und hier stimmt die Sortierung
595           nicht -- \emph{aber eigentlich doch})}

```

Let's add one more page with some more index entries:

```

596     \clearpage
597
598     A\index{A ist der erste Buchstabe}
599     Ae\index{Ae ist kein Buchstabe, sondern zwei}
600

```

And now, let's have some sub-entries and even a sub-sub-entry. One of the sub-entries will become a different sort position and will be marked with an emphasized page number.

```

601     Kompliziert\subindex{Diverses}{Untereintrag}
602     Noch komplizierter\subindex{Diverses}{Obereintrag}
603     Noch komplizierter\%
604     subindex{Diverses}[sort=Obereintra,pageformat=\emph]{Untereintrag}
605     Noch komplizierter\%
606     \subsubindex{Diverses}{Untereintrag}{Unteruntereintrag}
607

```

That's enough. Time time to print the index. Remember, that this is already a valid index, because we are using option `singlepass`.

```
608     \printindex
609     \end{document}
```

## Index

Numbers written in *italics* refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in *roman* refer to the code lines where the entry is used.

<b>A</b>		<b>N</b>	
<code>\alsoname</code> .....	<u>395</u>	<code>\newindex</code> .....	<u>399</u>
<b>I</b>		<code>\numbersname</code> .....	<u>536</u>
<code>\index</code> .....	<u>509</u>	<b>O</b>	
<code>\index@pagenumbersep</code> .....	<u>560</u>	Optionen:	
<code>\indexgroup</code> .....	<u>536</u>	<code>locale</code> .....	<u>376</u>
<code>\indexpagenumber</code> .....	<u>560</u>	<code>pageformat</code> .....	<u>388</u>
<code>\indexpagenumbers</code> .....	<u>560</u>	<code>singlepass</code> .....	<u>392</u>
<code>\indexpagenumbersep</code> .....	<u>560</u>	<code>sortorder</code> .....	<u>372</u>
<code>\indexspace</code> .....	<u>536</u>	<b>P</b>	
<b>L</b>		<code>pageformat (Option)</code> .....	<u>388</u>
<code>locale (Option)</code> .....	<u>376</u>	<code>\printindex</code> .....	<u>515</u>
<code>\lua@@@subindex</code> .....	<u>471</u>	<b>S</b>	
<code>\lua@subindex</code> .....	<u>435</u>	<code>\see</code> .....	<u>395</u>
<code>\lua@subindex</code> .....	<u>435</u>	<code>\seealso</code> .....	<u>395</u>
<code>\lua@subsubindex</code> .....	<u>471</u>	<code>\seename</code> .....	<u>395</u>
<code>\luaindex</code> .....	<u>413</u>	<code>\setupluaindex</code> .....	<u>394</u>
<code>\luaindex@pageformat</code> .....	<u>388</u>	<code>singlepass (Option)</code> .....	<u>392</u>
<code>\luaindex@sortorder</code> .....	<u>372</u>	<code>sortorder (Option)</code> .....	<u>372</u>
<code>\luasubindex</code> .....	<u>435</u>	<code>\subindex</code> .....	<u>509</u>
<code>\luasubsubindex</code> .....	<u>471</u>	<code>\subsubindex</code> .....	<u>509</u>
<b>M</b>		<code>\symbolsname</code> .....	<u>536</u>
<code>\makeindex</code> .....	<u>509</u>		

## Change History

v0.1		Using package <code>luatexbase-compat</code>	
General: start of new package ..	1	.....	13
v0.1b		Using package <code>luatexbase-modutils</code>	
General: prefix ‘koma.’ removed		.....	13
from Lua module .....	1		