Usage of LuaTEX module `luaindex` and LuaLATEX Package `luaindex` for Generating Indexes

Markus Kohm[*]

v0.1b

With LuaTEX it would not be a problem to call an index processor like MakeIndex while running LuaTEX. 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

1 Idea **2**

2 General Options **2**

3 Generating Index Entries **2**

4 Print an Index **2**

5 Known Issues **2**

6 Implementation of Lua Module `luaindex.lua` **3**

---

[*] komascript@gmx.info

7 Implementation of LaTeX Package `luaindex.sty` 12
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

8 Examples 21

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 my be designed and implemented in a better kind. The author himself is just learning LuaTeX.

Nevertheless you may report bugs and patches to komascript@gmx.info.

6 Implementation of Lua Module `luaindex.lua`

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

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

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

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

- Each index table has at least *two elements* assoziated to `presortreplaces` and `sortorderbychar`.
- There may be additional numericly assoziated elements, the *index entries*.
  - Each index entry has a least *two elements* assoziated to `sort` und `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 assoziated 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 hat numeric assoziations. Later addeed pages will be appended to the end of the table.
  - Each index entry may habe an element assoziated to `subindex`. This is an index table too, but do not have elements `presortreplaces` or `sortorderbychar`.

```
local indexes = {}
```

`newindex(`*`index name`*`)`

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

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

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

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

`sortorder(`*`index name`*`, `*`sort-order`*`)`

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

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

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.

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

```
      tex.error( "Unknown index `" .. indexname .. "'",
                 { "You've tried to add a new sortorder to an index, but there's
                   "given name.",
                   "You should define the index using lua function ",
                   "  `luaindex.newindex(\"" .. indexname .. "\")'",
                   "before."
                 }
               )
   else
      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 assoziatione `sortorderbychar` is a table. The characters are the assoziation and the wanted sort position is the assoziated value.

```
         local value
         i = 1
         repeat
            value = unicode.utf8.sub( sortorder, i, i )
⟨debug⟩               print( i, value )
            if value then
               index.sortorderbychar[value] = i
            end
            i = i + 1
         until value == ""
      else -- should be table
```

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

```
         for i, value in ipairs( sortorder ) do
            index.sortorderbychar[value] = i
         end
     end
   end
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.

```
function presortreplace( indexname, pass, pattern, replace )
   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.

```
   local index = indexes[indexname]
   if index == nil then
      tex.error( "Unknown index `" .. indexname .. "'",
                 { "You've tried to add a new presort-replace to an index, but t
```

```
                   "with the given name.",
                   "You should define the index using lua function ",
                   "  `luaindex.newindex(\"" .. indexname .. "\")'",
                   "before."
                 }
               )
   else
```

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

```
      for n = table.maxn(index.presortreplaces), pass, 1 do
         if ( index.presortreplaces[n] == nil ) then
            index.presortreplaces[n] = {}
         end
      end
```

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

```
      index.presortreplaces[pass][pattern]=replace
   end
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. Wether an UTF-8 character is a letter or not depends on the locale type "`collate`". You may set it using `os.setlocale("`*`locale`*`", "collate")`.

```
local function getclass( utfc )
   local i
   for i in unicode.utf8.gmatch( utfc, "%n" ) do
⟨debug⟩  print( utfc .. " is a number" )
      return 2
   end
   for i in unicode.utf8.gmatch( utfc, "%a" ) do
⟨debug⟩  print( utfc .. " is a letter" )
      return 3
   end
⟨debug⟩  print( utfc .. " is a symbol" )
   return 1
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 occurence of the assoziation should be replaced by the assoziated value.

```
local function do_presortreplaces( srcstr, presortreplace )
   if presortreplace then
      local pat, rep
      for pat, rep in pairs( presortreplace ) do
         srcstr = unicode.utf8.gsub( srcstr, pat, rep )
      end
   end
   return srcstr
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.

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

We build the TeX index item command: `\item`, `\subitem`, `\subsubitem` etc. depending on the level. So `level` is simply the number of `sub` at the index item command.

```
   local item="\\"
   for l = 1, level, 1 do
      item = item .. "sub"
   end
   item = item .. "item "
```

Walk through all index items.

```
   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 neccessary.

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

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

```
            newclass=getclass( firstchar )
            if ( newclass == 1 and class ~= newclass ) then
               tex.print( "\\indexgroup{\\symbolsname}" )
            elseif ( newclass == 3 ) then
               tex.print( "\\indexgroup{" .. firstchar .. "}" )
            elseif ( newclass == 2 and class ~= newclass ) then
               tex.print( "\\indexgroup{\\numbersname}" )
            end
            group=firstchar
            class=newclass
         end
      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.

```
      tex.sprint( item, t["value"] )
      if t["pages"] then
         tex.sprint( "\\indexpagenumbers{" )
         for n,p in ipairs( t["pages"] ) do
            tex.sprint( "\\indexpagenumber{", p, "}" )
         end
         tex.print( "}" )
      end
      if t["subindex"] then
         printsubindex( level+1, t["subindex"], presortreplaces_zero )
      end
   end
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, wether the named index exists or not.

```
function printindex( indexname )
   local index=indexes[indexname]
   if index == nil then
      tex.error( "Unknown index `" .. indexname .. "'",
                 { "You've tried to print an index, but there's no index with th
                   "given name.",
                   "You should define the index using lua function ",
                   "  `luaindex.newindex(\"" .. indexname .. "\")'",
                   "before."
                 }
               )
   else
      print( "Index: \"" .. indexname .. "\" with " .. table.maxn( index ) .. "
      tex.print( "\\begin{theindex}" )
      printsubindex(0,indexes[indexname],indexes[indexname].presortreplaces[0])
      tex.print( "\\end{theindex}" )
   end
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.

```
local function getsubclass( utfc )
   local i
```

Inside letters we want so sort upper case before lower case.

```
   for i in unicode.utf8.gmatch( utfc, "%l" ) do
      return 1
   end
   for i in unicode.utf8.gmatch( utfc, "%u" ) do
      return 2
   end
```

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

```
   for i in unicode.utf8.gmatch( utfc, "%c" ) do
      return 1
   end
   for i in unicode.utf8.gmatch( utfc, "%s" ) do
      return 2
   end
   for i in unicode.utf8.gmatch( utfc, "%p" ) do
      return 3
   end
   for i in unicode.utf8.gmatch( utfc, "%n" ) do
      return 4
   end
   return 10 -- unkown is the biggest sub class
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 depeding on this sort order.

```
local function do_strcmp( first, second, sortorderbychar )
   local secondtable = string.explode( second, "" )
   local firstutf
   local n = 1
⟨debug⟩  print( first .. ", " .. second );
   for firstutf in string.utfcharacters( first ) do
      local secondutf = unicode.utf8.sub( second, n, n )
      n = n + 1;
      if firstutf then
         if secondutf ~= "" then
⟨debug⟩           print( " " .. firstutf .. ", " .. secondutf )
            if firstutf ~= secondutf then
               local firstn, secondn
               if sortorderbychar then
                  firstn = sortorderbychar[firstutf]
                  secondn = sortorderbychar[secondutf]
               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.

```
               if firstn and secondn then
⟨debug⟩                 print( " n: " .. firstn .. ", " .. secondn )
                  if firstn < secondn then
                     return -1
                  elseif firstn > secondn then
                     return 1
                  end
               else
```

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

```
                  local firstclass = getclass( firstutf )
                  local secondclass = getclass( secondutf )
                  if firstclass < secondclass then
                     return -1
                  elseif firstclass == secondclass then
                     local firstsubclass = getsubclass( firstutf)
                     local secondsubclass = getsubclass( secondutf )
                     if firstsubclass < secondsubclass then
                        return -1
                     elseif firstsubclass == secondsubclass then
                        if firstutf < secondutf then
                           return -1
                        else
                           return 1
                        end
                     else
                        return 1
                     end
                  else
                     return 1
                  end
               end
            end
         else
```

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

```
            return 1
         end
      else
```

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

```
         if secondutf ~= "" then
            return -1
         else
            return 0 -- This should never happen!
         end
      end
   end
```

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

```
   if unicode.utf8.sub( second, n, n ) ~= "" then
      return -1
   else
      return 0
   end
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!

```
local function do_indexcmp( firstsort, secondsort,
                            presortreplaces, sortorderbychar )
   local pass = 0
   local ncmp = 0
   repeat
      if presortreplaces and presortreplaces[pass] then
         firstsort = do_presortreplaces( firstsort, presortreplaces[pass] )
         secondsort = do_presortreplaces( secondsort, presortreplaces[pass] )
⟨debug⟩        print( "Replace-Pass " .. pass .. ": " .. firstsort .. ", " .. se
      end
      pass = pass + 1
      ncmp = do_strcmp( firstsort, secondsort, sortorderbychar )
   until ( ncmp ~= 0 ) or ( pass > table.maxn( presortreplaces ) )
⟨*debug⟩
   if ncmp < 0 then
      print( firstsort .. "<" .. secondsort )
   elseif ncmp == 0 then
      print ( firstsort .. "=" .. secondsort )
   else
      print( firstsort .. ">" .. secondsort )
   end
⟨/debug⟩
   return ncmp
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.

```
local function subinsert( index, presortreplaces, sortorderbychar,
                          pagestring, sortvalue, outputvalue, ... )
   local min = 1
   local max = table.maxn(index)
   local updown = 0

   local n = math.ceil(( min + max ) / 2)
   while min <= max do
      updown = do_indexcmp( sortvalue, index[n].sort,
                            presortreplaces, sortorderbychar )
      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.

```
         if outputvalue == index[n].value then
⟨debug⟩           print( "The entries are same." )
            if ( ... ) then
⟨debug⟩              print( " Adding subentry to already existing entry" )
               if ( index[n].subindex == nil ) then
                  index[n].subindex = {}
               end
               subinsert( index[n].subindex, presortreplaces, sortorderbychar,
                          pagestring, ... )
            else
⟨debug⟩              print( " Is the pagestring already at the pages table?" )
               local i, p
               for i, p in ipairs( index[n].pages ) do
                  if pagestring == p then
⟨debug⟩                    print( "The pagestring is already at the pages table.
⟨debug⟩                    print( " We have nothing to do." )
                     return
                  end
⟨debug⟩                 print( pagestring, "!=", p )
               end
⟨debug⟩              print( "The pagestring was not at the pages table.",
⟨debug⟩                     "Add the new pagestring to the pages table",
⟨debug⟩                     "and stop processing." )
               table.insert( index[n].pages, pagestring )
            end
            return
         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.

```
⟨debug⟩           print( "The entries are not same.",
⟨debug⟩                  "Search for the last entry, with same sort." )
            repeat
               n = n + 1
               if n <= max then
                  updown = do_indexcmp( sortvalue, index[min].sort,
                                        presortreplaces, sortorderbychar )
               end
            until n > max or updown ~= 0
            min = n
            max = n-1
         end
      elseif updown > 0 then
         min = n+1
      else
         max = n-1
      end
      n = math.ceil(( min + max ) / 2)
⟨debug⟩     print ( min, max, n )
   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.

```
   if ( ... ) then
⟨debug⟩     print( "Generating new entry without page but subindex" )
      table.insert( index, n,
                    { sort=sortvalue, value=outputvalue, subindex={} } )
⟨debug⟩     print( "Add subindex to new generated entry" )
      subinsert( index[n].subindex, presortreplaces, sortorderbychar,
                 pagestring, ... )
   else
⟨debug⟩     print( "Generating new entry with page" )
      table.insert( index, n,
                    { sort=sortvalue, value=outputvalue, pages={pagestring} } )
   end
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.

```
function insert( indexname, pagestring, sortvalue, outputvalue, ... )
   local index=indexes[indexname]
   subinsert( index, index.presortreplaces, index.sortorderbychar,
              pagestring, sortvalue, outputvalue, ... )
end
```

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

```
function removeentries( indexname )
   local p = indexes[indexname].presortreplaces
   local s = indexes[indexname].sortorderbychar
   indexes[indexname]={ presortreplaces = p,
                        sortorderbychar = s }
end
```

7 Implementation of LaTeX Package `luaindex.sty`

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

7.1 Package Startup

LuaLaTeX must be used to use the package.

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

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

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

We need some LuaTeX primitives:

```
\luatexbase@ensure@primitive{luaescapestring}
```

We need some Lua functions:

```
\directlua{%
   if not tex.error then
      luatexbase.module_error('luaindex',
         'undefined function!\string\n%
          LuaTeX function tex.error() needed but not defined.\string\n%
          Maybe you are using the wrong version of LuaTeX.')
   end
   if not tex.print then
      luatexbase.module_error('luaindex',
         'undefined function!\string\n%
          LuaTeX function tex.print() needed but not defined.\string\n%
          Maybe you are using the wrong version of LuaTeX.')
   end
   if not tex.sprint then
      luatexbase.module_error('luaindex',
         'undefined function!\string\n%
          LuaTeX function tex.sprint() needed but not defined.\string\n%
          Maybe you are using the wrong version of LuaTeX.')
   end
}
```

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.

```
\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.

```
\newwrite\@indexfile
\AtBeginDocument{%
```

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

7.2 Options

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

```
\RequirePackage{scrbase}
\DefineFamily{luaindex}
\DefineFamilyMember{luaindex}
```

`sortorder`
`\luaindex@sortorder`

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

```
\newcommand*{\luaindex@sortorder}{}
\DefineFamilyKey{luaindex}{sortorder}{%
  \edef\luaindex@sortorder{#1}%
}
```

`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!

```
\DefineFamilyKey{luaindex}{locale}{%
  \if@atdocument
    \expandafter\@firstofone
  \else
    \expandafter\AtBeginDocument
  \fi
  {%
    \protected@write\@indexfile{}{%
      os.setlocale('#1','collate')
    }%
  }%
}
```

`pageformat`
`\luaindex@pageformat`

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

```
\newcommand*{\luaindex@pageformat}{}
\DefineFamilyKey{luaindex}{pageformat}{%
  \def\luaindex@pageformat{#1}%
}
```

`singlepass`

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

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

Processing all the options while loading the package.

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

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

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

7.3 Some Usual Index Commands

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

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

```
\newcommand*\see[2]{\emph{\seename} #1}
\providecommand*\seealso[2]{\emph{\alsoname} #1}
\providecommand\seename{see}
\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.

```
\newcommand*{\newindex}[2][]{%
  \directlua{luaindex.newindex('\luatexluaescapestring{#2}')}%
  \begingroup
    \setupluaindex{#1}%
    \ifx\luaindex@sortorder\@empty\else
      \AtBeginDocument{%
        \protected@write\@indexfile{}{%
          luaindex.sortorder('\luatexluaescapestring{#2}',
                             '\luaindex@sortorder')
      }}%
    \fi
  \endgroup
}
```

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

```
\@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*⟩.

```
\newcommand*{\luaindex}[1]{%
  \@bsphack
  \begingroup
    \edef\luaindex@name{#1}%
    \lua@index
}
\newcommand*{\lua@index}[2][]{%
    \set@display@protect
    \edef\luaindex@sort{#2}%
    \define@key{luaindex.setindex}{sort}{\edef\luaindex@sort{##1}}%
    \define@key{luaindex.setindex}{pageformat}{\def\luaindex@pageformat{##1}}%
    \define@key{luaindex.setindex}{locale}{\luaindexsetup{locale=#1}}%
    \setkeys{luaindex.setindex}{#1}%
    \protected@write\@indexfile{\let\luatexluaescapestring\relax}{%
        luaindex.insert('\luatexluaescapestring{\luaindex@name}',
                             '{\luatexluaescapestring{\luaindex@pageformat{\thepa
                             '\luatexluaescapestring{\luaindex@sort}',
                             '\luatexluaescapestring{#2}')
    }%
  \endgroup
  \@esphack
}
```

`\luasubindex`
`\lua@subindex`
`\lua@@subindex`

Same like `\luaindex` but to produce a sub entry:

`\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*⟩.

```
\newcommand*{\luasubindex}[1]{%
  \@bsphack
  \begingroup
    \edef\luaindex@name{#1}%
    \lua@subindex
}
\newcommand*{\lua@subindex}[2][]{%
    \set@display@protect
    \edef\luaindex@sort{#2}%
    \define@key{luaindex.setindex}{sort}{\edef\luaindex@sort{##1}}%
    \define@key{luaindex.setindex}{pageformat}{\def\luaindex@pageformat{##1}}%
    \define@key{luaindex.setindex}{locale}{\luaindexsetup{locale=#1}}%
    \setkeys{luaindex.setindex}{#1}%
    \protected@write\@indexfile{\let\luatexluaescapestring\relax}{%
        luaindex.insert('\luatexluaescapestring{\luaindex@name}',
                             '{\luatexluaescapestring{\luaindex@pageformat{\thepa
                             '\luatexluaescapestring{\luaindex@sort}',
                             '\luatexluaescapestring{#2}',
    }%
    \aftergroup\lua@@subindex
  \endgroup
}
\newcommand*{\lua@@subindex}[2][]{%
  \begingroup
    \set@display@protect
    \edef\luaindex@sort{#2}%
    \define@key{luaindex.setindex}{sort}{\edef\luaindex@sort{##1}}%
    \setkeys{luaindex.setindex}{#1}%
    \protected@write\@indexfile{\let\luatexluaescapestring\relax}{%
                             \@spaces
                             '\luatexluaescapestring{\luaindex@sort}',
                             '\luatexluaescapestring{#2}')
    }%
  \endgroup
  \@esphack
}
```

`\luasubsubindex`
`\lua@subsubindex`
`\lua@@@subindex`

Same like `\luaindex` but to produce a sub-sub-entry, that is a sub-entry to a sub-entry:

`\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*⟩.

```
\newcommand*{\luasubsubindex}[1]{%
  \@bsphack
  \begingroup
    \edef\luaindex@name{#1}%
    \lua@subsubindex
}
\newcommand*{\lua@subsubindex}[2][]{%
    \set@display@protect
    \edef\luaindex@sort{#2}%
    \define@key{luaindex.setindex}{sort}{\edef\luaindex@sort{##1}}%
    \define@key{luaindex.setindex}{pageformat}{\def\luaindex@pageformat{##1}}%
    \define@key{luaindex.setindex}{locale}{%
      \luaindexsetup{locale=#1}%
    }
    \setkeys{luaindex.setindex}{#1}%
    \protected@write\@indexfile{\let\luatexluaescapestring\relax}{%
        luaindex.insert('\luatexluaescapestring{\luaindex@name}',
                             '{\luatexluaescapestring{\luaindex@pageformat{\thepa
                             '\luatexluaescapestring{\luaindex@sort}',
                             '\luatexluaescapestring{#2}',
    }%
    \aftergroup\lua@@@subindex
  \endgroup
}
\newcommand*{\lua@@@subindex}[2][]{%
  \begingroup
    \set@display@protect
    \edef\luaindex@sort{#2}%
    \define@key{luaindex.setindex}{sort}{\edef\luaindex@sort{##1}}%
    \setkeys{luaindex.setindex}{#1}%
    \protected@write\@indexfile{\let\luatexluaescapestring\relax}{%
                             \@spaces
                             '\luatexluaescapestring{\luaindex@sort}',
                             '\luatexluaescapestring{#2}',
    }%
    \aftergroup\lua@@subindex
  \endgroup
}
```

`\makeindex`
`\index`
`\subindex`
`\subsubindex`

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

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

`singlepass=false`	`singlepass=true`
index of previous LuaLaTeX run will be printed	index of current LuaLaTeX 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*⟩`}`.

```
\renewcommand*{\makeindex}{%
  \newindex{general}%
  \renewcommand*\index{\luaindex{general}}%
  \newcommand*\subindex{\luasubindex{general}}%
  \newcommand*\subsubindex{\luasubsubindex{general}}%
}
```

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

```
\newcommand*{\printindex}[1][]{%
  \begingroup
    \edef\luaindex@name{general}%
    \define@key{luaindex.setindex}{index}{\edef\luaindex@name{##1}}%
    \define@key{luaindex.setindex}{singlepass}[true]{%
      \setupluaindex{singlepass}{##1}%
    }%
    \setkeys{luaindex.setindex}{#1}%
    \if@luaindexsinglepass
      \closeout\@indexfile
      \clearpage
      \directlua{%
        luaindex.removeentries('\luatexluaescapestring{\luaindex@name}')
        dofile('\jobname.ldx')
      }%
    \fi
    \directlua{%
      luaindex.printindex('\luatexluaescapestring{\luaindex@name}')
    }%
  \endgroup
}
```

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

`\indexgroup`
`\indexspace`
`\symbolsname`
`\numbersname`

Each index is grouped. Index groups are symbols, numbers and each first letter. Each group starts with `\indexgroup{`⟨*group*⟩`}` with group is either `\symbolsname`, `\numbersname` or a upper case letter. In difference to other index processors no automatic `\indexspace` will be added before each group. So we define `\indexgroup` to add it.

```
\providecommand*{\indexgroup}[1]{%
  \indexspace\textbf{#1}\nopagebreak
}
\providecommand*{\indexspace}{%
  \def\indexspace{\vskip\baselineskip}
}
\providecommand*{\symbolsname}{Symbols}
\providecommand*{\numbersname}{Numbers}
\AtBeginDocument{%
  \providecaptionname{english}\symbolsname{Symbols}%
  \providecaptionname{english}\numbersname{Numbers}%
  \providecaptionname{german}\symbolsname{Symbole}%
  \providecaptionname{german}\numbersname{Zahlen}%
  \providecaptionname{ngerman}\symbolsname{Symbole}%
  \providecaptionname{ngerman}\numbersname{Zahlen}%
  \providecaptionname{austrian}\symbolsname{Symbole}%
  \providecaptionname{austrian}\numbersname{Zahlen}%
  \providecaptionname{naustrian}\symbolsname{Symbole}%
  \providecaptionname{naustrian}\numbersname{Zahlen}%
  \providecaptionname{french}\symbolsname{Symbole}%
  \providecaptionname{french}\numbersname{Chiffres}%
  \providecaptionname{spanish}\symbolsname{Simbolos}%
  \providecaptionname{spanish}\numbersname{N\'umeros}%
}
```

`\indexpagenumbers`
`\indexpagenumber`
`\indexpagenumbersep`
`\index@pagenumbersep`

The page numbers of an entry are printed all together as argument 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.

```
\providecommand*{\indexpagenumbers}[1]{%
  \def\index@pagenumbersep{\let\index@pagenumbersep\indexpagenumbersep}%
  \nobreakspace-- #1}
\providecommand*{\indexpagenumber}[1]{\index@pagenumbersep #1}
\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.

```
\documentclass{article}
\usepackage[ngerman]{babel}
\usepackage{blindtext}
\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 LuaLaTeX run instead of two or more. But with this printing of the index will produce a new page.

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

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

```
\makeindex
```

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

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

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

```
\begin{document}

\blindtext[10]
A\index{B ist der zweite Buchstabe}
aber\index{aber ist ein Wort}
D\index{D ist der vierte Buchstabe}
A\index{A ist der erste Buchstabe}
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:

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

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

```
D\index{\textbf{D} ist der Buchstabe nach C}
Y\index{Y ist der \textbf{vorletzte} Buchstabe}
Z\index{Z ist der letzte Buchstabe}
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.

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

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

```
\clearpage

A\index{A ist der erste Buchstabe}
Ae\index{Ae ist kein Buchstabe, sondern zwei}
```

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.

```
Kompliziert\subindex{Diverses}{Untereintrag}
Noch komplizierter\subindex{Diverses}{Obereintrag}
Noch komplizierter\%
subindex{Diverses}[sort=Obereintra,pageformat=\emph]{Untereintrag}
Noch komplizierter%
\subsubindex{Diverses}{Untereintrag}{Unteruntereintrag}
```

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

```
\printindex
\end{document}
```

Index

Numbers written in italic 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.

A

`\alsoname` 395

I

`\index` 509
`\index@pagenumbersep` 560
`\indexgroup` 536
`\indexpagenumber` 560
`\indexpagenumbers` 560
`\indexpagenumbersep` 560
`\indexspace` 536

L

`locale` (Option) 376
`\lua@@@subindex` 471
`\lua@@subindex` 435
`\lua@subindex` 435
`\lua@subsubindex` 471
`\luaindex` 413
`\luaindex@pageformat` 388
`\luaindex@sortorder` 372
`\luasubindex` 435
`\luasubsubindex` 471

M

`\makeindex` 509

N

`\newindex` 399
`\numbersname` 536

O

Optionen:
  `locale` 376
  `pageformat` 388
  `singlepass` 392
  `sortorder` 372

P

`pageformat` (Option) 388
`\printindex` 515

S

`\see` 395
`\seealso` 395
`\seename` 395
`\setupluaindex` 394
`singlepass` (Option) 392
`sortorder` (Option) 372
`\subindex` 509
`\subsubindex` 509
`\symbolsname` 536

Change History

v0.1
  General: start of new package . . 1
v0.1b
  General: prefix 'koma.' removed from Lua module 1
  Using package `luatexbase-compat` 13
  Using package `luatexbase-modutils` 13