The `Lua-UCA` library

Michal Hoftich*

Version v0.1e
2025-03-31

Contents

1 Introduction

This package adds support for the Unicode collation algorithm[1] for Lua 5.3 and later. It is mainly intended for use with LuaTEXand working TEX distribution, but it can work also as a standalone Lua module. You will need to install a required Lua-uni-algos[2] package by hand in that case.

---

*<michal.h21@gmail.com>

[1] https://unicode.org/reports/tr10/

[2] https://github.com/latex3/lua-uni-algos

1.1 Usage

To sort a table using Czech collation rules:

```
kpse.set_program_name "luatex"
local ducet = require "lua-uca.lua-uca-ducet"
local collator = require "lua-uca.lua-uca-collator"
local languages = require "lua-uca.lua-uca-languages"

local collator_obj = collator.new(ducet)
-- load Czech rules
collator_obj = languages.cs(collator_obj)

local t = {"cihla",  "chochol", "hudba", "jasan", "čáp"}

table.sort(t, function(a,b)
  return collator_obj:compare_strings(a,b)
end)

for _, v in ipairs(t) do
  print(v)
end
```

The output:

```
cihla čáp hudba chochol jasan
```

More samples of the library usage can be found in the source repository of this package on Github[3].

1.2 Use with Xindex processor

Xindex[4] is flexible index processor written in Lua by Herbert Voß. It has built-in `Lua-UCA` support starting with version `0.23`. The support can be requested using the `-u` option:

```
xindex -u -l no -c norsk filename.idx
```

1.3 Use with LuaJIT

The default version of `lua-uca-ducet` fails with Luajit. You can use alternative version of this file, `lua-uca-ducet-jit`.

1.4 Change sorting rules

The simplest way to change the default sorting order is to use the `tailor_string` method of the `collator_obj` object. It updates the collator object using special syntax which is subset of the format used by the Unicode locale data markup language[5].

---

[3] https://github.com/michal-h21/lua-uca

[4] https://www.ctan.org/pkg/xindex

[5] https://www.unicode.org/reports/tr35/tr35-collation.html#Orderings

```
collator_obj:tailor_string "&a<b"
```

Full example with Czech rules:

```
kpse.set_program_name "luatex"
local ducet = require "lua-uca.lua-uca-ducet"
local collator = require "lua-uca.lua-uca-collator"
local languages = require "lua-uca.lua-uca-languages"

local collator_obj = collator.new(ducet)
local tailoring = function(s) collator_obj:tailor_string(s) end

tailoring "&c<č<<<Č"
tailoring "&h<ch<<<cH<<<Ch<<<CH"
tailoring "&R<ř<<<Ř"
tailoring "&s<š<<<Š"
tailoring "&z<ž<<<Ž"
```

Note that the sequence of letters `ch`, `Ch`, `cH` and `CH` will be sorted after `h`

It is also possible to expand a letter to multiple letters, like this example for DIN 2:

```
tailoring "&Ö=Oe"
tailoring "&ö=oe"
```

Some languages, like Norwegian, sort uppercase letters before lowercase. This can be enabled using `collator_obj:uppercase_first()` function:

```
local tailoring = function(s) collator_obj:tailor_string(s) end
collator_obj:uppercase_first()
tailoring("&D<<đ<<<Đ<<ð<<<Ð")
tailoring("&th<<<þ")
tailoring("&TH<<<Þ")
tailoring("&Y<<ü<<<Ü<<ű<<<Ű")
tailoring("&ǀ<æ<<<Æ<<ä<<<Ä<ø<<<Ø<<ö<<<Ö<<ő<<<Ő<å<<<Å<<<aa<<<Aa<<<AA")
tailoring("&oe<<œ<<<Œ")
```

Some languages, for example Canadian French, sort accent backwards, like gêne < gëne < gêné. In this case, you can set the `collator_obj.accents_backward` variable to `true`.

1.4.1 Script reordering

Many languages sort different scripts after the script this language uses. As Latin based scripts are sorted first, it is necessary to reorder scripts in such cases.

The `collator_obj:reorder` function takes table with scripts that need to be reordered. For example Cyrillic can be sorted before Latin using:

```
collator_obj:reorder {"cyrillic"}
```

In German or Czech, numbers should be sorted after all other characters. This can be done using:

```
collator_obj:reorder {"others", "digits"}
```

The special keyword ”others” means that the scripts that follows in the table will be sorted at the very end.

1.5 Headers for index entries

In some languages, for example Czech, multiple letters may count as one character. This is the case of the *ch* character.

Lua-UCA provides function `collator_obj:get_lowest_char()`. It returns table with UTF-8 codepoints for correct first character for a given language that can be used for example as an index header.

```
local czech = collator.new(ducet)
languages.cs(czech)
-- first we need to convert string to codepoints
local codepoints = czech:string_to_codepoints("Chrobák")
local first_char = czech:get_lowest_char(codepoints)
-- it should print letters "ch"
print(utf8.char(table.unpack(first_char)))
-- you can also specify position of the character
local second_char = czech:get_lowest_char(codepoints, 2)
-- it should print letter "h", as it is second codepoint in the string
print(utf8.char(table.unpack(second_char)))
```

1.6 Unicode normalization

By default, no Unicode normalization is used internally. You can explicitly request normalization that use the Uninormalize package[6]. Note that it will significantly increase the procesing time.

There are two normalization methods, NFC and NFD. They can be enabled using `collation.use_nfc()` and `collation.use_nfd()` functions.

2 What is missing

- Algorithm for setting implicit sort weights of characters that are not explicitly listed in DUCET.
- Special handling of CJK scripts.

3 Available Languages

The `lua-uca-languages` library provides the following langauges: `af`, `am`, `ar`, `as`, `az`, `be`, `bg`, `bn`, `bs`, `bs_cyrl`, `ca`, `chr`, `cs`, `cy`, `da`, `de`, `de_din2`, `dsb`, `dz`, `ee`, `el`, `en`, `eo`, `es`, `et`, `fa`, `fi`, `fil`, `fo`, `fr`, `fr_backward_accents`, `ga`, `gl`, `gu`, `ha`, `haw`, `he`, `hi`, `hr`, `hsb`, `hu`, `hy`, `id`, `ig`, `is`, `it`, `ja`, `ka`, `kk`, `kl`, `km`, `kn`, `ko`, `kok`, `ky`, `lb`, `lkt`, `ln`, `lo`, `lt`, `lv`, `mk`, `ml`, `mn`, `mr`, `ms`, `mt`, `my`, `nb`, `ne`, `nl`, `nn`, `no`, `om`, `or`, `pa`, `pl`, `ps`, `pt`, `ro`, `ru`, `se`, `si`,

---

[6] https://ctan.org/pkg/uninormalize?lang=en

`sk, sl, smn, sq, sr, sr_latn, sv, sw, ta, te, th, tk, to, tr, ug, uk, ur, uz, vi, vo, wae, wo, yi, yo, zh, zu`

If you want to requrest language not listed in this listing, or if you had created support code for one, please contact the package author by mail or using issue tracker on package's Github profile.

4 Lua-UCA hacking

You need the full installation from Github[7] in order to do stuff described in this section. Package distributed on CTAN doesn't contain all necessary files.

4.1 Install

The package needs to download Unicode collation data and convert it to a Lua table. It depends on `wget` and `unzip` utilities. All files can be downloaded using Make:

```
make
```

To install the package in the local TEXMF tree, run:

```
make install
```

4.2 New language support

To add a new language, add new function to `src/lua-uca/lua-uca-languages.lua` file. The function name should be short language code. Example function for the Russian language:

```
languages.ru = function(collator_obj)
  collator_obj:reorder{ "cyrillic" }
  return collator_obj
end
```

The language function takes the Collator object as a parameter. Methods showed in the *Change sorting rules* section can be used with this object.

The `data/common/collation/` directory in the source repository contains files from the `CLDR` project. They contain rules for many languages. The files needs to be normalized to the NFC form[8], for example using:

```
cat cs.xml | uconv -x any-nfc -o cs.xml
```

The `uconv` utility is a part of the ICU Project[9].

Sorting rules for a language are placed in the `<collation>` element. Multiple `<collation>` elements may be present in the XML file. It is usually best to chose the one with attribute `type="standard"`.

The following example contains code from `da.xml`:

---

[7] https://github.com/michal-h21/lua-uca

[8] https://en.wikipedia.org/wiki/Unicode_equivalence

[9] http://userguide.icu-project.org/

```
[caseFirst upper]
&D<<đ<<<Đ<<ð<<<Ð
&th<<<þ
&TH<<<Þ
&Y<<ü<<<Ü<<ű<<<Ű
&[before 1]ǀ<æ<<<Æ<<ä<<<Ä<ø<<<Ø<<ö<<<Ö<<ő<<<Ő<å<<<Å<<<aa<<<Aa<<<AA
&oe<<œ<<<Œ
```

This is translated to Lua code in `lua-uca-languages.lua` in the following way:

```
languages.da = function(collator_obj)
  -- helper function for more readable tailoring definition
  local tailoring = function(s) collator_obj:tailor_string(s) end
  collator_obj:uppercase_first()
  tailoring("&D<<đ<<<Đ<<ð<<<Ð")
  tailoring("&th<<<þ")
  tailoring("&TH<<<Þ")
  tailoring("&Y<<ü<<<Ü<<ű<<<Ű")
  tailoring("&ǀ<æ<<<Æ<<ä<<<Ä<ø<<<Ø<<ö<<<Ö<<ő<<<Ő<å<<<Å<<<aa<<<Aa<<<AA")
  tailoring("&oe<<œ<<<Œ")
  return collator_obj
end
```

Pull requests with new language support are highly appreciated.

4.3 Support files in the source distribution

The `xindex` directory contains some examples for configuration of `Xindex`, Lua based indexing system. Run `make xindex` command to compile them.

`Xindex` has built-in support for Lua-UCA since version `0.23`, it can be requested using the `-u` option.

The `tools/indexing-sample.lua` file provides a simple indexing processor, independent of any other tool.

4.4 Testing

You can run unit tests using the following command:

```
make test
```

Testing requires Busted[10] testing framework installed on your system. Tests are placed in the `spec` directory and they provide more examples of the package usage.

5 License

Copyright 2021 Michal Hoftich

---

[10] `https://olivinelabs.com/busted/`

6 Changelog

2025-03-31

- fixed `de_din2` sorting rules (thanks to Herbert Voss).

2024-05-09

- version `0.1d` released.

2024-05-07

- fixed French support (thanks to Daniel Flipo).

2024-05-04

- fixed Chinese support (thanks to Zeping Lee).
- version `0.1c` released.

2022-03-08

- working on better French support (thanks to Daniel Flipo).
- don’t use NFC normalization.
- added `accents_backward` option.

2021-11-10

- version `0.1b` released.

2021-11-09

- cache string to codepoint conversion.
- use NFC normalization with LuaTeX.

2021-09-16

- version `0.1a` released.
- added sorting rules for all languages contained in CLDR collation files.

2020-06-09

- moved development information that depends on files not distributed on CTAN to `HACKING.md`.
- extended documentation.

2020-03-24

- version `0.1` released.
- initial version for CTAN.