The `hyphen.cfg` file for LuaTEX

Khaled Hosny, Élie Roux, and Manuel Pégourié-Gonnard
`khaledhosny@eglug.org`
`elie.roux@telecom-bretagne.eu`
`mpg@elzevir.fr`

2013/05/16 v1.6

Abstract

This package is mainly a Lua module, to be used by **Babel** and **polyglossia** to adapt their hyphenation patterns loading mechanism to LuaTEX's dynamic pattern loading capabilities. It makes use of a `language.dat.lua` file (whose format is described below) that should be present in the distribution, in addition to the regular `language.dat` file.

Babel needed to be updated – this used to be the goal of this package – before version 3.9 (TEXLive 2013) and **polyglossia** handles LuaTEX since version 1.3 (TEXLive 2013).

There is a version of `etex.src` modified for the same reasons using similar code, which also makes use of the `luatex-hyphen.lua` and `language.dat.lua` files described here.

1 Documentation

Hyphenation patterns should be loaded at runtime with LuaTEX: if they appear in the format, they will be rehashed when the format is loaded anyway, which makes the format quite long to load (many seconds even on modern machines) and provides for bad user experience. Hence, it is desirable to load as few patterns as possible in the format, and load on-demand the needed patterns at runtime.

This package provides a modified version of hyphen.cfg adapted to LuaTEX, as well as a supporting Lua module. Since a lot of things, especially the catcodes, are not as predictable at runtime than at format creation time, we don't `\input` the usual pattern files, but rather load the patterns using the Lua interface, using a special plain text version of the pattern files if available.

The existence and file name of such a version cannot be guessed, so we need a specific database: the file `language.dat.lua`. This file should be loadable by Lua and return a table whose keys are the canonical language names as found in `language.dat`, and the values are Lua tables consisting of:

1. A fixed part with one mandatory field:

```
synonyms = { <string> alternative name, ...}
```

This field's value must be the same as in `language.dat`.

2. A variable part consisting of either:

   - For most languages:

     ```
     patterns = <string> filenames for patterns
     hyphenation = <string> filenames for exceptions
     ```

     Each string contains a coma-separated list of file names (whitespace before or after the coma is not accepted). The files given by `patterns` (resp. `hypenation`) must be plain text files encoded in UTF-8, with only patterns (resp. exceptions) and not even comments: their content will be used directly without being parsed by TEX. If one of these keys is missing or is the empty string, it is ignored and no patterns (resp. exceptions) are loaded for this language.

   - Special cases are supported by a field `special`. Currently, the following kind of values are recognized:

     `'disabled:<reason>'` allows to disable specific languages: when the user tries to load this language, an error will be issued, with the `<reason>`.

     `'language0'` only `english` should use this type of special, to indicate it is normally dumped in the format as `\language0` (see below).

     Special languages may have `*hyphenmin` information when it makes sense (mostly `\language0`).

3. Optional fields may be added. For example:

   ```
   loader = <string> name of the TeX loader
   lefthyphenmin = <number> value for \lefthyphenmin
   righthyphenmin = <number> value for \righthyphenmin
   ```

   Those fields are present in `language.dat.lua` as generated by `tlmgr`, for example, but they *are not* used by the present code in any way.

Languages that are mentioned in `language.dat` but not in `language.dat.lua` will be loaded in the format. So, if the `language.dat.lua` file is missing or incomplete, languages will just go back to the "old" behaviour, resulting in longer startup time, which seems less bad than complete breakage.

For backward compatibility, Knuth's original patterns for US English are always loaded in the format, as `\language0`.[1]

---

[1] It is assumed to be the first entry in `language.dat`.

2 Implementation

```
⟨*lua⟩
```

Start a Lua module, two functions for error and information reporting.

```
luatexhyphen = luatexhyphen or {}
local luatexhyphen = luatexhyphen
local function wlog(msg, ...)
    texio.write_nl(’log’, ’luatex-hyphen: ’..msg:format(...))
end
local function err(msg, ...)
    error(’luatex-hyphen: ’..msg:format(...), 2)
end
```

Load the `language.dat.lua` file with the Lua version of the language database.

```
local dbname = ”language.dat.lua”
local language_dat
local dbfile = kpse.find_file(dbname, ’lua’)
if not dbfile then
    err(”file not found: ”..dbname)
else
    wlog(’using data file: %s’, dbfile)
    language_dat = dofile(dbfile)
end
```

Look up a language in the database, and return the associated information, as well as the canonical name of the language.

```
local function lookupname(name)
    if language_dat[name] then
        return language_dat[name], name
    else
        for canon, data in pairs(language_dat) do
            for _,syn in ipairs(data.synonyms) do
                if syn == name then
                    return data, canon
                end
            end
        end
    end
end
luatexhyphen.lookupname = lookupname
```

Set hyphenation patterns and exceptions for a language given by its name (in the database) and number (value of `\language`). Doesn’t return anything, but will call `error()` if things go wrong.

```
local function loadlanguage(lname, id)
    if id == 0 then
        return
    end
    local msg = ”loading%s patterns and exceptions for: %s (\\language%d)”
```

Lookup the language in the database.

```
    local ldata, cname = lookupname(lname)
    if not ldata then
        err(”no entry in %s for this language: %s”, dbname, lname)
    end
```

Handle special languages.

```
    if ldata.special then
        if ldata.special:find(’^disabled:’) then
            err(”language disabled by %s: %s (%s)”, dbname, cname,
                ldata.special:gsub(’^disabled:’, ’’))
        elseif ldata.special == ’language0’ then
            err(”\\language0 should be dumped in the format”)
        else
            err(”bad entry in %s for language %s”)
        end
    end
```

The generic case: load hyphenation patterns and exceptions from files given by the language code.

```
    wlog(msg, ’’, cname, id)
    for _, item in ipairs{’patterns’, ’hyphenation’} do
        local filelist = ldata[item]
        if filelist ~= nil and filelist ~= ’’ then
          for _, file in ipairs(filelist:explode(’,’)) do
            local file = kpse.find_file(file) or err(”file not found: %s”, file)
            local fh = io.open(file, ’r’)
            local data = fh:read(’*a’) or err(”file not readable: %s”, f)
            fh:close()
            lang[item](lang.new(id), data)
          end
        else
            if item == ’hyphenation’ then item = item..’ exceptions’ end
            wlog(”info: no %s for this language”, item)
        end
    end
end
luatexhyphen.loadlanguage = loadlanguage
```

Add Babel’s “dialects” as synonyms.

```
local function adddialect(dialect, language)
    if dialect ~= ’0’ then
        dialect = dialect:gsub(’l@’, ’’)
        language = language:gsub(’l@’, ’’)
        data = lookupname(language)
        if data then
            data.synonyms[#data.synonyms+1] = dialect
        end
    end
end
luatexhyphen.adddialect = adddialect
⟨/lua⟩
```