The lparse package

Josef Friedrich
josef@friedrich.rocks
github.com/Josef-Friedrich/lparse

0.1.0 from 2023/01/29

```
\def\test{\par\directlua{
  local oarg, star, marg = lparse.scan('o s m')
  tex.print('o: ' .. tostring(oarg))
  tex.print('s: ' .. tostring(star))
  tex.print('m: ' .. tostring(marg))
}}

\test{marg} % o: nil s: false m: marg
\test[oarg]{marg} % o: oarg s: false m: marg
\test[oarg]*{marg} % o: oarg s: true m: marg
```

Contents

1 Introduction

The name `lparse` is derived from `xparse`. The `x` has been replaced by `l` because this package only works with LuaTEX. `l` stands for *Lua*. Just as with `xparse`, it is possible to use a special syntax consisting of single letters to express the arguments of a macro. However, `lparse` is able to read arguments regardless of the macro systemd used - whether LATEX or ConTEXt or even plain TEX. Of course, LuaTEX must always be used as the engine.

1.1 Similar projects

For ConTEXt there is a similar argument scanner (see ConTEXt Lua Document cld-mkiv). This scanner is implemented in the following files: toks-scn.lua toks-aux.lua toks-ini.lua ConTEXt scanner apparently uses the token library of the LuaTeX successor project luametaTEX: lmttokenlib.c

2 Description of the argument specification

The following lists, which describe the individual argument types, are taken from the manuals usrguide and xparse. The descriptive texts of the individual argument types have only been slightly adjusted. The argument types that are not yet supported are bracketed.

m A standard mandatory argument, which can either be a single token alone or multiple tokens surrounded by curly braces `{}`. Regardless of the input, the argument will be passed to the internal code without the outer braces. This is the `xparse` type specifier for a normal TEX argument.

r Given as `r`⟨*token1*⟩⟨*token2*⟩, this denotes a “required” delimited argument, where the delimiters are ⟨*token1*⟩ and ⟨*token2*⟩. If the opening delimiter ⟨*token1*⟩ is missing, `nil` will be returned after a suitable error.

R Given as `R`⟨*token1*⟩⟨*token2*⟩`{`⟨*default*⟩`}`, this is a “required” delimited argument as for `r`, but it has a user-definable recovery ⟨*default*⟩ instead of `nil`.

v Reads an argument “verbatim”, between the following character and its next occurrence.

(b) Not implemented! Only suitable in the argument specification of an environment, it denotes the body of the environment, between `\begin{`⟨*environment*⟩`}` and `\end{`⟨*environment*⟩`}`.

The types which define optional arguments are:

o A standard LATEX optional argument, surrounded with square brackets, which will supply `nil` if not given (as described later).

d Given as `d`⟨*token1*⟩⟨*token2*⟩, an optional argument which is delimited by ⟨*token1*⟩ and ⟨*token2*⟩. As with `o`, if no value is given `nil` is returned.

O Given as `O{`⟨*default*⟩`}`, is like `o`, but returns ⟨*default*⟩ if no value is given.

D Given as `D`⟨*token1*⟩⟨*token2*⟩`{`⟨*default*⟩`}`, it is as for `d`, but returns ⟨*default*⟩ if no value is given. Internally, the `o`, `d` and `O` types are short-cuts to an appropriated-constructed `D` type argument.

s An optional star, which will result in a value `true` if a star is present and `false` otherwise (as described later).

t An optional ⟨*token*⟩, which will result in a value `true` if ⟨*token*⟩ is present and `false` otherwise. Given as `t`⟨*token*⟩.

(e) Not implemented! Given as `e{`⟨*tokens*⟩`}`, a set of optional *embellishments*, each of which requires a *value*. If an embellishment is not present, `-NoValue-` is returned. Each embellishment gives one argument, ordered as for the list of ⟨*tokens*⟩ in the argument specification. All ⟨*tokens*⟩ must be distinct. *This is an experimental type.*

(E) Not implemented! As for `e` but returns one or more ⟨*defaults*⟩ if values are not given: `E{`⟨*tokens*⟩`}{`⟨*defaults*⟩`}`.

3 Implementation

3.1 lparse.lua

```lua
-- lparse.lua
-- Copyright 2023 Josef Friedrich
--
-- This work may be distributed and/or modified under the
-- conditions of the LaTeX Project Public License, either version 1.3c
-- of this license or (at your option) any later version.
-- The latest version of this license is in
--   http://www.latex-project.org/lppl.txt
-- and version 1.3c or later is part of all distributions of LaTeX
-- version 2008/05/04 or later.
--
-- This work has the LPPL maintenance status `maintained'.
--
-- The Current Maintainer of this work is Josef Friedrich.
--
-- This work consists of the files lparse.lua, lparse.tex,
-- and lparse.sty.
---
if lpeg == nil then
  lpeg = require('lpeg')
end

---
---@param spec string
---@return Argument[]
local function parse_spec(spec)
  local V = lpeg.V
  local P = lpeg.P
  local Set = lpeg.S
  local Range = lpeg.R
  local CaptureFolding = lpeg.Cf
  local CaptureTable = lpeg.Ct
  local Cc = lpeg.Cc
  local CaptureSimple = lpeg.C

  local function add_result(result, value)
    if not result then
      result = {}
    end
    table.insert(result, value)
    return result
  end

  local function collect_delims(a, b)
    return { init_delim = a, end_delim = b }
  end

  local function collect_token(a)
    return { token = a }
  end

  local function set_default(a)
    return { default = a }
  end

  local function combine(...)
    local args = { ... }

    local output = {}

    for _, arg in ipairs(args) do
      if type(arg) ~= 'table' then
        arg = {}
      end

      for key, value in pairs(arg) do
        output[key] = value
      end

    end

    return output
  end

  local function ArgumentType(letter)
    local function get_type(l)
      return { argument_type = l }
    end
    return CaptureSimple(P(letter)) / get_type
  end

  local T = ArgumentType

  local pattern = P({
    'init',
    init = V('whitespace') ^ 0 *
      CaptureFolding(CaptureTable('') * V('list'), add_result),

    list = (V('arg') * V('whitespace') ^ 1) ^ 0 * V('arg') ^ -1,

    arg = V('m') + V('r') + V('R') + V('v') + V('o') + V('d') + V('O') +
      V('D') + V('s') + V('t'),

    m = T('m') / combine,

    r = T('r') * V('delimiters') / combine,

    R = T('R') * V('delimiters') * V('default') / combine,

    v = T('v') * Cc({ verbatim = true }) / combine,

    o = T('o') * Cc({ optional = true }) / combine,

    d = T('d') * V('delimiters') * Cc({ optional = true }) / combine,

    O = T('O') * V('default') * Cc({ optional = true }) / combine,

    D = T('D') * V('delimiters') * V('default') *
      Cc({ optional = true }) / combine,

    s = T('s') * Cc({ star = true }) / combine,

    t = T('t') * V('token') / combine,

    token = V('delimiter') / collect_token,

    delimiter = CaptureSimple(Range('!~')),

    delimiters = V('delimiter') * V('delimiter') / collect_delims,

```

```lua
    whitespace = Set(' \t\n\r'),

    default = P('{') * CaptureSimple((1 - P('}')) ^ 0) * P('}') /
      set_default,
  })

  return pattern:match(spec)

end

---
---Scan for an optional argument.
---
---@param init_delim? string # The character that marks the beginning of an optional argument (by default `[`).
---@param end_delim? string # The character that marks the end of an optional argument (by default `]`).
---
---@return string|nil # The string that was enclosed by the delimiters. The delimiters themselves are not returned.
local function scan_delimited(init_delim, end_delim)
  if init_delim == nil then
    init_delim = '['
  end
  if end_delim == nil then
    end_delim = ']'
  end

  ---
  ---@param t Token
  ---
  ---@return string
  local function convert_token_to_string(t)
    if t.index ~= nil then
      return utf8.char(t.index)
    else
      return '\\' .. t.csname
    end
  end

  local delimiter_stack = 0

  local function get_next_char()
    local t = token.get_next()
    local char = convert_token_to_string(t)
    if char == init_delim then
      delimiter_stack = delimiter_stack + 1
    end

    if char == end_delim then
      delimiter_stack = delimiter_stack - 1
    end
    return char, t
  end

  local char, t = get_next_char()

  if t.cmdname == 'spacer' then
    char, t = get_next_char()
  end

  if char == init_delim then
```

```
    local output = {}

    char, t = get_next_char()

    -- "while" better than "repeat ... until": The end_delimiter is
    -- included in the result output.
    while not (char == end_delim and delimiter_stack == 0) do
      table.insert(output, char)
      char, t = get_next_char()
    end
    return table.concat(output, '')
  else
    token.put_next(t)
  end
end

---@class Argument
---@field argument_type? string
---@field optional? boolean
---@field init_delim? string
---@field end_delim? string
---@field dest? string
---@field star? boolean
---@field default? string
---@field verbatim? boolean
---@field token? string

---@class Parser
---@field args Argument[]
---@field result any[]
local Parser = {}
---@private
Parser.__index = Parser

function Parser:new(spec)
  local parser = {}
  setmetatable(parser, Parser)
  parser.spec = spec
  parser.args = parse_spec(spec)
  parser.result = parser:parse(parser.args)
  return parser
end

---@return any[]
function Parser:parse()
  local result = {}
  local index = 1
  for _, arg in pairs(self.args) do
    if arg.star then
      -- s
      result[index] = token.scan_keyword('*')
    elseif arg.token then
      -- t
      result[index] = token.scan_keyword(arg.token)
    elseif arg.optional then
      -- o d O D
      local oarg = scan_delimited(arg.init_delim, arg.end_delim)
      if arg.default and oarg == nil then
        oarg = arg.default
      end
      result[index] = oarg
    elseif arg.init_delim and arg.end_delim then
```

```
      -- r R
      local oarg = scan_delimited(arg.init_delim, arg.end_delim)
      if arg.default and oarg == nil then
        oarg = arg.default
      end
      if oarg == nil then
        tex.error('Missing required argument')
      end
      result[index] = oarg
    else
      -- m v
      local marg = token.scan_argument(arg.verbatim ~= true)
      if marg == nil then
        tex.error('Missing required argument')
      end
      result[index] = marg
    end
    index = index + 1
  end
  return result
end

---@private
function Parser:set_result(...)
  self.result = { ... }
end

function Parser:assert(...)
  local arguments = { ... }
  for index, arg in ipairs(arguments) do
    assert(self.result[index] == arg, string.format(
      'Argument at index %d doesn’t match: "%s" != "%s"',
      index, self.result[index], arg))
  end
end

---
---@return string|boolean|nil ...
function Parser:export()
  -- #self.arg: to get all elements of the result table, also elements
  -- with nil values.
  return table.unpack(self.result, 1, #self.args)
end

function Parser:debug()
  for index = 1, #self.args do
    print(index, self.result[index])
  end
end

---@return Parser
local function create_parser(spec)
  return Parser:new(spec)
end

local function scan(spec)
  local parser = create_parser(spec)
  return parser:export()
end

return { Parser = create_parser, scan = scan, parse_spec = parse_spec }
```

3.2 lparse.tex

```
%% lparse.tex
%% Copyright 2023 Josef Friedrich
%
% This work may be distributed and/or modified under the
% conditions of the LaTeX Project Public License, either version 1.3c
% of this license or (at your option) any later version.
% The latest version of this license is in
%   http://www.latex-project.org/lppl.txt
% and version 1.3c or later is part of all distributions of LaTeX
% version 2008/05/04 or later.
%
% This work has the LPPL maintenance status `maintained'.
%
% The Current Maintainer of this work is Josef Friedrich.
%
% This work consists of the files lparse.lua, lparse.tex,
% and lparse.sty.

\directlua
{
  lparse = require('lparse')
}
```

3.3 lparse.sty

```
%% lparse.sty
%% Copyright 2023 Josef Friedrich
%
% This work may be distributed and/or modified under the
% conditions of the LaTeX Project Public License, either version 1.3c
% of this license or (at your option) any later version.
% The latest version of this license is in
%   http://www.latex-project.org/lppl.txt
% and version 1.3c or later is part of all distributions of LaTeX
% version 2008/05/04 or later.
%
% This work has the LPPL maintenance status `maintained'.
%
% The Current Maintainer of this work is Josef Friedrich.
%
% This work consists of the files lparse.lua, lparse.tex,
% and lparse.sty.

\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{lparse}[2023/01/29 v0.1.0 Parse and scan macro arguments in Lua on
↪  LuaTeX using a xparse like argument specification]

\input lparse.tex
```