



Inhaltsverzeichnis

Modul:Parameters

Vorlage:Lua

Implements [Vorlage:TI](#)

```
-- This module implements [[Template:Parameters]].
-- [SublimeLinter luacheck-globals:mw]

local DEFINITIONS = {
    alt = {
        code = '<!-- text alternative for image; see WP:ALT -->',
        dlist = 'text alternative for image; see [[WP:ALT]]',
    },
    coordinates = {
        code = '<!-- use {{Coord}} -->',
        dlist = 'using {{tl|Coord}}',
    },
    coords = {
        code = '<!-- use {{Coord}} -->',
        dlist = 'using {{tl|Coord}}',
    },
    native_name = {
        code = '<!-- name in local language; if more than one, separate
            using {{Plainlist}} use {{lang}}, and omit native_name_1
            ..
            using {{tl|Plainlist}}, use {{tl|lang}}, and omit {{para
        dlist = 'name in local language; if more than one, separate ' ..
            'using {{tl|Plainlist}}, use {{tl|lang}}, and omit {{para
    },
    native_name_lang = {
        code = '<!-- language two- or three-letter ISO code -->',
        dlist = 'language two- or three-letter ISO code',
    },
    start_date = {
        code = '<!-- {{Start date|YYYY|MM|DD|df=y}} -->',
        dlist = 'use {{tlx|Start date|YYYY|MM|DD|df=y}}',
    },
    end_date = {
        code = '<!-- {{End date|YYYY|MM|DD|df=y}} -->',
        dlist = 'use {{tlx|Start date|YYYY|MM|DD|df=y}}',
    },
    url = {
        code = '<!-- use {{URL|example.com}} -->',
        dlist = 'using {{tl|URL}}',
    },
    website = {
        code = '<!-- use {{URL|example.com}} -->',
        dlist = 'using {{tls|URL|example.com}}',
    },
}

local p = {}
local removeDuplicates = require('Module:TableTools').removeDuplicates
local yesno = require('Module:Yesno')

local function makeInvokeFunction(funcName)
    return function(frame)
        local getArgs = require('Module:Arguments').getArgs
        return p[funcName](getArgs(frame, {removeBlanks = false}))
    end
end

local function extractParams(page)
    local source = mw.title.new(page, 'Template'):getContent()

    local parameters = {}
    for parameter in string.gmatch(source, '{{{(.-)%f[<>]}}') do
        table.insert(parameters, parameter)
    end
    return removeDuplicates(parameters)
end
```



```
end

local function map(tbl, transform)
    local returnTable = {}
    for k, v in pairs(tbl) do
        returnTable[k] = transform(v)
    end
    return returnTable
end

local function strMap(tbl, transform)
    local returnTable = map(tbl, transform)
    return table.concat(returnTable)
end

function p._check(args)
    local title = args.base or mw.title.getCurrentTitle().fullText
    return string.format(
        '{#{#invoke:Check for unknown parameters|check|unknown=' ..
        '[[Category:Pages using %s with unknown parameters]]|%s}}', title
        table.concat(extractParams(args.base), '|'))
end

function p._code(args)
    local definitions = yesno(args.definitions)
    local pad = yesno(args.pad)

    local parameters = extractParams(args.base)
    -- Space-pad the parameters to align the equal signs vertically
    if pad then
        local lengthPerPara = map(parameters, function (parameter)
            return string.len(parameter) end)
        -- Lua doesn't support printf's <*> to specify the width, appear
        local fs = string.format('%%-%ss', math.max(unpack(lengthPerPara)))
        for i, parameter in pairs(parameters) do
            parameters[i] = string.format(fs, parameter)
        end
    end

    local title = args.base or mw.title.getCurrentTitle().baseText
    return string.format([[ <nowiki>{{%s
%s}}</nowiki>]], title, strMap(parameters,
    function(s)
        if definitions then
            return string.format('| %s = %s\n', s,
                DEFINITIONS[s] and DEFINITIONS[s].code or
            else
                return string.format('| %s = \n', s)
            end
        end
    end))
end

function p._flatcode(args)
    local parameters = extractParams(args.base)
    local title = args.base or mw.title.getCurrentTitle().baseText
    return string.format(' {{tlp|%s%s}}', title, strMap(parameters,
        function(s)
            return string.format(' |%s{{=}}<var>%s</var>', s, s)
        end)
    )
end

function p._compare(args)
    local Set = require('Module:Set')
```

```
local function normaliseParams(parameters)
    local paramsNorm = {}
    -- Prepare a key lookup metatable, which will hold the original
    -- parameter names for each normalised parameter, e.g.
    -- [test] = {TEST, Test}. paramIndex functions like a Python
    -- defaultdict, where the default is a table.
    local paramIndex = setmetatable({}, {__index = function(t, k)
        if not rawget(t, k) then
            rawset(t, k, {})
        end
        return rawget(t, k)
    end})
    for _, parameter in pairs(parameters) do
        table.insert(paramsNorm,
            string.lower(string.gsub(parameter, '%A', '')))
        table.insert(paramIndex[
            string.lower(string.gsub(parameter, '%A', ''))],
            parameter)
    end

    paramsNorm = removeDuplicates(paramsNorm)
    -- Overload key lookup in paramsNorm. While [[Module:Set]] will
    -- operate on the table (which is to say, the normalised parameter
    -- array), key access will be by way of the paramIndex metatable.
    setmetatable(paramsNorm, {__index = paramIndex})
    return paramsNorm
end

local baseParams = extractParams(args.base)
local otherParams = extractParams(args.other)
local baseNormParams = normaliseParams(Set.valueComplement(
    otherParams, baseParams))
local otherNormParams = normaliseParams(otherParams)

return string.format([[Identical:
%s
Similar:
%s
Disparate:
%s]],
    strMap(Set.valueIntersection(baseParams, otherParams),
        function(v) return string.format('* %s\n', v) end),
    strMap(Set.valueIntersection(baseNormParams, otherNormParams),
        function(v) return string.format('* %s < %s [%s]\n',
            table.concat(baseNormParams[v], '; '),
            table.concat(otherNormParams[v], '; '),
            v)
        end),
    strMap(Set.valueComplement(otherNormParams, baseNormParams),
        function(v) return strMap(baseNormParams[v],
            function(s) return string.format('* %s\n', s) end)
        end))
end

function p._demo(args)
    local title = args.base and ('|_template=' .. args.base) or ''
    return string.format('{{Parameter names example%s|%s}}', title,
        table.concat(extractParams(args.base), '|'))
end

function p._dlist(args)
    local definitions = yesno(args.definitions, true)
    local defFormat = '; %s: %s\n'
    local nonDefFormat = '; %s: \n'
```



```
    if args._para then
        defFormat = '; {{para|%s}}: %s\n'
        nonDefFormat = '; {{para|%s}}: \n'
    end
    return strMap(extractParams(args.base),
        function(s)
            if definitions then
                return string.format(defFormat, s,
                    DEFINITIONS[s] and DEFINITIONS[s].dlist c
            else
                return string.format(nonDefFormat, s)
            end
        end)
end

function p._dlistpara(args)
    args._para = true
    return p._dlist(args)
end

function p._list(args)
    return strMap(extractParams(args.base),
        function(s) return string.format('* %s\n', s) end)
end

p.check = makeInvokeFunction('_check')
p.code = makeInvokeFunction('_code')
p.flatcode = makeInvokeFunction('_flatcode')
p.compare = makeInvokeFunction('_compare')
p.demo = makeInvokeFunction('_demo')
p.dlist = makeInvokeFunction('_dlist')
p.dlistpara = makeInvokeFunction('_dlistpara')
p.list = makeInvokeFunction('_list')

return p
```