



Inhaltsverzeichnis

1. Modul:Check for unknown parameters	2
2. Modul:Parameters	7



Modul:Check for unknown parameters

Vorlage:Lua

This module may be appended to a template to check for uses of unknown parameters.

Inhaltsverzeichnis	
1 Usage	2
1.1 Basic usage	2
1.2 Lua patterns	3
2 Example	3
3 See also	3

Usage

Basic usage

```

{{#invoke:check for unknown parameters|check
|unknown=[[Category:Some tracking category]]
|arg1|arg2|arg3|argN}}

```

or to sort the entries in the tracking category by parameter with a preview error message

```

{{#invoke:check for unknown parameters|check
|unknown=[[Category:Some tracking category|_VALUE_]]
|preview=unknown parameter "_VALUE_"
|arg1|arg2|...|argN}}

```

or for an explicit red error message

```

{{#invoke:check for unknown parameters|check
|unknown=<span class="error">Sorry, I don't recognize _VALUE_</span>
|arg1|arg2|...|argN}}

```

Here, arg1, arg2, ..., argN, are the known parameters. Unnamed (positional) parameters can be added too: |1|2|argname1|argname2|... Any parameter which is used, but not on this list, will cause the module to return whatever is passed with the unknown parameter. The _VALUE_ keyword, if used, will be changed to the name of the parameter. This is useful for either sorting the entries in a tracking category, or for provide more explicit information.

By default, the module makes no distinction between a defined-but-blank parameter and a non-blank parameter. That is, both unlisted **Vorlage:Para** and **Vorlage:Para** are reported. To only track non-blank parameters use **Vorlage:Para**.

By default, the module ignores blank positional parameters. That is, an unlisted **Vorlage:Para** is ignored. To *include* blank positional parameters in the tracking use **Vorlage:Para**.

Lua patterns

This module supports **Lua patterns** (similar to **regular expressions**), which are useful when there are many known parameters which use a systematic pattern. For example, **template:infobox3cols** uses

```
| regexp1 = header[%d][%d]*
| regexp2 = label[%d][%d]*
| regexp3 = data[%d][%d]*[abc]?
| regexp4 = class[%d][%d]*[abc]?
| regexp5 = rowclass[%d][%d]*
| regexp6 = rowstyle[%d][%d]*
| regexp7 = rowcellstyle[%d][%d]*
```

to match all parameters of the form headerNUM, labelNUM, dataNUM, dataNUMa, dataNUMb, dataNUMc, ..., rowcellstyleNUM, where NUM is a string of digits.

Example

```
{{Infobox
| above = {{{name|}}}

| label1 = Height
| data1 = {{{height|}}}

| label2 = Weight
| data2 = {{{weight|}}}

| label3 = Website
| data3 = {{{website|}}}
}}<!--
end infobox, start tracking
-->{{#invoke:Check for unknown parameters|check
| unknown = {{main other|[[Category:Some tracking category|_VALUE_]]}}
| preview = unknown parameter "_VALUE_"
| name
| height | weight
| website
}}
```

See also

- **Vorlage:Clc** (category page can have header **Vorlage:TI**)
- **Module:Check for deprecated parameters** – similar module that checks for deprecated parameters
- **Module:Check for clobbered parameters** – module that checks for conflicting parameters
- **Module:TemplatePar** – similar function (originally from dewiki)

- **Template:Parameters** and **Module:Parameters** - generates a list of parameter names for a given template
- **Project:TemplateData** based template parameter validation
- **Module:Parameter validation** checks a lot more
- **User:Bamyers99/TemplateParametersTool** - A tool for checking usage of template parameters

```
-- This module may be used to compare the arguments passed to the parent
-- with a list of arguments, returning a specified result if an argument is
-- not on the list
local p = {}

local function trim(s)
    return s:match('^%s*(.)%s*$')
end

local function isnotempty(s)
    return s and s:match('%S')
end

local function clean(text)
    -- Return text cleaned for display and truncated if too long.
    -- Strip markers are replaced with dummy text representing the original v
    local pos, truncated
    local function truncate(text)
        if truncated then
            return ''
        end
        if mw.ustring.len(text) > 25 then
            truncated = true
            text = mw.ustring.sub(text, 1, 25) .. '...'
        end
        return mw.text.nowiki(text)
    end
    local parts = {}
    for before, tag, remainder in text:gmatch('([^\127]*)\127[^\127]*-(%l+)%')
        pos = remainder
        table.insert(parts, truncate(before) .. '&lt;' .. tag .. '&gt;..'
    end
    table.insert(parts, truncate(text:sub(pos or 1)))
    return table.concat(parts)
end

function p._check(args, pargs)
    if type(args) ~= "table" or type(pargs) ~= "table" then
        -- TODO: error handling
        return
    end

    -- create the list of known args, regular expressions, and the return st
    local knownargs = {}
    local regexps = {}
    for k, v in pairs(args) do
        if type(k) == 'number' then
            v = trim(v)
            knownargs[v] = 1
        elseif k:find('^regexp[1-9][0-9]*$') then
            table.insert(regexps, '^' .. v .. '$')
        end
    end
end
```

```
-- loop over the parent args, and make sure they are on the list
local ignoreblank = isnotempty(args['ignoreblank'])
local showblankpos = isnotempty(args['showblankpositional'])
local values = {}
for k, v in pairs(pargs) do
    if type(k) == 'string' and knownargs[k] == nil then
        local knownflag = false
        for _, regexp in ipairs(regexps) do
            if mw.ustring.match(k, regexp) then
                knownflag = true
                break
            end
        end
        if not knownflag and ( not ignoreblank or isnotempty(v) ) then
            table.insert(values, clean(k))
        end
    elseif type(k) == 'number' and knownargs[tostring(k)] == nil then
        local knownflag = false
        for _, regexp in ipairs(regexps) do
            if mw.ustring.match(tostring(k), regexp) then
                knownflag = true
                break
            end
        end
        if not knownflag and ( showblankpos or isnotempty(v) ) then
            table.insert(values, k .. ' = ' .. clean(v))
        end
    end
end

-- add results to the output tables
local res = {}
if #values > 0 then
    local unknown_text = args['unknown'] or 'Found _VALUE_, '

    if mw.getCurrentFrame():preprocess( "{{REVISIONID}}" ) == "" then
        local preview_text = args['preview']
        if isnotempty(preview_text) then
            preview_text = require('Module:If preview')._warn
        elseif preview == nil then
            preview_text = unknown_text
        end
        unknown_text = preview_text
    end
    for _, v in pairs(values) do
        -- Fix odd bug for | = which gets stripped to the empty s
        -- breaks category links
        if v == '' then v = ' ' end

        -- avoid error with v = 'example%2' ("invalid capture in
        local r = unknown_text:gsub('_VALUE_', { _VALUE_ = v})
        table.insert(res, r)
    end
end

return table.concat(res)
end

function p.check(frame)
    local args = frame.args
    local pargs = frame:getParent().args
    return p._check(args, pargs)
end
```



```
return p
```

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
```