



Inhaltsverzeichnis

1. Modul:Navbar/Doku	2
2. Modul:Navbar	3
3. Modul:Navbox	6



Modul:Navbar/Doku

Dies ist die Dokumentationsseite für Modul:Navbar

Vorlage:Lua

This is a **Lua** implementation of **Vorlage:TI**. It is used in **Module:Navbar**.

Modul:Navbar

Vorlage:Lua

This is a [Lua](#) implementation of [Vorlage:TI](#). It is used in [Module:Navbar](#).

```
local p = {}
local cfg = mw.loadData('Module:Navbar/configuration')

local function get_title_arg(is_collapsible, template)
    local title_arg = 1
    if is_collapsible then title_arg = 2 end
    if template then title_arg = 'template' end
    return title_arg
end

local function choose_links(template, args)
    -- The show table indicates the default displayed items.
    -- view, talk, edit, hist, move, watch
    -- TODO: Move to configuration.
    local show = {true, true, true, false, false, false}
    if template then
        show[2] = false
        show[3] = false
        local index = {t = 2, d = 2, e = 3, h = 4, m = 5, w = 6,
            talk = 2, edit = 3, hist = 4, move = 5, watch = 6}
        -- TODO: Consider removing TableTools dependency.
        for _, v in ipairs(require('Module:TableTools').compressSparseArray)
            local num = index[v]
            if num then show[num] = true end
        end
    end

    local remove_edit_link = args.noedit
    if remove_edit_link then show[3] = false end

    return show
end

local function add_link(link_description, ul, is_mini, font_style)
    local l
    if link_description.url then
        l = {'[', '|', ']'}
    else
        l = {'[[', '|', ']]'}
    end
    end
    ul:tag('li')
        :addClass('nv-' .. link_description.full)
        :wikitext(l[1] .. link_description.link .. l[2])
        :tag(is_mini and 'abbr' or 'span')
            :attr('title', link_description.html_title)
            :cssText(font_style)
            :wikitext(is_mini and link_description.mini or link_description.full)
            :done()
        :wikitext(l[3])
        :done()
end
```

```
local function make_list(title_text, has_brackets, displayed_links, is_mini, font
    local title = mw.title.new(mw.text.trim(title_text), cfg.title_namespace)
    if not title then
        error(cfg.invalid_title .. title_text)
    end
    local talkpage = title.talkPageTitle and title.talkPageTitle.fullText or
-- TODO: Get link_descriptions and show into the configuration module.
-- link_descriptions should be easier...
    local link_descriptions = {
        { ['mini'] = 'v', ['full'] = 'view', ['html_title'] = 'View this
            ['link'] = title.fullText, ['url'] = false },
        { ['mini'] = 't', ['full'] = 'talk', ['html_title'] = 'Discuss th
            ['link'] = talkpage, ['url'] = false },
        { ['mini'] = 'e', ['full'] = 'edit', ['html_title'] = 'Edit this
            ['link'] = title:fullUrl('action=edit'), ['url'] = true }
        { ['mini'] = 'h', ['full'] = 'hist', ['html_title'] = 'History of
            ['link'] = title:fullUrl('action=history'), ['url'] = true
        { ['mini'] = 'm', ['full'] = 'move', ['html_title'] = 'Move this
            ['link'] = mw.title.new('Special:Movepage'):fullUrl('targ
        { ['mini'] = 'w', ['full'] = 'watch', ['html_title'] = 'Watch th
            ['link'] = title:fullUrl('action=watch'), ['url'] = true
    }

    local ul = mw.html.create('ul')
    if has_brackets then
        ul:addClass(cfg.classes.brackets)
        :cssText(font_style)
    end

    for i, _ in ipairs(displayed_links) do
        if displayed_links[i] then add_link(link_descriptions[i], ul, is_
    end
    return ul:done()
end

function p._navbar(args)

-- TODO: We probably don't need both fontstyle and fontcolor...
    local font_style = args.fontstyle
    local font_color = args.fontcolor
    local is_collapsible = args.collapsible
    local is_mini = args.mini
    local is_plain = args.plain

    local collapsible_class = nil
    if is_collapsible then
        collapsible_class = cfg.classes.collapsible
        if not is_plain then is_mini = 1 end
        if font_color then
            font_style = (font_style or '') .. '; color: ' .. font_co
        end
    end

    local navbar_style = args.style
    local div = mw.html.create():tag('div')
    div
        :addClass(cfg.classes.navbar)
        :addClass(cfg.classes.plainlinks)
        :addClass(cfg.classes.horizontal_list)
        :addClass(collapsible_class) -- we made the determination earlier
        :cssText(navbar_style)
```

```
if is_mini then div:addClass(cfg.classes.mini) end

local box_text = (args.text or cfg.box_text) .. ' '
-- the concatenated space guarantees the box text is separated
if not (is_mini or is_plain) then
    div
        :tag('span')
            :addClass(cfg.classes.box_text)
            :cssText(font_style)
            :wikitext(box_text)
    end

local template = args.template
local displayed_links = choose_links(template, args)
local has_brackets = args.brackets
local title_arg = get_title_arg(is_collapsible, template)
local title_text = args[title_arg] or (':' .. mw.getCurrentFrame():getTitle())
local list = make_list(title_text, has_brackets, displayed_links, is_mini)
div:node(list)

if is_collapsible then
    local title_text_class
    if is_mini then
        title_text_class = cfg.classes.collapsible_title_mini
    else
        title_text_class = cfg.classes.collapsible_title_full
    end
    div:done()
        :tag('div')
            :addClass(title_text_class)
            :cssText(font_style)
            :wikitext(args[1])
    end

return mw.getCurrentFrame():extensionTag{
    name = 'templatestyles', args = { src = cfg.templatestyles }
} .. tostring(div:done())
end

function p.navbar(frame)
    return p._navbar(require('Module:Arguments').getArgs(frame))
end

return p
```

Modul:Navbar

Vorlage:Lua Vorlage:Uses TemplateStyles Vorlage:Lua sidebar

This module implements the [Vorlage:TI](#) template. Please see the [template page](#) for usage instructions.

Tracking/maintenance categories

- [Vorlage:Clc](#)
- [Vorlage:Clc](#)
- [Vorlage:Clc](#)
- [Vorlage:Clc](#)
- [Vorlage:Clc](#)

```
local p = {}
local navbar = require('Module:Navbar')._navbar
local cfg = mw.loadData('Module:Navbar/configuration')
local getArgs -- lazily initialized
local args
local format = string.format

local function striped(wikitext, border)
    -- Return wikitext with markers replaced for odd/even striping.
    -- Child (subgroup) navboxes are flagged with a category that is removed
    -- by parent navboxes. The result is that the category shows all pages
    -- where a child navbox is not contained in a parent navbox.
    local orphanCat = cfg.category.orphan
    if border == cfg.keyword.border_subgroup and args[cfg.arg.orphan] ~= cfg
        -- No change; striping occurs in outermost navbox.
        return wikitext .. orphanCat
    end
    local first, second = cfg.class.navbox_odd_part, cfg.class.navbox_even_part
    if args[cfg.arg.evenodd] then
        if args[cfg.arg.evenodd] == cfg.keyword.evenodd_swap then
            first, second = second, first
        else
            first = args[cfg.arg.evenodd]
            second = first
        end
    end
    local changer
    if first == second then
        changer = first
    else
        local index = 0
        changer = function (code)
            if code == '0' then
                -- Current occurrence is for a group before a new
                -- Set it to first as a valid although pointless
                -- The next occurrence will be the first row after
                -- in a subgroup and will also be first.
                index = 0
                return first
            end
        end
    end
end
```

```

        index = index + 1
        return index % 2 == 1 and first or second
    end
end
local regex = orphanCat:gsub('([%[%]])', '%%%1')
return (wikitext:gsub(regex, '):gsub(cfg.marker.regex, changer)) -- ()
end

local function processItem(item, nowrapitems)
    if item:sub(1, 2) == '{|}' then
        -- Applying nowrap to lines in a table does not make sense.
        -- Add newlines to compensate for trim of x in |parm=x in a temp
        return '\n' .. item .. '\n'
    end
    if nowrapitems == cfg.keyword.nowrapitems_yes then
        local lines = {}
        for line in (item .. '\n'):gmatch('([^\n]*)\n') do
            local prefix, content = line:match('^([*:#]+)%s*(.*)')
            if prefix and not content:match(cfg.pattern.nowrap) then
                line = format(cfg.nowrap_item, prefix, content)
            end
            table.insert(lines, line)
        end
        item = table.concat(lines, '\n')
    end
    if item:match('^[*:#]') then
        return '\n' .. item .. '\n'
    end
    return item
end

end

-- we will want this later when we want to add tstyles for hlist/plainlist
local function has_navbar()
    return args[cfg.arg.navbar] ~= cfg.keyword.navbar_off
        and args[cfg.arg.navbar] ~= cfg.keyword.navbar_plain
        and (
            args[cfg.arg.name]
            or mw.getCurrentFrame():getParent():getTitle():gsub(cfg.p
                ~= cfg.pattern.navbox
        )
end

local function renderNavBar(titleCell)
    if has_navbar() then
        titleCell:wikitext(navbar{
            [cfg.navbar.name] = args[cfg.arg.name],
            [cfg.navbar.mini] = 1,
            [cfg.navbar.fontstyle] = (args[cfg.arg.basestyle] or '')
                (args[cfg.arg.titlestyle] or '') ..
                ';background:none transparent;border:none;box-sha
        })
    end
end

end

local function renderTitleRow(tbl)
    if not args[cfg.arg.title] then return end

    local titleRow = tbl:tag('tr')

    local titleCell = titleRow:tag('th'):attr('scope', 'col')

    local titleColspan = 2
    if args[cfg.arg.imageleft] then titleColspan = titleColspan + 1 end
end
```

```
        if args[cfg.arg.image] then titleColspan = titleColspan + 1 end
    titleCell
        :cssText(args[cfg.arg.basestyle])
        :cssText(args[cfg.arg.titlestyle])
        :addClass(cfg.class.navbox_title)
        :attr('colspan', titleColspan)

    renderNavBar(titleCell)

    titleCell
        :tag('div')
            -- id for aria-labelledby attribute
            :attr('id', mw.uri.anchorEncode(args[cfg.arg.title]))
            :addClass(args[cfg.arg.titleclass])
            :css('font-size', '114%')
            :css('margin', '0 4em')
            :wikitext(processItem(args[cfg.arg.title]))
end

local function getAboveBelowColspan()
    local ret = 2
    if args[cfg.arg.imageleft] then ret = ret + 1 end
    if args[cfg.arg.image] then ret = ret + 1 end
    return ret
end

local function renderAboveRow(tbl)
    if not args[cfg.arg.above] then return end

    tbl:tag('tr')
        :tag('td')
            :addClass(cfg.class.navbox_abovebelow)
            :addClass(args[cfg.arg.aboveclass])
            :cssText(args[cfg.arg.basestyle])
            :cssText(args[cfg.arg.abovestyle])
            :attr('colspan', getAboveBelowColspan())
            :tag('div')
                -- id for aria-labelledby attribute, if no title
                :attr('id', args[cfg.arg.title] and nil or mw.uri.anchorEncode(args[cfg.arg.title]))
                :wikitext(processItem(args[cfg.arg.above], args[cfg.arg.title]))
end

local function renderBelowRow(tbl)
    if not args[cfg.arg.below] then return end

    tbl:tag('tr')
        :tag('td')
            :addClass(cfg.class.navbox_abovebelow)
            :addClass(args[cfg.arg.belowclass])
            :cssText(args[cfg.arg.basestyle])
            :cssText(args[cfg.arg.belowstyle])
            :attr('colspan', getAboveBelowColspan())
            :tag('div')
                :wikitext(processItem(args[cfg.arg.below], args[cfg.arg.title]))
end

local function renderListRow(tbl, index, listnum, listnums_size)
    local row = tbl:tag('tr')

    if index == 1 and args[cfg.arg.imageleft] then
        row
            :tag('td')
                :addClass(cfg.class.noviewer)
    end
end
```

```
                :addClass(cfg.class.navbox_image)
                :addClass(args[cfg.arg.imageclass])
                :css('width', '1px') -- Minimize width
                :css('padding', '0 2px 0 0')
                :cssText(args[cfg.arg.imageleftstyle])
                :attr('rowspan', listnums_size)
                :tag('div')
                :wikitext(processItem(args[cfg.arg.image]))
end

local group_and_num = format(cfg.arg.group_and_num, listnum)
local groupstyle_and_num = format(cfg.arg.groupstyle_and_num, listnum)
if args[group_and_num] then
    local groupCell = row:tag('th')

    -- id for aria-labelledby attribute, if lone group with no title
    if listnum == 1 and not (args[cfg.arg.title] or args[cfg.arg.above])
        groupCell
            :attr('id', mw.uri.anchorEncode(args[cfg.arg.group_and_num]))
    end

    groupCell
        :attr('scope', 'row')
        :addClass(cfg.class.navbox_group)
        :addClass(args[cfg.arg.groupclass])
        :cssText(args[cfg.arg.basestyle])
        -- If groupwidth not specified, minimize width
        :css('width', args[cfg.arg.groupwidth] or '1%')

    groupCell
        :cssText(args[cfg.arg.groupstyle])
        :cssText(args[groupstyle_and_num])
        :wikitext(args[group_and_num])
end

local listCell = row:tag('td')

if args[group_and_num] then
    listCell
        :addClass(cfg.class.navbox_list_with_group)
else
    listCell:attr('colspan', 2)
end

if not args[cfg.arg.groupwidth] then
    listCell:css('width', '100%')
end

local rowstyle -- usually nil so cssText(rowstyle) usually adds nothing
if index % 2 == 1 then
    rowstyle = args[cfg.arg.oddstyle]
else
    rowstyle = args[cfg.arg.evenstyle]
end

local list_and_num = format(cfg.arg.list_and_num, listnum)
local listText = args[list_and_num]
local oddEven = cfg.marker.oddeven
if listText:sub(1, 12) == '</div><table' then
    -- Assume list text is for a subgroup navbar so no automatic stripping
    oddEven = listText:find(cfg.pattern.navbox_title) and cfg.marker
end

local liststyle_and_num = format(cfg.arg.liststyle_and_num, listnum)
```

```
local listclass_and_num = format(cfg.arg.listclass_and_num, listnum)
listCell
    :css('padding', '0')
    :cssText(args[cfg.arg.liststyle])
    :cssText(rowstyle)
    :cssText(args[liststyle_and_num])
    :addClass(cfg.class.navbox_list)
    :addClass(cfg.class.navbox_part .. oddEven)
    :addClass(args[cfg.arg.listclass])
    :addClass(args[listclass_and_num])
    :tag('div')
        :css('padding',
            (index == 1 and args[cfg.arg.listlpadding]) or a
        )
        :wikitext(processItem(listText, args[cfg.arg.nowrapitems]
end
if index == 1 and args[cfg.arg.image] then
    row
        :tag('td')
            :addClass(cfg.class.noviewer)
            :addClass(cfg.class.navbox_image)
            :addClass(args[cfg.arg.imageclass])
            :css('width', '1px') -- Minimize width
            :css('padding', '0 0 0 2px')
            :cssText(args[cfg.arg.imagestyle])
            :attr('rowspan', listnums_size)
            :tag('div')
                :wikitext(processItem(args[cfg.arg.image]
        end
    end
end

-- uses this now to make the needHlistCategory correct
-- to use later for when we add list styles via navbar
local function has_list_class(htmlclass)
    local class_args = { -- rough order of probability of use
        cfg.arg.bodyclass, cfg.arg.listclass, cfg.arg.aboveclass,
        cfg.arg.belowclass, cfg.arg.titleclass, cfg.arg.navboxclass,
        cfg.arg.groupclass, cfg.arg.imageclass
    }
    local patterns = {
        '^' .. htmlclass .. '$',
        '%s' .. htmlclass .. '$',
        '^' .. htmlclass .. '%s',
        '%s' .. htmlclass .. '%s'
    }
    for _, arg in ipairs(class_args) do
        for _, pattern in ipairs(patterns) do
            if mw.ustext.find(args[arg] or '', pattern) then
                return true
            end
        end
    end
    return false
end

local function needsHorizontalLists(border)
    if border == cfg.keyword.border_subgroup or args[cfg.arg.tracking] == cfg
    then
        return false
    end
    return not has_list_class(cfg.pattern.hlist) and not has_list_class(cfg.p
end

local function hasBackgroundColors()
```

```
        for _, key in ipairs({cfg.arg.titlestyle, cfg.arg.groupstyle,
            cfg.arg.basestyle, cfg.arg.abovestyle, cfg.arg.belowstyle}) do
            if tostring(args[key]):find('background', 1, true) then
                return true
            end
        end
    end
    return false
end

local function hasBorders()
    for _, key in ipairs({cfg.arg.groupstyle, cfg.arg.basestyle,
        cfg.arg.abovestyle, cfg.arg.belowstyle}) do
        if tostring(args[key]):find('border', 1, true) then
            return true
        end
    end
    return false
end

local function isIllegible()
    local styleratio = require('Module:Color contrast')._styleratio
    for key, style in pairs(args) do
        if tostring(key):match(cfg.pattern.style) then
            if styleratio{mw.text.unstripNowiki(style)} < 4.5 then
                return true
            end
        end
    end
    return false
end

local function getTrackingCategories(border)
    local cats = {}
    if needsHorizontalLists(border) then table.insert(cats, cfg.category.horizontal)
    if hasBackgroundColors() then table.insert(cats, cfg.category.background)
    if isIllegible() then table.insert(cats, cfg.category.illegible) end
    if hasBorders() then table.insert(cats, cfg.category.borders) end
    return cats
end

local function renderTrackingCategories(builder, border)
    local title = mw.title.getCurrentTitle()
    if title.namespace ~= 10 then return end -- not in template space
    local subpage = title.subpageText
    if subpage == cfg.keyword.subpage_doc or subpage == cfg.keyword.subpage_s
        or subpage == cfg.keyword.subpage_testcases then return end

    for _, cat in ipairs(getTrackingCategories(border)) do
        builder:wikitext('[[Category:' .. cat .. ']]')
    end
end

local function renderMainTable(border, listnums)
    local tbl = mw.html.create('table')
        :addClass(cfg.class.nowraplinks)
        :addClass(args[cfg.arg.bodyclass])

    local state = args[cfg.arg.state]
    if args[cfg.arg.title] and state ~= cfg.keyword.state_plain and state ~=
        if state == cfg.keyword.state_collapsed then
            state = cfg.class.collapsed
        end
    tbl
        :addClass(cfg.class.collapsible)
```

```

        :addClass(state or cfg.class.autocollapse)
    end

tbl:css('border-spacing', 0)
if border == cfg.keyword.border_subgroup or border == cfg.keyword.border_
tbl
        :addClass(cfg.class.navbox_subgroup)
        :cssText(args[cfg.arg.bodystyle])
        :cssText(args[cfg.arg.style])
else -- regular navbox - bodystyle and style will be applied to the wrap
tbl
        :addClass(cfg.class.navbox_inner)
        :css('background', 'transparent')
        :css('color', 'inherit')
end
tbl:cssText(args[cfg.arg.innerstyle])

renderTitleRow(tbl)
renderAboveRow(tbl)
local listnums_size = #listnums
for i, listnum in ipairs(listnums) do
    renderListRow(tbl, i, listnum, listnums_size)
end
renderBelowRow(tbl)

return tbl
end

local function add_navbox_styles()
    local frame = mw.getCurrentFrame()
    -- This is a lambda so that it doesn't need the frame as a parameter
    local function add_user_styles(templatestyles)
        if templatestyles and templatestyles ~= '' then
            return frame:extensionTag{
                name = 'templatestyles', args = { src = templatestyles }
            }
        end
        return ''
    end
end

-- get templatestyles. load base from config so that Lua only needs to do
-- the work once of parser tag expansion
local base_templatestyles = cfg.templatestyles
local templatestyles = add_user_styles(args[cfg.arg.templatestyles])
local child_templatestyles = add_user_styles(args[cfg.arg.child_templates])

-- The 'navbox-styles' div exists for two reasons:
-- 1. To wrap the styles to work around T200206 more elegantly. Instead
-- of combinatorial rules, this ends up being linear number of CSS ru
-- 2. To allow MobileFrontend to rip the styles out with 'nomobile' such
-- they are not dumped into the mobile view.
return mw.html.create('div')
        :addClass(cfg.class.navbox_styles)
        :addClass(cfg.class.nomobile)
        :wikitext(base_templatestyles .. templatestyles .. child_templatestyles)
        :done()
end

function p._navbox(navboxArgs)
    args = navboxArgs
    local listnums = {}

    for k, _ in pairs(args) do
        if type(k) == 'string' then

```

```
                local listnum = k:match(cfg.pattern.listnum)
                if listnum then table.insert(listnums, tonumber(listnum))
            end
        end
    end
    table.sort(listnums)

    local border = mw.text.trim(args[cfg.arg.border] or args[1] or '')
    if border == cfg.keyword.border_child then
        border = cfg.keyword.border_subgroup
    end

    -- render the main body of the navbox
    local tbl = renderMainTable(border, listnums)

    local res = mw.html.create()
    -- render the appropriate wrapper for the navbox, based on the border parameter

    if border == cfg.keyword.border_none then
        res:node(add_navbox_styles())
        local nav = res:tag('div')
            :attr('role', 'navigation')
            :node(tbl)
        -- aria-labelledby title, otherwise above, otherwise lone group
        if args[cfg.arg.title] or args[cfg.arg.above] or (args[cfg.arg.group1]
            and not args[cfg.arg.group2]) then
            nav:attr(
                'aria-labelledby',
                mw.uri.anchorEncode(
                    args[cfg.arg.title] or args[cfg.arg.above]
                )
            )
        else
            nav:attr('aria-label', cfg.aria_label)
        end
    elseif border == cfg.keyword.border_subgroup then
        -- We assume that this navbox is being rendered in a list cell of
        -- parent navbox, and is therefore inside a div with padding:0em
        -- We start with a </div> to avoid the padding being applied, and
        -- end add a <div> to balance out the parent's </div>
        res
            :wikitext('</div>')
            :node(tbl)
            :wikitext('<div>')
    else
        res:node(add_navbox_styles())
        local nav = res:tag('div')
            :attr('role', 'navigation')
            :addClass(cfg.class.navbox)
            :addClass(args[cfg.arg.navboxclass])
            :cssText(args[cfg.arg.bodystyle])
            :cssText(args[cfg.arg.style])
            :css('padding', '3px')
            :node(tbl)
        -- aria-labelledby title, otherwise above, otherwise lone group
        if args[cfg.arg.title] or args[cfg.arg.above]
            or (args[cfg.arg.group1] and not args[cfg.arg.group2]) then
            nav:attr(
                'aria-labelledby',
                mw.uri.anchorEncode(args[cfg.arg.title] or args[
                    cfg.arg.above
                ])
            )
        else
            nav:attr('aria-label', cfg.aria_label)
        end
    end
end
```

```
        if (args[cfg.arg.nocat] or cfg.keyword.nocat_false):lower() == cfg.keyword
            renderTrackingCategories(res, border)
        end
        return striped(tostring(res), border)
    end

function p.navbox(frame)
    if not getArgs then
        getArgs = require('Module:Arguments').getArgs
    end
    args = getArgs(frame, {wrappers = {cfg.pattern.navbox}})

    -- Read the arguments in the order they'll be output in, to make referenc
    -- number in the right order.
    local
        _ = args[cfg.arg.title]
        _ = args[cfg.arg.above]
    -- Limit this to 20 as covering 'most' cases (that's a SWAG) and because
    -- iterator approach won't work here
    for i = 1, 20 do
        _ = args[format(cfg.arg.group_and_num, i)]
        _ = args[format(cfg.arg.list_and_num, i)]
    end
    _ = args[cfg.arg.below]

    return p._navbox(args)
end

return p
```