پودمان:Navbox

.□□□□ □□□□ [□□□□□□□□/Navbox:□□□□□□□](#) □□ □□□□□□□□ □□□□□□□ □□□□ □□□□□□□□

```
require('strict')
local p = {}
local cfg = mw.loadData('Module:Navbox/configuration')
local inArray = require("Module:TableTools").inArray
local numConv = require('Module:Numeral converter').convert
local getArgs -- lazily initialized
local hiding_templatestyles = {}

-- global passthrough variables
local passthrough = {
[cfg.arg.above]=true,[cfg.arg.aboveclass]=true,[cfg.arg.abovestyle]=true,
        [cfg.arg.basestyle]=true,
[cfg.arg.below]=true,[cfg.arg.belowclass]=true,[cfg.arg.belowstyle]=true,
        [cfg.arg.bodyclass]=true,
        [cfg.arg.groupclass]=true,
[cfg.arg.image]=true,[cfg.arg.imageclass]=true,[cfg.arg.imagestyle]=true,
        [cfg.arg.imageleft]=true,[cfg.arg.imageleftstyle]=true,
        [cfg.arg.listclass]=true,
        [cfg.arg.name]=true,
        [cfg.arg.navbar]=true,
        [cfg.arg.state]=true,
[cfg.arg.title]=true,[cfg.arg.titleclass]=true,[cfg.arg.titlestyle]=true,
        argHash=true
}

-- helper functions
local andnum = function(s, n) return string.format(cfg.arg[s .. '_and_num'],
n) end
local isblank = function(v) return (v or '') == '' end

local function concatstrings(s)
        local r = table.concat(s, '')
        if r:match('^%s*$') then return nil end
        return r
end

local function concatstyles(s)
        local r = ''
        for _, v in ipairs(s) do
                v = mw.text.trim(v, "%s;")
                if not isblank(v) then r = r .. v .. ';' end
        end
```

```
        if isblank(r) then return nil end
        return r
end

local function getSubgroup(args, listnum, listText, prefix)
        local subArgs = {
                [cfg.arg.border] = cfg.keyword.border_subgroup,
                [cfg.arg.navbar] = cfg.keyword.navbar_plain,
                argHash = 0
        }
        local hasSubArgs = false
        local subgroups_and_num = prefix and {prefix} or
cfg.arg.subgroups_and_num
        for k, v in pairs(args) do
                k = tostring(k)
                for _, w in ipairs(subgroups_and_num) do
                        w = string.format(w, listnum) .. "_"
                        if (#k > #w) and (k:sub(1, #w) == w) then
                                subArgs[k:sub(#w + 1)] = v
                                hasSubArgs = true
                                subArgs.argHash = subArgs.argHash + (v and #v
or 0)
                        end
                end
        end
        return hasSubArgs and p._navbox(subArgs) or listText
end

-- Main functions
function p._navbox(args)
        if args.type == cfg.keyword.with_collapsible_groups then
                return p._withCollapsibleGroups(args)
        elseif args.type == cfg.keyword.with_columns then
                return p._withColumns(args)
        end

        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.keyword.orphan_yes then
                        -- 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 nested table.
                                        -- Set it to first as a valid
although pointless class.
                                        -- The next occurrence will be the
first row after a title
                                        -- in a subgroup and will also be
first.
                                        index = 0
                                        return first
                                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)) -- () omits gsub count
        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 template.
                        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 = string.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

        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.pattern.sandbox, '')
                                        ~= cfg.pattern.navbox
                        )
        end

        -- extract text color from css, which is the only permitted inline
CSS for the navbar
        local function extract_color(css_str)
                -- return nil because navbar takes its argument into mw.html
which handles
                -- nil gracefully, removing the associated style attribute
                return mw.ustring.match(';' .. css_str .. ';',
'.*;%s*([Cc][Oo][Ll][Oo][Rr]%s*:%s*.-)%s*;') or nil
        end

        local function renderNavBar(titleCell)
                if has_navbar() then
                        local navbar = require('Module:Navbar')._navbar
                        titleCell:wikitext(navbar{
                                [cfg.navbar.name] = args[cfg.arg.name],
                                [cfg.navbar.mini] = 1,
                                [cfg.navbar.fontstyle] = extract_color(
                                        (args[cfg.arg.basestyle] or '') ..
';' .. (args[cfg.arg.titlestyle] or '')
                                )
                        })
                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
                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]) .. args.argHash)
                                :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', (not args[cfg.arg.title])
and
(mw.uri.anchorEncode(args[cfg.arg.above]) .. args.argHash)
                                                or nil)
:wikitext(processItem(args[cfg.arg.above], args[cfg.arg.nowrapitems]))
        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.nowrapitems]))
        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)
                                        :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.imageleft]))
                end

                local group_and_num = andnum('group', listnum)
                local groupstyle_and_num = andnum('groupstyle', listnum)
                if args[group_and_num] then
                        local groupCell = row:tag('th')

                        -- id for aria-labelledby attribute, if lone group
with no title or above
                        if listnum == 1 and not (args[cfg.arg.title] or
args[cfg.arg.above] or args[cfg.arg.group2]) then
                                groupCell
                                        :attr('id',
mw.uri.anchorEncode(args[cfg.arg.group1]) .. args.argHash)
```

```
                        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 = andnum('list', listnum)
                local listText = inArray(cfg.keyword.subgroups,
args[list_and_num])
                        and getSubgroup(args, listnum, args[list_and_num]) or
args[list_and_num]

                local oddEven = cfg.marker.oddeven
                if listText:sub(1, 12) == '</div><table' then
                        -- Assume list text is for a subgroup navbox so no
automatic striping for this row.
                        oddEven = listText:find(cfg.pattern.navbox_title) and
cfg.marker.restart or cfg.class.navbox_odd_part
```

```
                end

                local liststyle_and_num = andnum('liststyle', listnum)
                local listclass_and_num = andnum('listclass', 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.list1padding]) or args[cfg.arg.listpadding] or '0 0.25em'
                                )
                                :wikitext(processItem(listText,
args[cfg.arg.nowrapitems]))

                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

        local function has_list_class(htmlclass)
                local patterns = {
                        '^' .. htmlclass .. '$',
                        '%s' .. htmlclass .. '$',
                        '^' .. htmlclass .. '%s',
                        '%s' .. htmlclass .. '%s'
                }

                for arg, _ in pairs(args) do
                        if type(arg) == 'string' and mw.ustring.find(arg,
cfg.pattern.class) then
                                for _, pattern in ipairs(patterns) do
                                        if mw.ustring.find(args[arg] or '',
pattern) then
```

```
                                                return true
                                        end
                                end
                        end
                end
                return false
        end

        -- there are a lot of list classes in the wild, so we add their
TemplateStyles
        local function add_list_styles()
                local frame = mw.getCurrentFrame()
                local function add_list_templatestyles(htmlclass,
templatestyles)
                        if has_list_class(htmlclass) then
                                return frame:extensionTag{
                                        name = 'templatestyles', args = { src
= templatestyles }
                                }
                        else
                                return ''
                        end
                end

                local hlist_styles = add_list_templatestyles('hlist',
cfg.hlist_templatestyles)
                local plainlist_styles = add_list_templatestyles('plainlist',
cfg.plainlist_templatestyles)

                -- a second workaround for [[phab:T303378]]
                -- when that issue is fixed, we can actually use has_navbar
not to emit the
                -- tag here if we want
                if has_navbar() and hlist_styles == '' then
                        hlist_styles = frame:extensionTag{
                                name = 'templatestyles', args = { src =
cfg.hlist_templatestyles }
                        }
                end

                -- hlist -> plainlist is best-effort to preserve old
Common.css ordering.
                -- this ordering is not a guarantee because most navboxes
will emit only
                -- one of these classes [hlist_note]
                return hlist_styles .. plainlist_styles
        end

        local function needsHorizontalLists(border)
                if border == cfg.keyword.border_subgroup or
args[cfg.arg.tracking] == cfg.keyword.tracking_no then
```

```
                        return false
                end
                return not has_list_class(cfg.pattern.hlist) and not
has_list_class(cfg.pattern.plainlist)
        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
                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_lists) end
                if hasBackgroundColors() then table.insert(cats,
cfg.category.background_colors) end
                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_sandbox
                        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 ~= cfg.keyword.state_off then
                        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_none then
                        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 wrapper table
                        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(hiding_templatestyles)
                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 not isblank(templatestyles) then
                                return frame:extensionTag{
                                        name = 'templatestyles', args = { src
= templatestyles }
                                }
                        end
                        return ''
                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_templatestyles])

                -- The 'navbox-styles' div exists to wrap the styles to work
around T200206
                -- more elegantly. Instead of combinatorial rules, this ends
up being linear
                -- number of CSS rules.
                return mw.html.create('div')
                        :addClass(cfg.class.navbox_styles)
                        :wikitext(
                                add_list_styles() .. -- see [hlist_note]
applied to 'before base_templatestyles'
                                base_templatestyles ..
                                templatestyles ..
                                child_templatestyles ..
                                table.concat(hiding_templatestyles)
                        )
                        :done()
        end
```

```
        -- work around [[phab:T303378]]
        -- for each arg: find all the templatestyles strip markers, insert
them into a
        -- table. then remove all templatestyles markers from the arg
        local strip_marker_pattern = '(\127[^\127]*UNIQ%-%-templatestyles%-
%x+%-QINU[^\127]*\127)'
        local argHash = 0
        for k, arg in pairs(args) do
                if type(arg) == 'string' then
                        for marker in string.gfind(arg, strip_marker_pattern)
do
                                table.insert(hiding_templatestyles, marker)
                        end
                        argHash = argHash + #arg
                        args[k] = string.gsub(arg, strip_marker_pattern, '')
                end
        end
        if not args.argHash then args.argHash = argHash end

        local listnums = {}

        for k, _ in pairs(args) do
                if type(k) == 'string' then
                        local listnum = k:match(cfg.pattern.listnum)
                        if listnum and args[andnum('list',
tonumber(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
param

        if border == cfg.keyword.border_none then
                res:node(add_navbox_styles(hiding_templatestyles))
                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] or args[cfg.arg.group1]
                                ) .. args.argHash
                        )
                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 a
                -- parent navbox, and is therefore inside a div with
padding:0em 0.25em.
                -- We start with a </div> to avoid the padding being applied,
and at the
                -- end add a <div> to balance out the parent's </div>
                res
                        :wikitext('</div>')
                        :node(tbl)
                        :wikitext('<div>')
        else
                res:node(add_navbox_styles(hiding_templatestyles))
                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] or args[cfg.arg.group1]
                                ) .. args.argHash
                        )
                else
                        nav:attr('aria-label', cfg.aria_label ..
args.argHash)
                end
```

```
        end

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

function p._withCollapsibleGroups(pargs)
        -- table for args passed to navbox
        local targs = {}

        -- process args
        local passthroughLocal = {
                [cfg.arg.bodystyle] = true,
                [cfg.arg.border] = true,
                [cfg.arg.style] = true,
        }
        for k,v in pairs(pargs) do
                if k and type(k) == 'string' then
                        if passthrough[k] or passthroughLocal[k] then
                                targs[k] = v
                        elseif (k:match(cfg.pattern.num)) then
                                local n = k:match(cfg.pattern.num)
                                local list_and_num = andnum('list', n)
                                if ((k:match(cfg.pattern.listnum) or
k:match(cfg.pattern.contentnum))
                                                and targs[list_and_num] ==
nil
                                                and pargs[andnum('group', n)]
== nil
                                                and pargs[andnum('sect', n)]
== nil
                                                and pargs[andnum('section',
n)] == nil) then
                                        targs[list_and_num] = concatstrings({
                                                pargs[list_and_num] or '',
                                                pargs[andnum('content', n)]
or ''
                                        })
                                        if (targs[list_and_num] and
inArray(cfg.keyword.subgroups, targs[list_and_num])) then
                                                targs[list_and_num] =
getSubgroup(pargs, n, targs[list_and_num])
                                        end
                                elseif ((k:match(cfg.pattern.groupnum) or
k:match(cfg.pattern.sectnum) or k:match(cfg.pattern.sectionnum))
                                                and targs[list_and_num] ==
nil) then
                                        local titlestyle = concatstyles({
```

```
                                                pargs[cfg.arg.groupstyle] or
'',
                                                pargs[cfg.arg.secttitlestyle]
or '',
                                                pargs[andnum('groupstyle',
n)] or '',
pargs[andnum('sectiontitlestyle', n)] or ''
                                        })
                                        local liststyle = concatstyles({
                                                pargs[cfg.arg.liststyle] or
'',
                                                pargs[cfg.arg.contentstyle]
or '',
                                                pargs[andnum('liststyle', n)]
or '',
                                                pargs[andnum('contentstyle',
n)] or ''
                                        })
                                        local title = concatstrings({
                                                pargs[andnum('group', n)] or
'',
                                                pargs[andnum('sect', n)] or
'',
                                                pargs[andnum('section', n)]
or ''
                                        })
                                        local list = concatstrings({
                                                pargs[list_and_num] or '',
                                                pargs[andnum('content', n)]
or ''
                                        })
                                        if list and
inArray(cfg.keyword.subgroups, list) then
                                                list = getSubgroup(pargs, n,
list)
                                        end
                                        local abbr_and_num = andnum('abbr',
n)
                                        local state = (pargs[abbr_and_num]
and pargs[abbr_and_num] == pargs[cfg.arg.selected])
                                                and
cfg.keyword.state_uncollapsed
                                                or (pargs[andnum('state', n)]
or cfg.keyword.state_collapsed)
                                        targs[list_and_num] =p._navbox({
                                                cfg.keyword.border_child,
                                                [cfg.arg.navbar] =
cfg.keyword.navbar_plain,
                                                [cfg.arg.state] = state,
                                                [cfg.arg.basestyle] =
pargs[cfg.arg.basestyle],
```

```
                                                [cfg.arg.title] = title,
                                                [cfg.arg.titlestyle] =
titlestyle,
                                                [andnum('list', 1)] = list,
                                                [cfg.arg.liststyle] =
liststyle,
                                                [cfg.arg.listclass] =
pargs[andnum('listclass', n)],
                                                [cfg.arg.image] =
pargs[andnum('image', n)],
                                                [cfg.arg.imageleft] =
pargs[andnum('imageleft', n)],
                                                [cfg.arg.listpadding] =
pargs[cfg.arg.listpadding],
                                                argHash = pargs.argHash
                                        })
                                end
                        end
                end
        end
        -- ordering of style and bodystyle
        targs[cfg.arg.style] = concatstyles({targs[cfg.arg.style] or '',
targs[cfg.arg.bodystyle] or ''})
        targs[cfg.arg.bodystyle] = nil

        -- child or subgroup
        if targs[cfg.arg.border] == nil then targs[cfg.arg.border] = pargs[1]
end

        return p._navbox(targs)
end --p._withCollapsibleGroups

function p._withColumns(pargs)
        -- table for args passed to navbox
        local targs = {}

        -- tables of column numbers
        local colheadernums = {}
        local colnums = {}
        local colfooternums = {}

        -- process args
        local passthroughLocal = {
                [cfg.arg.evenstyle]=true,
                [cfg.arg.groupstyle]=true,
                [cfg.arg.liststyle]=true,
                [cfg.arg.oddstyle]=true,
                [cfg.arg.state]=true,
        }
        for k,v in pairs(pargs) do
                if passthrough[k] or passthroughLocal[k] then
```

```
                        targs[k] = v
                elseif type(k) == 'string' then
                        if k:match(cfg.pattern.listnum) then
                                local n = k:match(cfg.pattern.listnum)
                                targs[andnum('liststyle', n + 2)] =
pargs[andnum('liststyle', n)]
                                targs[andnum('group', n + 2)] =
pargs[andnum('group', n)]
                                targs[andnum('groupstyle', n + 2)] =
pargs[andnum('groupstyle', n)]
                                if v and inArray(cfg.keyword.subgroups, v)
then
                                        targs[andnum('list', n + 2)] =
getSubgroup(pargs, n, v)
                                else
                                        targs[andnum('list', n + 2)] = v
                                end
                        elseif (k:match(cfg.pattern.colheadernum) and v ~=
'') then
                                table.insert(colheadernums,
tonumber(k:match(cfg.pattern.colheadernum)))
                        elseif (k:match(cfg.pattern.colnum) and v ~= '') then
                                table.insert(colnums,
tonumber(k:match(cfg.pattern.colnum)))
                        elseif (k:match(cfg.pattern.colfooternum) and v ~=
'') then
                                table.insert(colfooternums,
tonumber(k:match(cfg.pattern.colfooternum)))
                        end
                end
        end
        table.sort(colheadernums)
        table.sort(colnums)
        table.sort(colfooternums)

        -- HTML table for list1
        local coltable = mw.html.create( 'table' ):addClass('navbox-columns-
table')
        local row, col

        local tablestyle = ( (#colheadernums > 0) or (not
isblank(pargs[cfg.arg.fullwidth])) )
                and 'width:100%'
                or 'width:auto; margin-left:auto; margin-right:auto'

        coltable:cssText(concatstyles({
                'border-spacing: 0px; text-align:left',
                tablestyle,
                pargs[cfg.arg.coltablestyle] or ''
        }))
```

```
        --- Header row ---
        if (#colheadernums > 0) then
                row = coltable:tag('tr')
                for k, n in ipairs(colheadernums) do
                        col = row:tag('td'):addClass('navbox-abovebelow')
                        col:cssText(concatstyles({
                                (k > 1) and 'border-left:2px solid #fdfdfd'
or '',
                                'font-weight:bold',
                                pargs[cfg.arg.colheaderstyle] or '',
                                pargs[andnum('colheaderstyle', n)] or ''
                        }))
                        col:attr('colspan',
tonumber(pargs[andnum('colheadercolspan', n)]))
                        col:wikitext(pargs[andnum('colheader', n)])
                end
        end

        --- Main columns ---
        row = coltable:tag('tr'):css('vertical-align', 'top')
        for k, n in ipairs(colnums) do
                if k == 1 and isblank(pargs[andnum('colheader', 1)])
                                and isblank(pargs[andnum('colfooter', 1)])
                                and isblank(pargs[cfg.arg.fullwidth]) then
                        local nopad = inArray(
                                {'off', '0', '0em', '0px'},
                                mw.ustring.gsub(pargs[cfg.arg.padding] or '',
'[;%%]', ''))
                        if not nopad then
                                row:tag('td'):wikitext('   ')
                                        :css('width', (pargs[cfg.arg.padding]
or '5em'))
                        end
                end
                col = row:tag('td'):addClass('navbox-list')
                col:cssText(concatstyles({
                        (k > 1) and 'border-left:2px solid #fdfdfd' or '',
                        'padding:0px',
                        pargs[cfg.arg.colstyle] or '',
                        ((n%2 == 0) and pargs[cfg.arg.evencolstyle] or
pargs[cfg.arg.oddcolstyle]) or '',
                        pargs[andnum('colstyle', n)] or '',
                        'width:' .. (pargs[andnum('colwidth', n)] or
pargs[cfg.arg.colwidth] or '10em')
                }))
                local wt = pargs[andnum('col', n)]
                if wt and inArray(cfg.keyword.subgroups, wt) then
                        wt = getSubgroup(pargs, n, wt, cfg.arg.col_and_num)
                end
                col:tag('div'):newline():wikitext(wt):newline()
        end
```

```
        --- Footer row ---
        if (#colfooternums > 0) then
                row = coltable:tag('tr')
                for k, n in ipairs(colfooternums) do
                        col = row:tag('td'):addClass('navbox-abovebelow')
                        col:cssText(concatstyles({
                                (k > 1) and 'border-left:2px solid #fdfdfd'
or '',
                                'font-weight:bold',
                                pargs[cfg.arg.colfooterstyle] or '',
                                pargs[andnum('colfooterstyle', n)] or ''
                        }))
                        col:attr('colspan',
tonumber(pargs[andnum('colfootercolspan', n)]))
                        col:wikitext(pargs[andnum('colfooter', n)])
                end
        end

        -- assign table to list1
        targs[andnum('list', 1)] = tostring(coltable)
        if isblank(pargs[andnum('colheader', 1)])
                        and isblank(pargs[andnum('col', 1)])
                        and isblank(pargs[andnum('colfooter', 1)]) then
                targs[andnum('list', 1)] = targs[andnum('list', 1)] ..
                        cfg.category.without_first_col
        end

        -- Other parameters
        targs[cfg.arg.border] = pargs[cfg.arg.border] or pargs[1]
        targs[cfg.arg.evenodd] = (not isblank(pargs[cfg.arg.evenodd])) and
pargs[cfg.arg.evenodd] or nil
        targs[cfg.arg.list1padding] = '0px'
        targs[andnum('liststyle', 1)] =
'background:transparent;color:inherit;'
        targs[cfg.arg.style] = concatstyles({pargs[cfg.arg.style],
pargs[cfg.arg.bodystyle]})
        targs[cfg.arg.tracking] = 'no'
        return p._navbox(targs)
end --p._withColumns

-- Template entry points
function p.navbox (frame, boxtype)
        local function readArgs(args, prefix)
                -- Read the arguments in the order they'll be output in, to
make references
                -- number in the right order.
                local _ = 0
                _ = _ + (args[prefix .. cfg.arg.title] and #args[prefix ..
cfg.arg.title] or 0)
                _ = _ + (args[prefix .. cfg.arg.above] and #args[prefix ..
cfg.arg.above] or 0)
```

```
                -- 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[prefix .. andnum('group', i)] and
#args[prefix .. andnum('group', i)] or 0)
                        if inArray(cfg.keyword.subgroups, args[prefix ..
andnum('list', i)]) then
                                for _, v in ipairs(cfg.arg.subgroups_and_num)
do
                                        readArgs(args, prefix ..
string.format(v, i) .. "_")
                                end
                                readArgs(args, prefix .. andnum('col', i) ..
"_")
                        end
                end
                _ = _ + (args[prefix .. cfg.arg.below] and #args[prefix ..
cfg.arg.below] or 0)
                return _
        end

        if not getArgs then
                getArgs = require('Module:Arguments').getArgs
        end
        local args = getArgs(frame, {wrappers = {cfg.pattern[boxtype or
'navbox']}})
        args.argHash = readArgs(args, "")
        args.type = args.type or cfg.keyword[boxtype]

        -- افزودن پارامترهای فارسی:
        local function aliasParam(original, aliases)
                for _, alias in ipairs(aliases) do
                        if args[original] == nil and args[alias] ~= nil then
                                args[original] = args[alias]
                        end
                end
        end

        --  پارامترهای فارسی پایه:
        aliasParam('title', {'عنوان'})
        aliasParam('above', {'بالا'})
        aliasParam('below', {'پایین'})
        aliasParam('image', {'تصویر'})
        aliasParam('imageright', {'تصویر راست'})
        aliasParam('name', {'نام'})
        aliasParam('state', {'حالت'})
        aliasParam('navbar', {'سرپیوند'})
        aliasParam('border', {'حاشیه', 'اطراف'})
        aliasParam('groupwidth', {'عرض گروه'})
        aliasParam('style', {'سبک', 'سبک بدنه'})
```

```
        aliasParam('imagestyle', {'سبک تصویر'})
        aliasParam('imagerightstyle', {'سبک تصویر راست'})
        aliasParam('titlestyle', {'سبک عنوان'})
        aliasParam('basestyle', {'سبک پایه'})
        aliasParam('abovestyle', {'سبک بالا'})
        aliasParam('belowstyle', {'سبک پایین'})
        aliasParam('groupstyle', {'سبک گروه'})
        aliasParam('liststyle', {'سبک فهرست'})
        aliasParam('evenstyle', {'سبک زوج'})
        aliasParam('oddstyle', {'سبک فرد'})
        aliasParam('evenodd', {'زوج فرد'})
        aliasParam('listpadding', {'فاصله فهرست'})
        aliasParam('bodyclass', {'کلاس بدنه'})
        aliasParam('listclass', {'کلاس فهرست'})
        aliasParam('belowclass', {'کلاس پایین'})

        -- گروه‌ها و فهرست‌ها:
        for i = 1, 40 do
                aliasParam('group' .. i, {'گروه' .. numConv('fa', i)})
                aliasParam('list' .. i, {'فهرست' .. numConv('fa', i)})
                aliasParam('group' .. i .. 'style', {'سبک گروه' ..
numConv('fa', i)})
                aliasParam('list' .. i .. 'style', {'سبک فهرست' ..
numConv('fa', i)})
        end

        -- فهرست و گروه پیش‌فرض:
        aliasParam('group1', {'گروه'})
        aliasParam('list1', {'فهرست'})

        return p['_navbox'](args)
end

p[cfg.keyword.with_collapsible_groups] = function (frame)
        return p.navbox(frame, 'with_collapsible_groups')
end

p[cfg.keyword.with_columns] = function (frame)
        return p.navbox(frame, 'with_columns')
end

return p
```