پودمان:Infobox

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

```
local p = {}
local args = {}
local origArgs = {}
local root
local empty_row_categories = {}
local category_in_empty_row_pattern =
'%[%[%s*[Cc][Aa][Tt][Ee][Gg][Oo][Rr][Yy]%s*:[^]]*]]'
local has_rows = false
local lists = {
        plainlist_t = {
                patterns = {
                        '^plainlist$',
                        '%splainlist$',
                        '^plainlist%s',
                        '%splainlist%s'
                },
                found = false,
                styles = 'Plainlist/styles.css'
        },
        hlist_t = {
                patterns = {
                        '^hlist$',
                        '%shlist$',
                        '^hlist%s',
                        '%shlist%s'
                },
                found = false,
                styles = 'Hlist/styles.css'
        }
}

local function has_list_class(args_to_check)
        for _, list in pairs(lists) do
                if not list.found then
                        for _, arg in pairs(args_to_check) do
                                for _, pattern in ipairs(list.patterns) do
                                        if mw.ustring.find(arg or '',
pattern) then
                                                list.found = true
                                                break
                                        end
                                end
```

```
                                if list.found then break end
                        end
                end
        end
end

local function fixChildBoxes(sval, tt)
        local function notempty( s ) return s and s:match( '%S' ) end
        if notempty(sval) then
                local marker = '<span class=special_infobox_marker>'
                local s = sval
                -- start moving templatestyles and categories inside of table
rows
                local slast = ''
                while slast ~= s do
                        slast = s
                        s = mw.ustring.gsub(s,
'(</[Tt][Rr]%s*>%s*)(%[%[%s*[Cc][Aa][Tt][Ee][Gg][Oo][Rr][Yy]%s*:[^]]*%]%])',
'%2%1')
                        s = mw.ustring.gsub(s,
'(</[Tt][Rr]%s*>%s*)(\127[^\127]*UNIQ%-%-templatestyles%-%x+%-
QINU[^\127]*\127)', '%2%1')
                end
                -- end moving templatestyles and categories inside of table
rows
                s = mw.ustring.gsub(s, '(<%s*[Tt][Rr])', marker .. '%1')
                s = mw.ustring.gsub(s, '(</[Tt][Rr]%s*>)', '%1' .. marker)
                if s:match(marker) then
                        s = mw.ustring.gsub(s, marker .. '%s*' .. marker, '')
                        s = mw.ustring.gsub(s, '([\r\n]|-[^\r\n]*[\r\n])%s*'
.. marker, '%1')
                        s = mw.ustring.gsub(s, marker .. '%s*([\r\n]|-)',
'%1')
                        s = mw.ustring.gsub(s,
'(</[Cc][Aa][Pp][Tt][Ii][Oo][Nn]%s*>%s*)' .. marker, '%1')
                        s = mw.ustring.gsub(s,
'(<%s*[Tt][Aa][Bb][Ll][Ee][^<>]*>%s*)' .. marker, '%1')
                        s = mw.ustring.gsub(s, '^(%{|[^\r\n]*[\r\n]%s*)' ..
marker, '%1')
                        s = mw.ustring.gsub(s, '([\r\n]%{|[^\r\n]*[\r\n]%s*)'
.. marker, '%1')
                        s = mw.ustring.gsub(s, marker ..
'(%s*</[Tt][Aa][Bb][Ll][Ee]%s*>)', '%1')
                        s = mw.ustring.gsub(s, marker .. '(%s*\n|%})', '%1')
                end
                if s:match(marker) then
                        local subcells = mw.text.split(s, marker)
                        s = ''
                        for k = 1, #subcells do
                                if k == 1 then
                                        s = s .. subcells[k] .. '</' .. tt ..
```

```
'></tr>'
                                elseif k == #subcells then
                                        local rowstyle = '
style="display:none"'
                                        if notempty(subcells[k]) then
rowstyle = ''        end
                                        s = s .. '<tr' .. rowstyle ..'><' ..
tt .. ' colspan=2>\n' ..
                                                subcells[k]
                                elseif notempty(subcells[k]) then
                                        if (k % 2) == 0 then
                                                s = s .. subcells[k]
                                        else
                                                s = s .. '<tr><' .. tt .. '
colspan=2>\n' ..
                                                        subcells[k] .. '</'
.. tt .. '></tr>'
                                        end
                                end
                        end
                end
                -- the next two lines add a newline at the end of lists for
the PHP parser
                -- [[Special:Diff/849054481]]
                -- remove when [[:phab:T191516]] is fixed or OBE
                s = mw.ustring.gsub(s, '([\r\n][%*#;:][^\r\n]*)$', '%1\n')
                s = mw.ustring.gsub(s, '^([%*#;:][^\r\n]*)$', '%1\n')
                s = mw.ustring.gsub(s, '^([%*#;:])', '\n%1')
                s = mw.ustring.gsub(s, '^(%{%|)', '\n%1')
                return s
        else
                return sval
        end
end

-- Cleans empty tables
local function cleanInfobox()
        root = tostring(root)
        if has_rows == false then
                root = mw.ustring.gsub(root, '<table[^<>]*>%s*</table>', '')
        end
end

-- Returns the union of the values of two tables, as a sequence.
local function union(t1, t2)

        local vals = {}
        for k, v in pairs(t1) do
                vals[v] = true
        end
        for k, v in pairs(t2) do
```

```
                vals[v] = true
        end
        local ret = {}
        for k, v in pairs(vals) do
                table.insert(ret, k)
        end
        return ret
end

-- Returns a table containing the numbers of the arguments that exist
-- for the specified prefix. For example, if the prefix was 'data', and
-- 'data1', 'data2', and 'data5' exist, it would return {1, 2, 5}.
local function getArgNums(prefix)
        local nums = {}
        for k, v in pairs(args) do
                local num = tostring(k):match('^' .. prefix .. '([1-9]%d*)$')
                if num then table.insert(nums, tonumber(num)) end
        end
        table.sort(nums)
        return nums
end

-- Adds a row to the infobox, with either a header cell
-- or a label/data cell combination.
local function addRow(rowArgs)
        if rowArgs.header and rowArgs.header ~= '_BLANK_' then
                has_rows = true
                has_list_class({ rowArgs.rowclass, rowArgs.class,
args.headerclass })
                root
                        :tag('tr')
                                :addClass(rowArgs.rowclass)
                                :cssText(rowArgs.rowstyle)
                                :tag('th')
                                        :attr('colspan', '2')
                                        :addClass('infobox-header')
                                        :addClass(rowArgs.class)
                                        :addClass(args.headerclass)
                                        -- @deprecated next; target .infobox-
<name> .infobox-header
                                        :cssText(args.headerstyle)
                                        :cssText(rowArgs.rowcellstyle)
:wikitext(fixChildBoxes(rowArgs.header, 'th'))
                if rowArgs.data then
                        root:wikitext(
                                '[[Category:Pages using infobox templates
with ignored data cells]]'
                        )
                end
        elseif rowArgs.data and
rowArgs.data:gsub(category_in_empty_row_pattern, ''):match('^%S') then
```

```
                has_rows = true
                has_list_class({ rowArgs.rowclass, rowArgs.class })
                local row = root:tag('tr')
                row:addClass(rowArgs.rowclass)
                row:cssText(rowArgs.rowstyle)
                if rowArgs.label then
                        row
                                :tag('th')
                                        :attr('scope', 'row')
                                        :addClass('infobox-label')
                                        -- @deprecated next; target .infobox-
<name> .infobox-label
                                        :cssText(args.labelstyle)
                                        :cssText(rowArgs.rowcellstyle)
                                        :wikitext(rowArgs.label)
                                        :done()
                end

                local dataCell = row:tag('td')
                dataCell
                        :attr('colspan', not rowArgs.label and '2' or nil)
                        :addClass(not rowArgs.label and 'infobox-full-data'
or 'infobox-data')
                        :addClass(rowArgs.class)
                        -- @deprecated next; target .infobox-<name>
.infobox(-full)-data
                        :cssText(rowArgs.datastyle)
                        :cssText(rowArgs.rowcellstyle)
                        :wikitext(fixChildBoxes(rowArgs.data, 'td'))
        else
                table.insert(empty_row_categories, rowArgs.data or '')
        end
end

local function renderTitle()
        if not args.title then return end

        has_rows = true
        has_list_class({args.titleclass})
        root
                :tag('caption')
                        :addClass('infobox-title')
                        :addClass(args.titleclass)
                        -- @deprecated next; target .infobox-<name> .infobox-
title
                        :cssText(args.titlestyle)
                        :wikitext(args.title)
end

local function renderAboveRow()
        if not args.above then return end
```

```
        has_rows = true
        has_list_class({ args.aboveclass })
        root
                :tag('tr')
                        :tag('th')
                                :attr('colspan', '2')
                                :addClass('infobox-above')
                                :addClass(args.aboveclass)
                                -- @deprecated next; target .infobox-<name>
.infobox-above
                                :cssText(args.abovestyle)
                                :wikitext(fixChildBoxes(args.above,'th'))
end

local function renderBelowRow()
        if not args.below then return end

        has_rows = true
        has_list_class({ args.belowclass })
        root
                :tag('tr')
                        :tag('td')
                                :attr('colspan', '2')
                                :addClass('infobox-below')
                                :addClass(args.belowclass)
                                -- @deprecated next; target .infobox-<name>
.infobox-below
                                :cssText(args.belowstyle)
                                :wikitext(fixChildBoxes(args.below,'td'))
end

local function addSubheaderRow(subheaderArgs)
        if subheaderArgs.data and
                subheaderArgs.data:gsub(category_in_empty_row_pattern,
''):match('^%S') then
                has_rows = true
                has_list_class({ subheaderArgs.rowclass, subheaderArgs.class
})
                local row = root:tag('tr')
                row:addClass(subheaderArgs.rowclass)

                local dataCell = row:tag('td')
                dataCell
                        :attr('colspan', '2')
                        :addClass('infobox-subheader')
                        :addClass(subheaderArgs.class)
                        :cssText(subheaderArgs.datastyle)
                        :cssText(subheaderArgs.rowcellstyle)
                        :wikitext(fixChildBoxes(subheaderArgs.data, 'td'))
        else
                table.insert(empty_row_categories, subheaderArgs.data or '')
```

```
        end
end

local function renderSubheaders()
        if args.subheader then
                args.subheader1 = args.subheader
        end
        if args.subheaderrowclass then
                args.subheaderrowclass1 = args.subheaderrowclass
        end
        local subheadernums = getArgNums('subheader')
        for k, num in ipairs(subheadernums) do
                addSubheaderRow({
                        data = args['subheader' .. tostring(num)],
                        -- @deprecated next; target .infobox-<name> .infobox-
subheader
                        datastyle = args.subheaderstyle,
                        rowcellstyle = args['subheaderstyle' ..
tostring(num)],
                        class = args.subheaderclass,
                        rowclass = args['subheaderrowclass' .. tostring(num)]
                })
        end
end

local function addImageRow(imageArgs)

        if imageArgs.data and
                imageArgs.data:gsub(category_in_empty_row_pattern,
''):match('^%S') then

                has_rows = true
                has_list_class({ imageArgs.rowclass, imageArgs.class })
                local row = root:tag('tr')
                row:addClass(imageArgs.rowclass)

                local dataCell = row:tag('td')
                dataCell
                        :attr('colspan', '2')
                        :addClass('infobox-image')
                        :addClass(imageArgs.class)
                        :cssText(imageArgs.datastyle)
                        :wikitext(fixChildBoxes(imageArgs.data, 'td'))
        else
                table.insert(empty_row_categories, imageArgs.data or '')
        end
end

local function renderImages()
        if args.image then
                args.image1 = args.image
```

```
        end
        if args.caption then
                args.caption1 = args.caption
        end
        local imagenums = getArgNums('image')
        for k, num in ipairs(imagenums) do
                local caption = args['caption' .. tostring(num)]
                local data = mw.html.create():wikitext(args['image' ..
tostring(num)])
                if caption then
                        data
                                :tag('div')
                                        :addClass('infobox-caption')
                                        -- @deprecated next; target .infobox-
<name> .infobox-caption
                                        :cssText(args.captionstyle)
                                        :wikitext(caption)
                end
                addImageRow({
                        data = tostring(data),
                        -- @deprecated next; target .infobox-<name> .infobox-
image
                        datastyle = args.imagestyle,
                        class = args.imageclass,
                        rowclass = args['imagerowclass' .. tostring(num)]
                })
        end
end

-- When autoheaders are turned on, preprocesses the rows
local function preprocessRows()
        if not args.autoheaders then return end
        local rownums = union(getArgNums('header'), getArgNums('data'))
        table.sort(rownums)
        local lastheader
        for k, num in ipairs(rownums) do
                if args['header' .. tostring(num)] then
                        if lastheader then
                                args['header' .. tostring(lastheader)] = nil
                        end
                        lastheader = num
                elseif args['data' .. tostring(num)] and
                        args['data' .. tostring(num)]:gsub(
                                category_in_empty_row_pattern, ''
                        ):match('^%S') then
                        local data = args['data' .. tostring(num)]
                        if data:gsub(category_in_empty_row_pattern,
''):match('%S') then
                                lastheader = nil
                        end
                end
```

```
        end
        if lastheader then
                args['header' .. tostring(lastheader)] = nil
        end
end

-- Gets the union of the header and data argument numbers,
-- and renders them all in order
local function renderRows()

        local rownums = union(getArgNums('header'), getArgNums('data'))
        table.sort(rownums)
        for k, num in ipairs(rownums) do
                addRow({
                        header = args['header' .. tostring(num)],
                        label = args['label' .. tostring(num)],
                        data = args['data' .. tostring(num)],
                        datastyle = args.datastyle,
                        class = args['class' .. tostring(num)],
                        rowclass = args['rowclass' .. tostring(num)],
                        -- @deprecated next; target .infobox-<name> rowclass
                        rowstyle = args['rowstyle' .. tostring(num)],
                        rowcellstyle = args['rowcellstyle' .. tostring(num)]
                })
        end
end

local function renderNavBar()
        if not args.name then return end

        has_rows = true
        root
                :tag('tr')
                        :tag('td')
                                :attr('colspan', '2')
                                :addClass('infobox-navbar')
                                :wikitext(require('Module:Navbar')._navbar{
                                        args.name,
                                        mini = 1,
                                })
end

local function renderItalicTitle()
        local italicTitle = args['italic title'] and
mw.ustring.lower(args['italic title'])
        if italicTitle == '' or italicTitle == 'force' or italicTitle ==
'yes' then
                root:wikitext(require('Module:Italic title')._main({}))
        end
end
```

```
-- Categories in otherwise empty rows are collected in empty_row_categories.
-- This function adds them to the module output. It is not affected by
-- args.decat because this module should not prevent module-external
categories
-- from rendering.
local function renderEmptyRowCategories()
	for _, s in ipairs(empty_row_categories) do
		root:wikitext(s)
	end
end

-- Render tracking categories. args.decat == turns off tracking categories.
local function renderTrackingCategories()
	if args.decat == 'yes' then return end
	if args.child == 'yes' then
		if args.title then
			root:wikitext(
				'[[Category:Pages using embedded infobox
templates with the title parameter]]'
			)
		end
	elseif #(getArgNums('data')) == 0 and
mw.title.getCurrentTitle().namespace == 0 then
		root:wikitext('[[Category:Articles using infobox templates
with no data rows]]')
	end
end

--[=[
Loads the templatestyles for the infobox.

TODO: FINISH loading base templatestyles here rather than in
MediaWiki:Common.css. There are 4-5000 pages with 'raw' infobox tables.
See [[Mediawiki_talk:Common.css/to_do#Infobox]] and/or come help :).
When we do this we should clean up the inline CSS below too.
Will have to do some bizarre conversion category like with sidebar.

]=]
local function loadTemplateStyles()
	local frame = mw.getCurrentFrame()
	local hlist_templatestyles = ''
	if lists.hlist_t.found then
		hlist_templatestyles = frame:extensionTag{
			name = 'templatestyles', args = { src =
lists.hlist_t.styles }
		}
	end
	local plainlist_templatestyles = ''
	if lists.plainlist_t.found then
		plainlist_templatestyles = frame:extensionTag{
			name = 'templatestyles', args = { src =
```

```
lists.plainlist_t.styles }
                }
        end
        -- See function description
        local base_templatestyles = frame:extensionTag{
                name = 'templatestyles', args = { src =
'Module:Infobox/styles.css' }
        }

        local templatestyles = ''
        if args['templatestyles'] then
                templatestyles = frame:extensionTag{
                        name = 'templatestyles', args = { src =
args['templatestyles'] }
                }
        end
        local child_templatestyles = ''
        if args['child templatestyles'] then
                child_templatestyles = frame:extensionTag{
                        name = 'templatestyles', args = { src = args['child
templatestyles'] }
                }
        end
        local grandchild_templatestyles = ''
        if args['grandchild templatestyles'] then
                grandchild_templatestyles = frame:extensionTag{
                        name = 'templatestyles', args = { src =
args['grandchild templatestyles'] }
                }
        end
        return table.concat({
                -- hlist -> plainlist -> base is best-effort to preserve old
Common.css ordering.
                -- this ordering is not a guarantee because the rows of
interest invoking
                -- each class may not be on a specific page
                hlist_templatestyles,
                plainlist_templatestyles,
                base_templatestyles,
                templatestyles,
                child_templatestyles,
                grandchild_templatestyles
        })
end

-- common functions between the child and non child cases
local function structure_infobox_common()
        renderSubheaders()
        renderImages()
        preprocessRows()
        renderRows()
```

```
        renderBelowRow()
        renderNavBar()
        renderItalicTitle()
        renderEmptyRowCategories()
        renderTrackingCategories()
        cleanInfobox()
end

-- Specify the overall layout of the infobox, with special settings if the
-- infobox is used as a 'child' inside another infobox.
local function _infobox()
        if args.child ~= 'yes' then
                root = mw.html.create('table')

                root
                        :addClass(args.subbox == 'yes' and 'infobox-subbox'
or 'infobox')
                        :addClass(args.bodyclass)
                        -- @deprecated next; target .infobox-<name>
                        :cssText(args.bodystyle)
                has_list_class({ args.bodyclass })

                renderTitle()
                renderAboveRow()
        else
                root = mw.html.create()

                root
                        :wikitext(args.title)
        end
        structure_infobox_common()
        return loadTemplateStyles() .. root
end

-- If the argument exists and isn't blank, add it to the argument table.
-- Blank arguments are treated as nil to match the behaviour of
ParserFunctions.
local function preprocessSingleArg(argName)
        if origArgs[argName] and origArgs[argName] ~= '' then
                args[argName] = origArgs[argName]
        end
end

-- Assign the parameters with the given prefixes to the args table, in order,
in
-- batches of the step size specified. This is to prevent references etc.
from
-- appearing in the wrong order. The prefixTable should be an array
containing
-- tables, each of which has two possible fields, a "prefix" string and a
-- "depend" table. The function always parses parameters containing the
```

```
"prefix"
-- string, but only parses parameters in the "depend" table if the prefix
-- parameter is present and non-blank.
local function preprocessArgs(prefixTable, step)
	if type(prefixTable) ~= 'table' then
		error("Non-table value detected for the prefix table", 2)
	end
	if type(step) ~= 'number' then
		error("Invalid step value detected", 2)
	end

	-- Get arguments without a number suffix, and check for bad input.
	for i,v in ipairs(prefixTable) do
		if type(v) ~= 'table' or type(v.prefix) ~= "string" or
			(v.depend and type(v.depend) ~= 'table') then
			error('Invalid input detected to preprocessArgs
prefix table', 2)
		end
		preprocessSingleArg(v.prefix)
		-- Only parse the depend parameter if the prefix parameter is
present
		-- and not blank.
		if args[v.prefix] and v.depend then
			for j, dependValue in ipairs(v.depend) do
				if type(dependValue) ~= 'string' then
					error('Invalid "depend" parameter
value detected in preprocessArgs')
				end
				preprocessSingleArg(dependValue)
			end
		end
	end

	-- Get arguments with number suffixes.
	local a = 1 -- Counter variable.
	local moreArgumentsExist = true
	while moreArgumentsExist == true do
		moreArgumentsExist = false
		for i = a, a + step - 1 do
			for j,v in ipairs(prefixTable) do
				local prefixArgName = v.prefix .. tostring(i)
				if origArgs[prefixArgName] then
					-- Do another loop if any arguments
are found, even blank ones.
					moreArgumentsExist = true
					preprocessSingleArg(prefixArgName)
				end
				-- Process the depend table if the prefix
argument is present
				-- and not blank, or we are processing
"prefix1" and "prefix" is
```

```
                                -- present and not blank, and if the depend
table is present.
                                if v.depend and (args[prefixArgName] or (i ==
1 and args[v.prefix])) then
                                        for j,dependValue in ipairs(v.depend)
do
                                                local dependArgName =
dependValue .. tostring(i)
preprocessSingleArg(dependArgName)
                                        end
                                end
                        end
                end
                a = a + step
        end
end

-- Parse the data parameters in the same order that the old {{infobox}} did,
so
-- that references etc. will display in the expected places. Parameters that
-- depend on another parameter are only processed if that parameter is
present,
-- to avoid phantom references appearing in article reference lists.
local function parseDataParameters()

        preprocessSingleArg('autoheaders')
        preprocessSingleArg('child')
        preprocessSingleArg('bodyclass')
        preprocessSingleArg('subbox')
        preprocessSingleArg('bodystyle')
        preprocessSingleArg('title')
        preprocessSingleArg('titleclass')
        preprocessSingleArg('titlestyle')
        preprocessSingleArg('above')
        preprocessSingleArg('aboveclass')
        preprocessSingleArg('abovestyle')
        preprocessArgs({
                {prefix = 'subheader', depend = {'subheaderstyle',
'subheaderrowclass'}}
        }, 10)
        preprocessSingleArg('subheaderstyle')
        preprocessSingleArg('subheaderclass')
        preprocessArgs({
                {prefix = 'image', depend = {'caption', 'imagerowclass'}}
        }, 10)
        preprocessSingleArg('captionstyle')
        preprocessSingleArg('imagestyle')
        preprocessSingleArg('imageclass')
        preprocessArgs({
                {prefix = 'header'},
                {prefix = 'data', depend = {'label'}},
```

```
                {prefix = 'rowclass'},
                {prefix = 'rowstyle'},
                {prefix = 'rowcellstyle'},
                {prefix = 'class'}
        }, 50)
        preprocessSingleArg('headerclass')
        preprocessSingleArg('headerstyle')
        preprocessSingleArg('labelstyle')
        preprocessSingleArg('datastyle')
        preprocessSingleArg('below')
        preprocessSingleArg('belowclass')
        preprocessSingleArg('belowstyle')
        preprocessSingleArg('name')
        -- different behaviour for italics if blank or absent
        args['italic title'] = origArgs['italic title']
        preprocessSingleArg('decat')
        preprocessSingleArg('templatestyles')
        preprocessSingleArg('child templatestyles')
        preprocessSingleArg('grandchild templatestyles')
end

-- If called via #invoke, use the args passed into the invoking template.
-- Otherwise, for testing purposes, assume args are being passed directly in.
function p.infobox(frame)
        if frame == mw.getCurrentFrame() then
                origArgs = frame:getParent().args
        else
                origArgs = frame
        end
        parseDataParameters()
        return _infobox()
end

-- For calling via #invoke within a template
function p.infoboxTemplate(frame)
        origArgs = {}
        for k,v in pairs(frame.args) do origArgs[k] = mw.text.trim(v) end
        parseDataParameters()
        return _infobox()
end
return p
```