Modul:Section link

This module creates links to sections, nicely formatted with the "§" symbol instead of the default "#".

Inhaltsverzeichnis

1 Usage 1
1.1 From wikitext 1
1.2 From Lua 1
2 Examples 2
3 See also 2

Usage

From wikitext

From wikitext, this module should be used via the template Vorlage:Tl. Please see the template page for documentation.

From Lua

First, load the module:

```
local mSectionLink = require('Module:Section link')
```

You can then make section links via the _main function.

```
mSectionLink._main(page, sections, options, title)
```

Parameters:

- *page* - the page name to link to. Defaults to the full page name of *title*, or the current title if that is not specified.
- *sections* - an array of section names to link to, or a string to link to just one section name.
- *options* - a table of options. Accepts the following fields:
  - *nopage* - set this to true to avoid displaying the base page name in links.
- *title* - a default mw.title object to use instead of the current title. Intended for testing purposes.

All parameters are optional.

Examples

Lua code	Wikitext code	Result
Vorlage: Code	{{section link|Paris}}	Vorlage:Section link
Vorlage: Code	{{section link|Paris|Architecture}}	Vorlage:Section link
Vorlage: Code	{{section link|Paris|Architecture|Culture}}	Vorlage:Section link
Vorlage: Code	{{section link|Paris|Architecture|Culture|Sport}}	Vorlage:Section link
Vorlage: Code	{{section link|Paris|Architecture|Culture|Sport|nopage=yes}}	Vorlage:Section link

See also

- The character used is Vorlage:Unichar

```
-- This module implements {{section link}}.
require('Module:No globals');

local checkType = require('libraryUtil').checkType

local p = {}

local function makeSectionLink(page, section, display)
	display = display or section
	page = page or ''
	-- MediaWiki doesn't allow these in `page`, so only need to do for `secti
	if type(section) == 'string' then
		section = string.gsub(section, "{", "&#x7B;")
		section = string.gsub(section, "}", "&#x7D;")
	end
	return string.format('[[%s#%s|%s]]', page, section, display)
end

local function normalizeTitle(title)
	title = mw.ustring.gsub(mw.ustring.gsub(title, "'", ""), '"', '')
	title = mw.ustring.gsub(title, "%b<>", "")
	return mw.title.new(title).prefixedText
end

function p._main(page, sections, options, title)
	-- Validate input.
	checkType('_main', 1, page, 'string', true)
	checkType('_main', 3, options, 'table', true)
	if sections == nil then
		sections = {}
	elseif type(sections) == 'string' then
		sections = {sections}
```

```
	elseif type(sections) ~= 'table' then
		error(string.format(
			"type error in argument #2 to '_main' " ..
			"(string, table or nil expected, got %s)",
			type(sections)
		), 2)
	end
	options = options or {}
	title = title or mw.title.getCurrentTitle()

	-- Deal with blank page names elegantly
	if page and not page:find('%S') then
		page = nil
		options.nopage = true
	end

	-- Make the link(s).
	local isShowingPage = not options.nopage
	if #sections <= 1 then
		local linkPage = page or ''
		local section = sections[1] or 'Notes'
		local display = '§ ' .. section
		if isShowingPage then
			page = page or title.prefixedText
			if options.display and options.display ~= '' then
				if normalizeTitle(options.display) == normalizeTi
					display = options.display .. ' ' .. displ
				else
					error(string.format(
						'Display title "%s" was ignored s
						"not equivalent to the page's act
						options.display
					), 0)
				end
			else
				display = page .. ' ' .. display
			end
		end
		return makeSectionLink(linkPage, section, display)
	else
		-- Multiple sections. First, make a list of the links to display
		local ret = {}
		for i, section in ipairs(sections) do
			ret[i] = makeSectionLink(page, section)
		end

		-- Assemble the list of links into a string with mw.text.listToTe
		-- We use the default separator for mw.text.listToText, but a cus
		-- conjunction. There is also a special case conjunction if we or
		-- have two links.
		local conjunction
		if #sections == 2 then
			conjunction = '&#8203; and '
		else
			conjunction = ', and '
		end
		ret = mw.text.listToText(ret, nil, conjunction)

		-- Add the intro text.
		local intro = '§§ '
		if isShowingPage then
			intro = (page or title.prefixedText) .. ' ' .. intro
		end
		ret = intro .. ret
```

```
                return ret
        end
end

function p.main(frame)
        local yesno = require('Module:Yesno')
        local args = require('Module:Arguments').getArgs(frame, {
                wrappers = 'Template:Section link',
                valueFunc = function (key, value)
                        value = value:match('^%s*(.-)%s*$') -- Trim whitespace
                        -- Allow blank first parameters, as the wikitext template
                        if value ~= '' or key == 1 then
                                return value
                        end
                end
        })

        for k, v in pairs(args) do
                if 'number' == type(k) then
                        if not yesno (args['keep-underscores']) then
                                args[k] = mw.uri.decode (v, 'WIKI');
                        else
                                args[k] = mw.uri.decode (v, 'PATH');
                        end
                end
        end

        -- Sort the arguments.
        local page
        local sections, options = {}, {}
        for k, v in pairs(args) do
                if k == 1 then
                        -- Doing this in the loop because of a bug in [[Module:Ar
                        -- when using pairs with deleted arguments.
                        page = mw.text.decode(v, true)
                elseif type(k) == 'number' then
                        sections[k] = v
                else
                        options[k] = v
                end
        end

        options.nopage = yesno (options.nopage);

        -- Extract section from page, if present
        if page then
                local p, s = page:match('^(.-)#(.*)$')
                if p then page, sections[1] = p, s end
        end

        -- Compress the sections array.
        local function compressArray(t)
                local nums, ret = {}, {}
                for num in pairs(t) do
                        nums[#nums + 1] = num
                end
                table.sort(nums)
                for i, num in ipairs(nums) do
                        ret[i] = t[num]
                end
                return ret
        end
        sections = compressArray(sections)
```

```
        return p._main(page, sections, options)
end

return p
```