Inhaltsverzeichnis

- 1. Modul:Format link
- 2. Modul:Hatnote

Modul:Format link

This module, migrated from origins in Module:Hatnote, provides functionality for formatting links for display, including that powering the Vorlage:Tl template.

It can pretty-format section links with the section symbol ("§") and appropriate whitespace, it automatically escapes category and file names with the colon trick, and includes functionality for italicizing the page or section name, and for detecting and categorizing results that produce red links.

Inhaltsverzeichnis

- 1 Use from wikitext
- 2 Use from other Lua modules
  - 2.1 _formatLink
  - 2.2 formatPages
- 3 Errors

Use from wikitext

The functions in this module cannot be used directly from #invoke, and must be used through templates instead. Please see Template:Format link for documentation on usage of that template.

Use from other Lua modules

To load this module from another Lua module, use the following code:

```
local mFormatLink = require('Module:Format link')
```

You can then use the functions as documented below.

_formatLink

```
mFormatLink._formatLink{
    link = 'Link',
    display = 'Display',
    target = 'Target',
    italicizePage = true,
    italicizeSection = true,
    categorizeMissing = 'Pages using formatted red links'
}
```

Formats *link* as a wikilink. Categories and files are automatically escaped with the colon trick, and links to sections are automatically formatted as *page § section*, rather than the MediaWiki default of *page#section*.

Several options modify the output:

- If the *display* value is present, then it will be used as a display value. Any manual piping (using the {{!}} magic word or similar) present in *link* will be overridden by the *display* value if present.
- If the *target* value is present, then it will override *link* as a target, but the result will still be *displayed* using either the value from *display*, or the result of formatting *link*.
- If *italicizePage* is true, then the page portion of the link is italicized if present.
- If *italicizeSection* is true, then the section portion of the link is italicized if present.
- If *categorizeMissing* is a non-empty string, then that value is used as a category name, and that category will be applied if the resulting target of the link (no matter whether through *link* or through *target*) doesn't exist.

Examples

`mFormatLink._formatLink{link = 'Foo#Bar'}` → [[:Foo#Bar|Foo § Bar]] → Vorlage:Format link
`mFormatLink._formatLink{link = 'Baz', display = 'Qux'}` → [[:Baz|Qux]] → Vorlage:Format link
`mFormatLink._formatLink{link = 'Foo|Bar', display = 'Baz'}` → [[:Foo|Baz]] → Vorlage:Format link
`mFormatLink._formatLink{link = '#Foo', target = 'Example#Foo'}` → [[:Example#Foo|§ Foo]] → Vorlage:Format link
`mFormatLink._formatLink{link = 'The Lord of the Rings#Plot', italicizePage = true}` → [[:The Lord of the Rings#Plot|''The Lord of the Rings'' § Plot]] → Vorlage:Format link
`mFormatLink._formatLink{link = 'Cybercrime Prevention Act of 2012#Disini v. Secretary of Justice', italicizeSection = true}` → [[:Cybercrime Prevention Act of 2012#Disini v. Secretary of Justice|Cybercrime Prevention Act of 2012 § ''Disini v. Secretary of Justice'']] → Vorlage:Format link
`mFormatLink._formatLink{link = 'Nonexistent page', categorizeMissing = 'Example'}` → [[:Nonexistent page]][[Category:Example]] → Vorlage:Format link
`mFormatLink._formatLink{link = 'Existing', categorizeMissing = 'Example'}` → [[:Existing]] → Vorlage:Format link

formatPages

`mFormatLink.formatPages(options, pages)`

This derived function is useful for lists that format many links. It formats an array of pages using the _formatLink function, and returns the result as an array. Options in the *options* table are applied, and use the same names as the options for *_formatLink*.

Example

Vorlage:Code → Vorlage:Code

Errors

If *_formatLink* is used and neither a *link* nor a *target* argument is provided, then the module will produce an error message instead of its usual output, as it *cannot* then produce valid output.

You can solve this error by providing appropriate parameters to *_formatLink*, or you may want to ensure that a more descriptive error is provided by a downstream template or module when it would otherwise call *_formatLink* with inadequate arguments.

```
--------------------------------------------------------------------------------
-- Format link
--
-- Makes a wikilink from the given link and display values. Links are escaped
-- with colons if necessary, and links to sections are detected and displayed
-- with " § " as a separator rather than the standard MediaWiki "#". Used in
-- the {{format link}} template.
--------------------------------------------------------------------------------
local libraryUtil = require('libraryUtil')
local checkType = libraryUtil.checkType
local checkTypeForNamedArg = libraryUtil.checkTypeForNamedArg
local mArguments -- lazily initialise [[Module:Arguments]]
local mError -- lazily initialise [[Module:Error]]
local yesno -- lazily initialise [[Module:Yesno]]

local p = {}

--------------------------------------------------------------------------------
-- Helper functions
--------------------------------------------------------------------------------

local function getArgs(frame)
        -- Fetches the arguments from the parent frame. Whitespace is trimmed and
        -- blanks are removed.
        mArguments = require('Module:Arguments')
        return mArguments.getArgs(frame, {parentOnly = true})
end

local function removeInitialColon(s)
        -- Removes the initial colon from a string, if present.
        return s:match('^:?(.*)')
end

local function maybeItalicize(s, shouldItalicize)
        -- Italicize s if s is a string and the shouldItalicize parameter is true.
        if s and shouldItalicize then
                return '<i>' .. s .. '</i>'
        else
                return s
        end
end

local function parseLink(link)
        -- Parse a link and return a table with the link's components.
```

```
	-- These components are:
	-- - link: the link, stripped of any initial colon (always present)
	-- - page: the page name (always present)
	-- - section: the page name (may be nil)
	-- - display: the display text, if manually entered after a pipe (may be nil)
	link = removeInitialColon(link)

	-- Find whether a faux display value has been added with the {{!}} magic
	-- word.
	local prePipe, display = link:match('^(.-)|(.*)$')
	link = prePipe or link

	-- Find the page, if it exists.
	-- For links like [[#Bar]], the page will be nil.
	local preHash, postHash = link:match('^(.-)#(.*)$')
	local page
	if not preHash then
		-- We have a link like [[Foo]].
		page = link
	elseif preHash ~= '' then
		-- We have a link like [[Foo#Bar]].
		page = preHash
	end

	-- Find the section, if it exists.
	local section
	if postHash and postHash ~= '' then
		section = postHash
	end

	return {
		link = link,
		page = page,
		section = section,
		display = display,
	}
end

local function formatDisplay(parsed, options)
	-- Formats a display string based on a parsed link table (matching the
	-- output of parseLink) and an options table (matching the input options for
	-- _formatLink).
	local page = maybeItalicize(parsed.page, options.italicizePage)
	local section = maybeItalicize(parsed.section, options.italicizeSection)
	if (not section) then
		return page
	elseif (not page) then
		return mw.ustring.format('§ %s', section)
	else
		return mw.ustring.format('%s § %s', page, section)
	end
end
```

```
local function missingArgError(target)
	mError = require('Module:Error')
	return mError.error{message =
		'Error: no link or target specified! ([[' .. target .. '#Errors|help]])'
	}
end

--------------------------------------------------------------------------------
-- Main functions
--------------------------------------------------------------------------------

function p.formatLink(frame)
	-- The formatLink export function, for use in templates.
	yesno = require('Module:Yesno')
	local args = getArgs(frame)
	local link = args[1] or args.link
	local target = args[3] or args.target
	if not (link or target) then
		return missingArgError('Template:Format link')
	end

	return p._formatLink{
		link = link,
		display = args[2] or args.display,
		target = target,
		italicizePage = yesno(args.italicizepage),
		italicizeSection = yesno(args.italicizesection),
		categorizeMissing = args.categorizemissing
	}
end

function p._formatLink(options)
	-- The formatLink export function, for use in modules.
	checkType('_formatLink', 1, options, 'table')
	local function check(key, expectedType) --for brevity
		checkTypeForNamedArg(
			'_formatLink', key, options[key], expectedType or 'string', true
		)
	end
	check('link')
	check('display')
	check('target')
	check('italicizePage', 'boolean')
	check('italicizeSection', 'boolean')
	check('categorizeMissing')

	-- Normalize link and target and check that at least one is present
	if options.link == '' then options.link = nil end
	if options.target == '' then options.target = nil end
	if not (options.link or options.target) then
		return missingArgError('Module:Format link')
```

```
	end

	local parsed = parseLink(options.link)
	local display = options.display or parsed.display
	local catMissing = options.categorizeMissing
	local category = ''

	-- Find the display text
	if not display then display = formatDisplay(parsed, options) end

	-- Handle the target option if present
	if options.target then
		local parsedTarget = parseLink(options.target)
		parsed.link = parsedTarget.link
		parsed.page = parsedTarget.page
	end

	-- Test if page exists if a diagnostic category is specified
	if catMissing and (mw.ustring.len(catMissing) > 0) then
		local title = nil
		if parsed.page then title = mw.title.new(parsed.page) end
		if title and (not title.isExternal) and (not title.exists) then
			category = mw.ustring.format('[[Category:%s]]', catMissing)
		end
	end

	-- Format the result as a link
	if parsed.link == display then
		return mw.ustring.format('[[:%s]]%s', parsed.link, category)
	else
		return mw.ustring.format('[[:%s|%s]]%s', parsed.link, display, category)
	end
end

--------------------------------------------------------------------------------
-- Derived convenience functions
--------------------------------------------------------------------------------

function p.formatPages(options, pages)
	-- Formats an array of pages using formatLink and the given options table,
	-- and returns it as an array. Nil values are not allowed.
	local ret = {}
	for i, page in ipairs(pages) do
		ret[i] = p._formatLink{
			link = page,
			categorizeMissing = options.categorizeMissing,
			italicizePage = options.italicizePage,
			italicizeSection = options.italicizeSection
		}
	end
	return ret
end
```

```
return p
```

Modul:Hatnote

Vorlage:Lua

This is a meta-module that provides various functions for making hatnotes. It implements the Vorlage:Tl template, for use in hatnotes at the top of pages, and the Vorlage:Tl template, which is used to format a wikilink for use in hatnotes. It also contains a number of helper functions for use in other Lua hatnote modules.

Inhaltsverzeichnis

- 1 Use from wikitext
- 2 Use from other Lua modules
  - 2.1 Hatnote
  - 2.2 Find namespace id
  - 2.3 Make wikitext error
- 3 Examples

Use from wikitext

The functions in this module cannot be used directly from #invoke, and must be used through templates instead. Please see Template:Hatnote and Template:Format link for documentation.

Use from other Lua modules

To load this module from another Lua module, use the following code.

```
local mHatnote = require('Module:Hatnote')
```

You can then use the functions as documented below.

Hatnote

```
mHatnote._hatnote(s, options)
```

Formats the string *s* as a hatnote. This encloses *s* in the tags Vorlage:Tag. Options are provided in the *options* table. Options include:

- *options.extraclasses* - a string of extra classes to provide
- *options.selfref* - if this is not nil or false, adds the class "selfref", used to denote self-references to Wikipedia (see Template:Selfref))

The CSS of the hatnote class is defined in Module:Hatnote/styles.css.

Example 1

```
mHatnote._hatnote('This is a hatnote.')
```

Produces: Vorlage:Tag

Displays as: Vorlage:Hatnote

Example 2

```
mHatnote._hatnote('This is a hatnote.', {extraclasses = 'boilerplate seealso', selfref = true})
```

Produces: Vorlage:Tag

Displayed as: Vorlage:Hatnote

Find namespace id

```
mHatnote.findNamespaceId(link, removeColon)
```

Finds the namespace id of the string *link*, which should be a valid page name, with or without the section name. This function will not work if the page name is enclosed with square brackets. When trying to parse the namespace name, colons are removed from the start of the link by default. This is helpful if users have specified colons when they are not strictly necessary. If you do not need to check for initial colons, set *removeColon* to false.

Examples

Vorlage:Code → 0
Vorlage:Code → 14
Vorlage:Code → 14
Vorlage:Code → 0 (the namespace is detected as ":Category", rather than "Category")

Make wikitext error

```
mHatnote.makeWikitextError(msg, helpLink, addTrackingCategory)
```

Formats the string *msg* as a red wikitext error message, with optional link to a help page *helpLink*. Normally this function also adds Vorlage:Clc. To suppress categorization, pass `false` as third parameter of the function (`addTrackingCategory`).

Examples:

Vorlage:Code → **Error: an error has occurred.**
Vorlage:Code → **Error: an error has occurred (help).**

Examples

For an example of how this module is used in other Lua modules, see Module:Main

---

```
--------------------------------------------------------------------------------
--                            Module:Hatnote                                  --
--                                                                            --
-- This module produces hatnote links and links to related articles. It       --
```

```
-- implements the {{hatnote}} and {{format link}} meta-templates and includes --
-- helper functions for other Lua hatnote modules.                            --
--------------------------------------------------------------------------------

local libraryUtil = require('libraryUtil')
local checkType = libraryUtil.checkType
local checkTypeForNamedArg = libraryUtil.checkTypeForNamedArg
local mArguments -- lazily initialise [[Module:Arguments]]
local yesno -- lazily initialise [[Module:Yesno]]
local formatLink -- lazily initialise [[Module:Format link]] ._formatLink

local p = {}

--------------------------------------------------------------------------------
-- Helper functions
--------------------------------------------------------------------------------

local curNs = mw.title.getCurrentTitle().namespace
p.missingTargetCat =
        --Default missing target category, exported for use in related modules
        ((curNs ==  0) or (curNs == 14)) and
        'Articles with hatnote templates targeting a nonexistent page' or nil

local function getArgs(frame)
        -- Fetches the arguments from the parent frame. Whitespace is trimmed and
        -- blanks are removed.
        mArguments = require('Module:Arguments')
        return mArguments.getArgs(frame, {parentOnly = true})
end

local function removeInitialColon(s)
        -- Removes the initial colon from a string, if present.
        return s:match('^:?(.*)')
end

function p.findNamespaceId(link, removeColon)
        -- Finds the namespace id (namespace number) of a link or a pagename. This
        -- function will not work if the link is enclosed in double brackets. Colons
        -- are trimmed from the start of the link by default. To skip colon
        -- trimming, set the removeColon parameter to false.
        checkType('findNamespaceId', 1, link, 'string')
        checkType('findNamespaceId', 2, removeColon, 'boolean', true)
        if removeColon ~= false then
                link = removeInitialColon(link)
        end
        local namespace = link:match('^(.-):')
        if namespace then
                local nsTable = mw.site.namespaces[namespace]
                if nsTable then
                        return nsTable.id
                end
        end
```

```
        return 0
end

function p.makeWikitextError(msg, helpLink, addTrackingCategory, title)
        -- Formats an error message to be returned to wikitext. If
        -- addTrackingCategory is not false after being returned from
        -- [[Module:Yesno]], and if we are not on a talk page, a tracking category
        -- is added.
        checkType('makeWikitextError', 1, msg, 'string')
        checkType('makeWikitextError', 2, helpLink, 'string', true)
        yesno = require('Module:Yesno')
        title = title or mw.title.getCurrentTitle()
        -- Make the help link text.
        local helpText
        if helpLink then
                helpText = ' ([[' .. helpLink .. '|help]])'
        else
                helpText = ''
        end
        -- Make the category text.
        local category
        if not title.isTalkPage -- Don't categorise talk pages
                and title.namespace ~= 2 -- Don't categorise userspace
                and yesno(addTrackingCategory) ~= false -- Allow opting out
        then
                category = 'Hatnote templates with errors'
                category = mw.ustring.format(
                        '[[%s:%s]]',
                        mw.site.namespaces[14].name,
                        category
                )
        else
                category = ''
        end
        return mw.ustring.format(
                '<strong class="error">Error: %s%s.</strong>%s',
                msg,
                helpText,
                category
        )
end

function p.disambiguate(page, disambiguator)
        -- Formats a page title with a disambiguation parenthetical,
        -- i.e. "Example" → "Example (disambiguation)".
        checkType('disambiguate', 1, page, 'string')
        checkType('disambiguate', 2, disambiguator, 'string', true)
        disambiguator = disambiguator or 'disambiguation'
        return mw.ustring.format('%s (%s)', page, disambiguator)
end

--------------------------------------------------------------------------------
```

```
-- Hatnote
--
-- Produces standard hatnote text. Implements the {{hatnote}} template.
--------------------------------------------------------------------------------

function p.hatnote(frame)
	local args = getArgs(frame)
	local s = args[1]
	if not s then
		return p.makeWikitextError(
			'no text specified',
			'Template:Hatnote#Errors',
			args.category
		)
	end
	return p._hatnote(s, {
		extraclasses = args.extraclasses,
		selfref = args.selfref
	})
end

function p._hatnote(s, options)
	checkType('_hatnote', 1, s, 'string')
	checkType('_hatnote', 2, options, 'table', true)
	options = options or {}
	local inline = options.inline
	local hatnote = mw.html.create(inline == 1 and 'span' or 'div')
	local extraclasses
	if type(options.extraclasses) == 'string' then
		extraclasses = options.extraclasses
	end

	hatnote
		:attr('role', 'note')
		:addClass(inline == 1 and 'hatnote-inline' or 'hatnote')
		:addClass('navigation-not-searchable')
		:addClass(extraclasses)
		:addClass(options.selfref and 'selfref')
		:wikitext(s)

	return mw.getCurrentFrame():extensionTag{
		name = 'templatestyles', args = { src = 'Module:Hatnote/styles.css' }
	} .. tostring(hatnote)
end

return p
```