

Modul:Format link

This module, migrated from origins in [Module:Hatnote](#), provides functionality for formatting links for display, including that powering the [Vorlage:TI](#) 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	1
2 Use from other Lua modules	1
2.1 <code>_formatLink</code>	1
2.2 <code>formatPages</code>	2
3 Errors	3

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 §&nbsp;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|§&nbsp;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" §&nbsp;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 §&nbsp;"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
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
    -- _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.usttring.format('$&nbsp;%s', section)
    else
        return mw.usttring.format('%s $&nbsp;%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|
    }
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'
        )
    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.ustr.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.ustr.format('[[Category:%s]]', catMissing)
        end
    end

    -- Format the result as a link
    if parsed.link == display then
```



```
        else      return mw.ustring.format('[[:%s]]%s', parsed.link, category)
        end      return mw.ustring.format('[[:%s|%s]]%s', parsed.link, display, ca
    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
```