

Toggle menu
Blue Gold Program Wiki

Navigation

- [Main page](#)
- [Recent changes](#)
- [Random page](#)
- [Help about MediaWiki](#)

Tools

- [What links here](#)
- [Related changes](#)
- [Special pages](#)
- [Permanent link](#)
- [Page information](#)

Personal tools

- [Log in](#)

personal-extra

Toggle search

Search

Random page

Views

- [View](#)
- [View source](#)
- [History](#)
- [PDF Export](#)

Actions

Module:Hatnote

From Blue Gold Program Wiki

The printable version is no longer supported and may have rendering errors. Please update your browser bookmarks and please use the default browser print function instead.

Documentation for this module may be created at [Module:Hatnote/doc](#)

```

-----
---
--
--
--
--
-- 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 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

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.formatPages(...)
    -- Formats a list of pages using formatLink and returns it as an
    array. Nil
    -- values are not allowed.
    local pages = {...}
    local ret = {}
    for i, page in ipairs(pages) do
        ret[i] = p._formatLink{link = page}
    end
    return ret
end

function p.formatPageTables(...)
    -- Takes a list of page/display tables and returns it as a list of
    -- formatted links. Nil values are not allowed.
    local pages = {...}
    local links = {}
    for i, t in ipairs(pages) do
        checkType('formatPageTables', i, t, 'table')
        local link = t[1]
        local display = t[2]
        links[i] = p._formatLink{link = link, display = display}
    end
    return links
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 = string.format(
        '[[%s:%s]]',
        mw.site.namespaces[14].name,
        category
    )
else
    category = ''
end
return string.format(
    '<strong class="error">Error: %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 string.format('%s (%s)', page, disambiguator)
end

```

```

-----
---
-- 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.
-----
---

```

```

function p.formatLink(frame)
    -- The formatLink export function, for use in templates.
    yesno = require('Module:Yesno')

```

```

local args = getArgs(frame)
local link = args[1]
if not link then
    return p.makeWikitextError(
        'no link specified',
        'Template:Format link#Errors',
        args.category
    )
end
return p._formatLink{
    link = link,
    display = args[2],
    italicizePage = yesno(args.italicizepage),
    italicizeSection = yesno(args.italicizesection),
}
end

local function italicize(s)
    -- Italicize a string.
    return '<i>' .. s .. '</i>'
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 italicize(s)
    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

function p._formatLink(options)
    -- The formatLink export function, for use in modules.
    checkType('_formatLink', 1, options, 'table')
    checkTypeForNamedArg('_formatLink', 'link', options.link, 'string',
false)
    checkTypeForNamedArg(
        '_formatLink',
        'display',
        options.display,
        'string',
        true
    )
    checkTypeForNamedArg(
        '_formatLink',
        'italicizePage',
        options.italicizePage,
        'boolean',
        true
    )
    checkTypeForNamedArg(
        '_formatLink',
        'italicizeSection',
        options.italicizeSection,
        'boolean',
        true
    )

    local parsed = parseLink(options.link)
    local display = options.display or parsed.display

```

```

-- Deal with the case where we don't have to pipe the link
if not display and not parsed.section and not options.italicizePage
then
    return string.format('[[:%s]]', parsed.link)
end
-- Find the display text for piped links
if not display then
    local page = maybeItalicize(parsed.page,
options.italicizePage)
    local section = maybeItalicize(parsed.section,
options.italicizeSection)
    if not page then
        display = string.format('$&nbsp;%s', section)
    elseif section then
        display = string.format('%s $&nbsp;%s', page,
section)
    else
        display = page
    end
end
return string.format('[[:%s|%s]]', parsed.link, display)
end

```

```

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

```

```

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

```

```

function p._hatnote(s, options)
    checkType('_hatnote', 1, s, 'string')
    checkType('_hatnote', 2, options, 'table', true)
    options = options or {}

```

```

local classes = {'hatnote', 'navigation-not-searchable'}
local extraclasses = options.extraclasses
local selfref = options.selfref
if type(extraclasses) == 'string' then
    classes[#classes + 1] = extraclasses
end
if selfref then
    classes[#classes + 1] = 'selfref'
end
return string.format(
    '<div role="note" class="%s">%s</div>',
    table.concat(classes, ' '),
    s
)
end

return p

```

Retrieved from "<https://www.bluegoldwiki.com/index.php?title=Module:Hatnote&oldid=3588>"

Namespaces

- [Module](#)
- [Discussion](#)

Variants

This page was last edited on 15 November 2020, at 11:23.

Blue Gold Program Wiki

The wiki version of the Lessons Learnt Report of the Blue Gold program, documents the experiences of a technical assistance (TA) team working in a development project implemented by the Bangladesh Water Development Board (BWDB) and the Department of Agricultural Extension (DAE) over an eight+ year period from March 2013 to December 2021. The wiki lessons learnt report (LLR) is intended to complement the BWDB and DAE project completion reports (PCRs), with the aim of recording lessons learnt for use in the design and implementation of future interventions in the coastal zone.

- [Privacy policy](#)
- [About Blue Gold Program Wiki](#)
- [Disclaimers](#)

Developed and maintained by Big Blue Communications for Blue Gold Program



[Blue Gold Program Wiki](#)