Modul:Hatnote list

Vorlage:Used in system

Inhaltsverzeichnis

Usage from wikitext

This module is not designed be used directly from wikitext even though `forSee` does take an initial `frame` argument and could potentially be used from wikitext, e.g.:

- `{{hatnote|PREFIX {{#invoke:Hatnote list|forSee|{{tl|For}}|Module:For|{{tl|About}}|Module:About}} POSTFIX}}` → Vorlage:Hatnote

Usage from Lua

To call the module, use

```
local mHatList = require('Module:Hatnote list')
```

or similar, then access its methods through the `mHatList` variable (or whatever was used).

andList

`andList` takes a list in table form, and returns a string with the list separated with "and" and commas as appropriate.

orList

`orList` takes a list in table form, and returns a string with the list separated with "or" and commas as appropriate.

forSee

`_forSee` takes three arguments: a table of trimmed arguments with blanks removed, a "from" number with the index to start at, and an options table, and returns a string with a number of "`For X, see [[Y]]`" sentences. The links are formatted using the methods from Module:Hatnote.

As a convenience, the `forSee` method (without the leading underscore) takes the same arguments except with a frame instead of an args table, using `getArgs()` from Module:Arguments to preprocess the arguments.

```
--------------------------------------------------------------------------------
--                            Module:Hatnote list                             --
--                                                                            --
-- This module produces and formats lists for use in hatnotes. In particular, --
-- it implements the for-see list, i.e. lists of "For X, see Y" statements,   --
-- as used in {{about}}, {{redirect}}, and their variants. Also introduced    --
-- are andList & orList helpers for formatting lists with those conjunctions. --
--------------------------------------------------------------------------------

local mArguments --initialize lazily
local mFormatLink = require('Module:Format link')
local mHatnote = require('Module:Hatnote')
local libraryUtil = require('libraryUtil')
local checkType = libraryUtil.checkType
local p = {}

--------------------------------------------------------------------------------
-- List stringification helper functions
--
-- These functions are used for stringifying lists, usually page lists inside
-- the "Y" portion of "For X, see Y" for-see items.
--------------------------------------------------------------------------------

--default options table used across the list stringification functions
local stringifyListDefaultOptions = {
        conjunction = "and",
        separator = ",",
        altSeparator = ";",
        space = " ",
        formatted = false
}

--Searches display text only
local function searchDisp(haystack, needle)
        return string.find(
                string.sub(haystack, (string.find(haystack, '|') or 0) + 1), neec
        )
end

-- Stringifies a list generically; probably shouldn't be used directly
local function stringifyList(list, options)
        -- Type-checks, defaults, and a shortcut
        checkType("stringifyList", 1, list, "table")
        if #list == 0 then return nil end
        checkType("stringifyList", 2, options, "table", true)
        options = options or {}
        for k, v in pairs(stringifyListDefaultOptions) do
                if options[k] == nil then options[k] = v end
        end
        local s = options.space
        -- Format the list if requested
        if options.formatted then
                list = mFormatLink.formatPages(
                        {categorizeMissing = mHatnote.missingTargetCat}, list
                )
        end
        -- Set the separator; if any item contains it, use the alternate separato
```

```
	local separator = options.separator
	for k, v in pairs(list) do
		if searchDisp(v, separator) then
			separator = options.altSeparator
			break
		end
	end
	-- Set the conjunction, apply Oxford comma, and force a comma if #1 has '
	local conjunction = s .. options.conjunction .. s
	if #list == 2 and searchDisp(list[1], "§") or #list > 2 then
		conjunction = separator .. conjunction
	end
	-- Return the formatted string
	return mw.text.listToText(list, separator .. s, conjunction)
end

--DRY function
function p.conjList (conj, list, fmt)
	return stringifyList(list, {conjunction = conj, formatted = fmt})
end

-- Stringifies lists with "and" or "or"
function p.andList (...) return p.conjList("and", ...) end
function p.orList (...) return p.conjList("or", ...) end

--------------------------------------------------------------------------------
-- For see
--
-- Makes a "For X, see [[Y]]." list from raw parameters. Intended for the
-- {{about}} and {{redirect}} templates and their variants.
--------------------------------------------------------------------------------

--default options table used across the forSee family of functions
local forSeeDefaultOptions = {
	andKeyword = 'and',
	title = mw.title.getCurrentTitle().text,
	otherText = 'other uses',
	forSeeForm = 'For %s, see %s.',
}

--Collapses duplicate punctuation
local function punctuationCollapse (text)
	local replacements = {
		["%.%.$"] = ".",
		["%?%.$"] = "?",
		["%!%.$"] = "!",
		["%.%]%]%.$"] = ".]]",
		["%?%]%]%.$"] = "?]]",
		["%!%]%]%.$"] = "!]]"
	}
	for k, v in pairs(replacements) do text = string.gsub(text, k, v) end
	return text
end

-- Structures arguments into a table for stringification, & options
function p.forSeeArgsToTable (args, from, options)
	-- Type-checks and defaults
	checkType("forSeeArgsToTable", 1, args, 'table')
	checkType("forSeeArgsToTable", 2, from, 'number', true)
	from = from or 1
	checkType("forSeeArgsToTable", 3, options, 'table', true)
	options = options or {}
	for k, v in pairs(forSeeDefaultOptions) do
		if options[k] == nil then options[k] = v end
```

```
	end
	-- maxArg's gotten manually because getArgs() and table.maxn aren't frier
	local maxArg = 0
	for k, v in pairs(args) do
		if type(k) == 'number' and k > maxArg then maxArg = k end
	end
	-- Structure the data out from the parameter list:
	-- * forTable is the wrapper table, with forRow rows
	-- * Rows are tables of a "use" string & a "pages" table of pagename stri
	-- * Blanks are left empty for defaulting elsewhere, but can terminate li
	local forTable = {}
	local i = from
	local terminated = false
	-- If there is extra text, and no arguments are given, give nil value
	-- to not produce default of "For other uses, see foo (disambiguation)"
	if options.extratext and i > maxArg then return nil end
	-- Loop to generate rows
	repeat
		-- New empty row
		local forRow = {}
		-- On blank use, assume list's ended & break at end of this loop
		forRow.use = args[i]
		if not args[i] then terminated = true end
		-- New empty list of pages
		forRow.pages = {}
		-- Insert first pages item if present
		table.insert(forRow.pages, args[i + 1])
		-- If the param after next is "and", do inner loop to collect par
		-- until the "and"'s stop. Blanks are ignored: "1|and||and|3" → {
		while args[i + 2] == options.andKeyword do
			if args[i + 3] then
				table.insert(forRow.pages, args[i + 3])
			end
			-- Increment to next "and"
			i = i + 2
		end
		-- Increment to next use
		i = i + 2
		-- Append the row
		table.insert(forTable, forRow)
	until terminated or i > maxArg

	return forTable
end

-- Stringifies a table as formatted by forSeeArgsToTable
function p.forSeeTableToString (forSeeTable, options)
	-- Type-checks and defaults
	checkType("forSeeTableToString", 1, forSeeTable, "table", true)
	checkType("forSeeTableToString", 2, options, "table", true)
	options = options or {}
	for k, v in pairs(forSeeDefaultOptions) do
		if options[k] == nil then options[k] = v end
	end
	-- Stringify each for-see item into a list
	local strList = {}
	if forSeeTable then
		for k, v in pairs(forSeeTable) do
			local useStr = v.use or options.otherText
			local pagesStr =
				p.andList(v.pages, true) or
				mFormatLink._formatLink{
					categorizeMissing = mHatnote.missingTarge
					link = mHatnote.disambiguate(options.titl
```

```
                                }
                        local forSeeStr = string.format(options.forSeeForm, useSt
                        forSeeStr = punctuationCollapse(forSeeStr)
                        table.insert(strList, forSeeStr)
                end
        end
        if options.extratext then table.insert(strList, punctuationCollapse(optic
        -- Return the concatenated list
        return table.concat(strList, ' ')
end

-- Produces a "For X, see [[Y]]" string from arguments. Expects index gaps
-- but not blank/whitespace values. Ignores named args and args < "from".
function p._forSee (args, from, options)
        local forSeeTable = p.forSeeArgsToTable(args, from, options)
        return p.forSeeTableToString(forSeeTable, options)
end

-- As _forSee, but uses the frame.
function p.forSee (frame, from, options)
        mArguments = require('Module:Arguments')
        return p._forSee(mArguments.getArgs(frame), from, options)
end

return p
```