Inhaltsverzeichnis

1. Modul:Shortcut 2
2. Modul:Shortcut/config 6
3. Modul:Yesno 7

Modul:Shortcut

This module makes a box showing the shortcut links to a page.

Inhaltsverzeichnis

1 Usage 2
1.1 From wikitext 2
1.2 From Lua 2
2 Technical details 2

Usage

From wikitext

From wikitext, this module should be called from a template, usually Vorlage:Tl. Please see the template page for documentation. However, it can also be called using the syntax `{{#invoke:shortcut|main|`*`arguments`*`}}`.

From Lua

To use this module from Lua, first load it.

```
local mShortcut = require('Module:Shortcut')
```

Then you can create shortcut boxes with the following syntax:

```
mShortcut._main(shortcuts, options, frame, cfg)
```

- *shortcuts* is an array of shortcut page names. (required)
- *options* is a table of options. The following keys are supported:
  - `msg` - a message to leave after the list of shortcuts.
  - `category` - if set to false (or a value regarded as false by Module:Yesno, such as "no"), categories are suppressed.
- *frame* a frame object. This is optional, and only intended to be used internally.
- *cfg* a table of config values. This is optional, and is only intended for testing.

Technical details

This module has a configuration file at Module:Shortcut/config. It can be used to translate this module into different languages or to change details like category names.

```
-- This module implements {{shortcut}}.

-- Set constants
local CONFIG_MODULE = 'Module:Shortcut/config'

-- Load required modules
local checkType = require('libraryUtil').checkType
local yesno = require('Module:Yesno')

local p = {}

local function message(msg, ...)
	return mw.message.newRawMessage(msg, ...):plain()
end

local function makeCategoryLink(cat)
	return string.format('[[%s:%s]]', mw.site.namespaces[14].name, cat)
end

function p._main(shortcuts, options, frame, cfg)
	checkType('_main', 1, shortcuts, 'table')
	checkType('_main', 2, options, 'table', true)
	options = options or {}
	frame = frame or mw.getCurrentFrame()
	cfg = cfg or mw.loadData(CONFIG_MODULE)
	local templateMode = options.template and yesno(options.template)
	local redirectMode = options.redirect and yesno(options.redirect)
	local isCategorized = not options.category or yesno(options.category) ~=

	-- Validate shortcuts
	for i, shortcut in ipairs(shortcuts) do
		if type(shortcut) ~= 'string' or #shortcut < 1 then
			error(message(cfg['invalid-shortcut-error'], i), 2)
		end
	end

	-- Make the list items. These are the shortcuts plus any extra lines such
	-- as options.msg.
	local listItems = {}
	for i, shortcut in ipairs(shortcuts) do
		local templatePath, prefix
		if templateMode then
			-- Namespace detection
			local titleObj = mw.title.new(shortcut, 10)
			if titleObj.namespace == 10 then
				templatePath = titleObj.fullText
			else
				templatePath = shortcut
			end
			prefix = options['pre' .. i] or options.pre or ''
		end
		if options.target and yesno(options.target) then
			listItems[i] = templateMode
				and string.format("&#123;&#123;%s[[%s|%s]]&#125;&
				or string.format("[[%s]]", shortcut)
		else
			listItems[i] = frame:expandTemplate{
				title = 'No redirect',
				args = templateMode and {templatePath, shortcut}
			}
			if templateMode then
				listItems[i] = string.format("&#123;&#123;%s%s&#1
			end
		end
```

```
	end
	table.insert(listItems, options.msg)

	-- Return an error if we have nothing to display
	if #listItems < 1 then
		local msg = cfg['no-content-error']
		msg = string.format('<strong class="error">%s</strong>', msg)
		if isCategorized and cfg['no-content-error-category'] then
			msg = msg .. makeCategoryLink(cfg['no-content-error-categ
		end
		return msg
	end

	local root = mw.html.create()
	root:wikitext(frame:extensionTag{ name = 'templatestyles', args = { src =
	-- Anchors
	local anchorDiv = root
		:tag('div')
			:addClass('module-shortcutanchordiv')
	for i, shortcut in ipairs(shortcuts) do
		local anchor = mw.uri.anchorEncode(shortcut)
		anchorDiv:tag('span'):attr('id', anchor)
	end

	-- Shortcut heading
	local shortcutHeading
	do
		local nShortcuts = #shortcuts
		if nShortcuts > 0 then
			local headingMsg = options['shortcut-heading'] or
				redirectMode and cfg['redirect-heading'] or
				cfg['shortcut-heading']
			shortcutHeading = message(headingMsg, nShortcuts)
			shortcutHeading = frame:preprocess(shortcutHeading)
		end
	end

	-- Shortcut box
	local shortcutList = root
		:tag('div')
			:addClass('module-shortcutboxplain plainlist noprint')
			:attr('role', 'note')
	if options.float and options.float:lower() == 'left' then
		shortcutList:addClass('module-shortcutboxleft')
	end
	if options.clear and options.clear ~= '' then
		shortcutList:css('clear', options.clear)
	end
	if shortcutHeading then
		shortcutList
			:tag('div')
				:addClass('module-shortcutlist')
				:wikitext(shortcutHeading)
	end
	local list = shortcutList:tag('ul')
	for i, item in ipairs(listItems) do
		list:tag('li'):wikitext(item)
	end
	return tostring(root)
end

function p.main(frame)
	local args = require('Module:Arguments').getArgs(frame)
```

```
        -- Separate shortcuts from options
        local shortcuts, options = {}, {}
        for k, v in pairs(args) do
                if type(k) == 'number' then
                        shortcuts[k] = v
                else
                        options[k] = v
                end
        end

        -- Compress the shortcut array, which may contain nils.
        local function compressArray(t)
                local nums, ret = {}, {}
                for k in pairs(t) do
                        nums[#nums + 1] = k
                end
                table.sort(nums)
                for i, num in ipairs(nums) do
                        ret[i] = t[num]
                end
                return ret
        end
        shortcuts = compressArray(shortcuts)

        return p._main(shortcuts, options, frame)
end

return p
```

Modul:Shortcut/config

```
-- This module holds configuration data for [[Module:Shortcut]].

return {

-- The heading at the top of the shortcut box. It accepts the following parameter
-- $1 - the total number of shortcuts. (required)
['shortcut-heading'] = '[[Wikipedia:Shortcut|{{PLURAL:$1|Shortcut|Shortcuts}}]]',

-- The heading when |redirect=yes is given. It accepts the following parameter:
-- $1 - the total number of shortcuts. (required)
['redirect-heading'] = '[[Wikipedia:Redirect|{{PLURAL:$1|Redirect|Redirects}}]]',

-- The error message to display when a shortcut is invalid (is not a string, or
-- is the blank string). It accepts the following parameter:
-- $1 - the number of the shortcut in the argument list. (required)
['invalid-shortcut-error'] = 'shortcut #$1 was invalid (shortcuts must be ' ..
        'strings of at least one character in length)',

-- The error message to display when no shortcuts or other displayable content
-- were specified. (required)
['no-content-error'] = 'Error: no shortcuts were specified and the ' ..
        mw.text.nowiki('|msg=') ..
        ' parameter was not set.',

-- A category to add when the no-content-error message is displayed. (optional)
['no-content-error-category'] = 'Shortcut templates with missing parameters',
}
```

Modul:Yesno

This module provides a consistent interface for processing boolean or boolean-style string input. While Lua allows the `true` and `false` boolean values, wikicode templates can only express boolean values through strings such as "yes", "no", etc. This module processes these kinds of strings and turns them into boolean input for Lua to process. It also returns `nil` values as `nil`, to allow for distinctions between `nil` and `false`. The module also accepts other Lua structures as input, i.e. booleans, numbers, tables, and functions. If it is passed input that it does not recognise as boolean or `nil`, it is possible to specify a default value to return.

Inhaltsverzeichnis

1 Syntax 7
2 Usage 7
2.1 Undefined input ('foo') 8
2.2 Handling nil results 9

Syntax

```
yesno(value, default)
```

`value` is the value to be tested. Boolean input or boolean-style input (see below) always evaluates to either `true` or `false`, and `nil` always evaluates to `nil`. Other values evaluate to `default`.

Usage

First, load the module. Note that it can only be loaded from other Lua modules, not from normal wiki pages. For normal wiki pages you can use Vorlage:Tl instead.

```
local yesno = require('Module:Yesno')
```

Some input values always return `true`, and some always return `false`. `nil` values always return `nil`.

```
-- These always return true:
yesno('yes')
yesno('y')
yesno('true')
yesno('t')
yesno('1')
yesno(1)
yesno(true)

-- These always return false:
yesno('no')
```

```
yesno('n')
yesno('false')
yesno('f')
yesno('0')
yesno(0)
yesno(false)

-- A nil value always returns nil:
yesno(nil)
```

String values are converted to lower case before they are matched:

```
-- These always return true:
yesno('Yes')
yesno('YES')
yesno('yEs')
yesno('Y')
yesno('tRuE')

-- These always return false:
yesno('No')
yesno('NO')
yesno('nO')
yesno('N')
yesno('fALsE')
```

Undefined input ('foo')

You can specify a default value if yesno receives input other than that listed above. If you don't supply a default, the module will return `nil` for these inputs.

```
-- These return nil:
yesno('foo')
yesno({})
yesno(5)
yesno(function() return 'This is a function.' end)
yesno(nil, true)
yesno(nil, 'bar')

-- These return true:
yesno('foo', true)
yesno({}, true)
yesno(5, true)
yesno(function() return 'This is a function.' end, true)

-- These return "bar":
yesno('foo', 'bar')
yesno({}, 'bar')
yesno(5, 'bar')
yesno(function() return 'This is a function.' end, 'bar')
```

Note that the empty string also functions this way:

```
yesno('')        -- Returns nil.
yesno('', true)  -- Returns true.
yesno('', 'bar') -- Returns "bar".
```

Although the empty string usually evaluates to false in wikitext, it evaluates to true in Lua. This module prefers the Lua behaviour over the wikitext behaviour. If treating the empty string as false is important for your module, you will need to convert empty strings to a value that evaluates to false before passing them to this module. In the case of arguments received from wikitext, this can be done by using Module:Arguments.

Handling nil results

By definition

```
yesno(nil)         -- Returns nil.
yesno('foo')       -- Returns nil.
yesno(nil, true)   -- Returns nil.
yesno(nil, false)  -- Returns nil.
yesno('foo', true) -- Returns true.
```

To get the binary `true/false`-only values, use code like:

```
myvariable = yesno(value) or false -- When value is nil, result is false.
myvariable = yesno(value) or true  -- When value is nil, result is true.
myvariable = yesno('foo') or false  -- Unknown string returns nil, result is fals
myvariable = yesno('foo', true) or false  -- Default value (here: true) applies,
```

```
-- Function allowing for consistent treatment of boolean-like wikitext input.
-- It works similarly to the template {{yesno}}.

return function (val, default)
        -- If your wiki uses non-ascii characters for any of "yes", "no", etc., y
        -- should replace "val:lower()" with "mw.ustring.lower(val)" in the
        -- following line.
        val = type(val) == 'string' and val:lower() or val
        if val == nil then
                return nil
        elseif val == true
                or val == 'yes'
                or val == 'y'
                or val == 'true'
                or val == 't'
                or val == 'on'
                or tonumber(val) == 1
        then
                return true
        elseif val == false
                or val == 'no'
                or val == 'n'
                or val == 'false'
                or val == 'f'
                or val == 'off'
                or tonumber(val) == 0
```

```
        then
                return false
        else
                return default
        end
end
```