Inhaltsverzeichnis

Modul:File link

Vorlage:Lua This module is used to construct wikitext links to files. It is primarily useful for templates and modules that use complicated logic to make file links. Simple file links should be made with wikitext markup directly, as it uses less resources than calling this module. For help with wikitext file markup please refer to the documentation at mediawiki.org.

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:File link|main|`*`arguments`*`}}`.

Usage from Lua

First, you need to import the module.

```
local mFileLink = require('Module:File link')
```

Then you can make file links using the `_main` function.

```
mFileLink._main(args)
```

args is a table of arguments that can have the following keys:

- `file` - the filename. (required)
- `format` - the file format, e.g. 'thumb', 'thumbnail', 'frame', 'framed', or 'frameless'.
- `formatfile` - a filename to specify with the 'thumbnail' format option. The filename specified will be used instead of the automatically generated thumbnail.
- `border` - set this to true or "yes" (or any other value recognized as true by Module:Yesno) to set a border for the image.
- `location` - the horizontal alignment of the file, e.g. 'right', 'left', 'center', or 'none'.
- `alignment` - the vertical alignment of the file, e.g. 'baseline', 'middle', 'sub', 'super', 'text-top', 'text-bottom', 'top', or 'bottom'.
- `size` - the size of the image, e.g. '100px', 'x100px' or '100x100px'.
- `upright` - the 'upright' parameter, used for setting the size of tall and thin images.
- `link` - the page that the file should link to. Use the blank string '' to suppress the default link to the file description page.
- `alt` - the alt text. Use the blank string '' to suppress the default alt text.
- `caption` - a caption for the file.
- `page` - sets a page number for multi-paged files such as PDFs.

- `class` - adds a `class` parameter to image links. The MediaWiki software adds this parameter to the `class="..."` attribute of the image's `<img />` element when the page is rendered into HTML.
- `lang` - adds a language attribute to specify what language to render the file in.
- `start` - specifies a start time for audio and video files.
- `end` - specifies an end time for audio and video files.
- `thumbtime` - specifies the time to use to generate the thumbnail image for video files.

To see the effect of each of these parameters, see the images help page on mediawiki.org.

Examples

With the file only:

```
mFileLink.main{file = 'Example.png'}
-- Renders as [[File:Example.png]]
```

With format, size, link and caption options:

```
mFileLink.main{
        file = 'Example.png',
        format = 'thumb',
        size = '220px',
        link = 'Wikipedia:Sandbox',
        caption = 'An example.'
}
-- Renders as [[File:Example.png|thumb|220px|link=Wikipedia:Sandbox|An example.]]
```

With format, size, and border:

```
mFileLink.main{
        file = 'Example.png',
        format = 'frameless',
        size = '220px',
        border = true
}
-- Renders as [[File:Example.png|frameless|border|220px]]
```

```
-- This module provides a library for formatting file wikilinks.

local yesno = require('Module:Yesno')
local checkType = require('libraryUtil').checkType

local p = {}

function p._main(args)
        checkType('_main', 1, args, 'table')

        -- This is basically libraryUtil.checkTypeForNamedArg, but we are rolling
        -- own function to get the right error level.
```

```
	local function checkArg(key, val, level)
		if type(val) ~= 'string' then
			error(string.format(
				"type error in '%s' parameter of '_main' (expecte
				key, type(val)
			), level)
		end
	end

	local ret = {}

	-- Adds a positional parameter to the buffer.
	local function addPositional(key)
		local val = args[key]
		if not val then
			return nil
		end
		checkArg(key, val, 4)
		ret[#ret + 1] = val
	end

	-- Adds a named parameter to the buffer. We assume that the parameter nam
	-- is the same as the argument key.
	local function addNamed(key)
		local val = args[key]
		if not val then
			return nil
		end
		checkArg(key, val, 4)
		ret[#ret + 1] = key .. '=' .. val
	end

	-- Filename
	checkArg('file', args.file, 3)
	ret[#ret + 1] = 'File:' .. args.file

	-- Format
	if args.format then
		checkArg('format', args.format)
		if args.formatfile then
			checkArg('formatfile', args.formatfile)
			ret[#ret + 1] = args.format .. '=' .. args.formatfile
		else
			ret[#ret + 1] = args.format
		end
	end

	-- Border
	if yesno(args.border) then
		ret[#ret + 1] = 'border'
	end

	addPositional('location')
	addPositional('alignment')
	addPositional('size')
	addNamed('upright')
	addNamed('link')
	addNamed('alt')
	addNamed('page')
	addNamed('class')
	addNamed('lang')
	addNamed('start')
	addNamed('end')
	addNamed('thumbtime')
```

```
        addPositional('caption')

        return string.format('[[%s]]', table.concat(ret, '|'))
end

function p.main(frame)
        local origArgs = require('Module:Arguments').getArgs(frame, {
                wrappers = 'Template:File link'
        })
        if not origArgs.file then
                error("'file' parameter missing from [[Template:File link]]", 0)
        end

        -- Copy the arguments that were passed to a new table to avoid looking up
        -- every possible parameter in the frame object.
        local args = {}
        for k, v in pairs(origArgs) do
                -- Make _BLANK a special argument to add a blank parameter. For u
                -- conditional templates etc. it is useful for blank arguments to
                -- ignored, but we still need a way to specify them so that we ca
                -- things like [[File:Example.png|link=]].
                if v == '_BLANK' then
                        v = ''
                end
                args[k] = v
        end
        return p._main(args)
end

return p
```

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

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
```