



Modul:Submit an edit request

Vorlage:Lua This module implements the **Vorlage:TI** and **Vorlage:TI** templates.

Usage from wikitext

To use this template from wikitext, you should normally use the **Vorlage:TI** and **Vorlage:TI** templates. However, the module can also be used directly from `#invoke`. For the edit request button, use `{{#invoke:Submit an edit request|button|args}}`, and for the edit request link only, use `{{#invoke:Submit an edit request|link|args}}`. Please see the respective template pages for a list of available parameters.

Usage from Lua modules

To use this module from other Lua modules, first load the module.

```
local mEditRequest = require('Module:Submit an edit request')
```

You can then use the `_button` function to generate an edit request button, and the `_link` function to generate an edit request link.

```
mEditRequest._button(args)
mEditRequest._link(args)
```

The *args* variable should be a table containing the arguments to pass to the module. To see the different arguments that can be specified and how they affect the module output, please refer to the documentation of **Vorlage:TI** and **Vorlage:TI**.

Configuration

This module can be translated and configured for other wikis by editing **Module:Submit an edit request/config**.

```
-- This module implements {{Submit an edit request}}.
local CONFIG_MODULE = 'Module:Submit an edit request/config'

-- Load necessary modules
local mRedirect = require('Module:Redirect')
local cfg = mw.loadData(CONFIG_MODULE)
local effectiveProtectionLevel = require('Module:Effective protection level')._m
local escape = require("Module:String")._escapePattern
local lang = mw.language.getContentLanguage()

local p = {}
```



```
local validLevels = {
    semi = 'semi',
    extended = 'extended',
    template = 'template',
    full = 'full',
    interface = 'interface',
    manual = 'manual'
}

local function message(key, ...)
    local params = {...}
    local msg = cfg[key]
    if #params < 1 then
        return msg
    else
        return mw.message.newRawMessage(msg):params(params):plain()
    end
end

local function validateLevel(level)
    return level and validLevels[level] or 'full'
end

local function getLevelInfo(level, field)
    return cfg.protectionLevels[level][field]
end

local function resolveRedirect(page)
    return mRedirect.luaMain(page)
end

local function isProtected(page)
    local action = mw.title.new(page).exists and 'edit' or 'create'
    return effectiveProtectionLevel(action, page) ~= '*'
end

function p.makeRequestUrl(level, titleObj)
    titleObj = titleObj or mw.title.getCurrentTitle()
    local basePage = titleObj.basePageTitle.fullText
    if cfg['main-page-content'][basePage] then
        return tostring(mw.uri.fullUrl(message('main-page-request-page')))
    end

    local talkPageName = titleObj.talkPageTitle
    if talkPageName == nil then
        return tostring(mw.uri.fullUrl(message('protected-talk-page-reqe
    end
    talkPageName = resolveRedirect(talkPageName.prefixedText)
    if isProtected(talkPageName) then
        return tostring(mw.uri.fullUrl(message('protected-talk-page-reqe
    end
    level = validateLevel(level)
    if level == 'manual' then
        return tostring(mw.uri.fullUrl(talkPageName, {
            action = 'edit',
            section = 'new'
        })))
    end
    local sectionname = message(
        'preload-title-text',
        getLevelInfo(level, 'levelText'),
        lang:formatDate(message('preload-title-date-format'))
    )
    local content = mw.title.new(talkPageName):getContent()
```

```
        if content and content:find("== *" .. escape(sectionname) .. " *==") then
            local dedup = 2
            while true do
                local newname = message("preload-title-dedup-suffix", sectionname, dedup)
                if not content:find("== *" .. escape(newname) .. " *==") then
                    sectionname = newname
                    break
                end
                dedup = dedup + 1
            end
        end
        local url = mw.uri.fullUrl(talkPageName, {
            action = 'edit',
            editintro = getLevelInfo(level, 'editintro'),
            preload = message('preload-template'),
            preloadtitle = sectionname,
            section = 'new'
        })
        url = tostring(url)

        -- Add the preload parameters. @TODO: merge this into the mw.uri.fullUrl
        -- query table once [[phab:T93059]] is fixed.
        local function encodeParam(key, val)
            return string.format('&%s=%s', mw.uri.encode(key), mw.uri.encode(val))
        end
        url = url .. encodeParam('preloadparams[]', getLevelInfo(level, 'requestname'))
        url = url .. encodeParam('preloadparams[]', titleObj.prefixedText)

        return url
    end

    function p._link(args)
        return string.format(
            '<span class="plainlinks">[%s %s]</span>',
            p.makeRequestUrl(args.type),
            args.display or message('default-display-value')
        )
    end

    function p._button(args)
        return require('Module:Clickable button 2').luaMain{
            [1] = args.display or message('default-display-value'),
            url = p.makeRequestUrl(args.type),
            class = 'mw-ui-progressive'
        }
    end

    local function makeInvokeFunc(func, wrapper)
        return function (frame)
            local args = require('Module:Arguments').getArgs(frame, {
                wrappers = {wrapper}
            })
            return func(args)
        end
    end

    p.link = makeInvokeFunc(p._link, message('link-wrapper-template'))
    p.button = makeInvokeFunc(p._button, message('button-wrapper-template'))

    return p
end
```