Modul:Protection banner

Vorlage:Lua This module creates protection banners and padlock icons that are placed at the top of protected pages.

Inhaltsverzeichnis

Usage

Most users will not need to use this module directly. For adding protection templates to pages you can use the Vorlage:Tl template, or you may find it more convenient to use one of the more specific protection templates in the table below.

Vorlage:Protection templates

From wikitext

```
{{#invoke:Protection banner|main
| 1        = reason
| small    = yes/no
| action   = action
| date     = protection date
| user     = username
| section  = talk page section name
| category = no
}}
```

The #invoke syntax can be used for creating protection templates more specific than Vorlage:Tl. For example, it is possible to create a protection template which always shows a padlock icon by using the code `{{#invoke:Protection banner|main|small=yes}}`. Pages which call this template will still be able to use other arguments, like *action*. However, this only works one level deep; a page calling a template which calls another template containing the above code will not automatically be able to use parameters like *action*.

Note: *You should no longer specify the expiry, as it is automatically retrieved in all cases.*

From Lua

First, load the module.

```
local mProtectionBanner = require('Module:Protection banner')
```

Then you can make protection banners by using the _main function.

```
mProtectionBanner._main(args, cfg, titleObj)
```

args is a table of arguments to pass to the module. For possible keys and values for this table, see the parameters section. The *cfg* and *titleObj* variables are intended only for testing; *cfg* specifies a customised config table to use instead of Module:Protection banner/config, and *titleObj* specifies a mw.title object to use instead of the current title. *args*, *cfg* and *titleObj* are all optional.

Parameters

All parameters are optional.

- **1** – the reason that the page was protected. If set, this must be one of the values listed in the reasons table.
- **small** – if set to "yes", "y", "1", or "true", a padlock icon is generated instead of a full protection banner.
- **action** – the protection action. Must be one of "edit" (for normal protection), "move" (for move-protection) or "autoreview" (for pending changes). The default value is "edit".
- **date** – the protection date. This must be valid input to the second parameter of the #time parser function. This argument has an effect for reasons that use the PROTECTIONDATE parameter in their configuration. As of July 2014, those were the "office" and "reset" reasons.
- **user** – the username of the user to generate links for. As of July 2014, this only has an effect when the "usertalk" reason is specified.
- **section** – the section name of the protected page's talk page where discussion is taking place. This works for most, but not all, values of *reason*.
- **category** – categories are suppressed if this is set to "no", "n", "0", or "false".
- **catonly** – if set to "yes", "y", "1", or "true", will only return the protection categories, and not return the banner or padlock. This has no visible output.

Reasons

The following table contains the available reasons, plus the actions for which they are available.

Skriptfehler: Ein solches Modul „Protection banner/documentation“ ist nicht vorhanden.

Errors

Below is a list of some of the common errors that this module can produce, and how to fix them.

Invalid protection date

Vorlage:Error

This error is produced if you supply a Vorlage:Para parameter value that is not recognised as a valid date by the #time parser function. If in doubt, you can just use a date in the format "dd Month YYYY", e.g. "4 Juli 2026". To see a full range of valid inputs, see the #time documentation (only the first parameter, the *format string*, may be specified).

Invalid action

Vorlage:Error

This error is produced if you specify an invalid protection action. There are only three valid actions: `edit` (the default, for normal protection), `move` (for move-protection), and `autoreview` (for pending changes). This should only be possible if you are using a template that supports manually specifying the protection action, such as Vorlage:Tl, or if you are using #invoke directly. If this is not the case, please leave a message on Module talk:Protection banner.

Reasons cannot contain the pipe character

Vorlage:Error

This error is produced if you specify a reason using the Vorlage:Para parameter that includes a pipe character ("|"). Please check that you are not entering the Vorlage:Tl template into this parameter by mistake. The pipe character is disallowed as the module uses it internally. A list of valid reasons can be seen in the reasons section.

Other errors

If you see an error other than the ones above, it is likely to either be a bug in the module or mistake in the configuration. Please post a message about it at Module talk:Protection banner.

Technical details

This module uses configuration data from Module:Protection banner/config. Most of the module's behaviour can be configured there, making it easily portable across different wikis and different languages.

General test cases for the module can be found at Module:Protection banner/testcases, and test cases specific to enwiki's config can be found at Module:Protection banner/config/testcases.

Bug reports and feature requests should be made on the module's talk page.

```
-- This module implements {{pp-meta}} and its daughter templates such as
-- {{pp-dispute}}, {{pp-vandalism}} and {{pp-sock}}.

-- Initialise necessary modules.
require('Module:No globals')
local makeFileLink = require('Module:File link')._main
local effectiveProtectionLevel = require('Module:Effective protection level')._ma
local effectiveProtectionExpiry = require('Module:Effective protection expiry')._
local yesno = require('Module:Yesno')

-- Lazily initialise modules and objects we don't always need.
local getArgs, makeMessageBox, lang

-- Set constants.
local CONFIG_MODULE = 'Module:Protection banner/config'

--------------------------------------------------------------------------------
-- Helper functions
--------------------------------------------------------------------------------

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

-- Validation function for the expiry and the protection date
local function validateDate(dateString, dateType)
	if not lang then
		lang = mw.language.getContentLanguage()
	end
	local success, result = pcall(lang.formatDate, lang, 'U', dateString)
	if success then
		result = tonumber(result)
		if result then
			return result
		end
	end
	error(string.format(
		'invalid %s: %s',
		dateType,
		tostring(dateString)
	), 4)
end

local function makeFullUrl(page, query, display)
	return string.format(
		'[%s %s]',
		tostring(mw.uri.fullUrl(page, query)),
		display
	)
end

-- Given a directed graph formatted as node -> table of direct successors,
-- get a table of all nodes reachable from a given node (though always
-- including the given node).
local function getReachableNodes(graph, start)
	local toWalk, retval = {[start] = true}, {}
	while true do
```

```
                -- Can't use pairs() since we're adding and removing things as we
                local k = next(toWalk) -- This always gets the "first" key
                if k == nil then
                        return retval
                end
                toWalk[k] = nil
                retval[k] = true
                for _,v in ipairs(graph[k]) do
                        if not retval[v] then
                                toWalk[v] = true
                        end
                end
        end
end

-------------------------------------------------------------------------------
-- Protection class
-------------------------------------------------------------------------------

local Protection = {}
Protection.__index = Protection

Protection.supportedActions = {
        edit = true,
        move = true,
        autoreview = true,
        upload = true
}

Protection.bannerConfigFields = {
        'text',
        'explanation',
        'tooltip',
        'alt',
        'link',
        'image'
}

function Protection.new(args, cfg, title)
        local obj = {}
        obj._cfg = cfg
        obj.title = title or mw.title.getCurrentTitle()

        -- Set action
        if not args.action then
                obj.action = 'edit'
        elseif Protection.supportedActions[args.action] then
                obj.action = args.action
        else
                error(string.format(
                        'invalid action: %s',
                        tostring(args.action)
                ), 3)
        end

        -- Set level
        obj.level = args.demolevel or effectiveProtectionLevel(obj.action, obj.ti
        if not obj.level or (obj.action == 'move' and obj.level == 'autoconfirmed
                -- Users need to be autoconfirmed to move pages anyway, so treat
                -- semi-move-protected pages as unprotected.
                obj.level = '*'
        end

        -- Set expiry
```

```
	local effectiveExpiry = effectiveProtectionExpiry(obj.action, obj.title)
	if effectiveExpiry == 'infinity' then
		obj.expiry = 'indef'
	elseif effectiveExpiry ~= 'unknown' then
		obj.expiry = validateDate(effectiveExpiry, 'expiry date')
	end

	-- Set reason
	if args[1] then
		obj.reason = mw.ustring.lower(args[1])
		if obj.reason:find('|') then
			error('reasons cannot contain the pipe character ("|")',
		end
	end

	-- Set protection date
	if args.date then
		obj.protectionDate = validateDate(args.date, 'protection date')
	end

	-- Set banner config
	do
		obj.bannerConfig = {}
		local configTables = {}
		if cfg.banners[obj.action] then
			configTables[#configTables + 1] = cfg.banners[obj.action]
		end
		if cfg.defaultBanners[obj.action] then
			configTables[#configTables + 1] = cfg.defaultBanners[obj
			configTables[#configTables + 1] = cfg.defaultBanners[obj
		end
		configTables[#configTables + 1] = cfg.masterBanner
		for i, field in ipairs(Protection.bannerConfigFields) do
			for j, t in ipairs(configTables) do
				if t[field] then
					obj.bannerConfig[field] = t[field]
					break
				end
			end
		end
	end
	return setmetatable(obj, Protection)
end

function Protection:isUserScript()
	-- Whether the page is a user JavaScript or CSS page.
	local title = self.title
	return title.namespace == 2 and (
		title.contentModel == 'javascript' or title.contentModel == 'css'
	)
end

function Protection:isProtected()
	return self.level ~= '*'
end

function Protection:shouldShowLock()
	-- Whether we should output a banner/padlock
	return self:isProtected() and not self:isUserScript()
end

-- Whether this page needs a protection category.
Protection.shouldHaveProtectionCategory = Protection.shouldShowLock
```

```
function Protection:isTemporary()
	return type(self.expiry) == 'number'
end

function Protection:makeProtectionCategory()
	if not self:shouldHaveProtectionCategory() then
		return ''
	end

	local cfg = self._cfg
	local title = self.title

	-- Get the expiry key fragment.
	local expiryFragment
	if self.expiry == 'indef' then
		expiryFragment = self.expiry
	elseif type(self.expiry) == 'number' then
		expiryFragment = 'temp'
	end

	-- Get the namespace key fragment.
	local namespaceFragment = cfg.categoryNamespaceKeys[title.namespace]
	if not namespaceFragment and title.namespace % 2 == 1 then
		namespaceFragment = 'talk'
	end

	-- Define the order that key fragments are tested in. This is done with a
	-- array of tables containing the value to be tested, along with its
	-- position in the cfg.protectionCategories table.
	local order = {
		{val = expiryFragment,    keypos = 1},
		{val = namespaceFragment, keypos = 2},
		{val = self.reason,       keypos = 3},
		{val = self.level,        keypos = 4},
		{val = self.action,       keypos = 5}
	}

	--[[
	-- The old protection templates used an ad-hoc protection category system
	-- with some templates prioritising namespaces in their categories, and
	-- others prioritising the protection reason. To emulate this in this mod
	-- we use the config table cfg.reasonsWithNamespacePriority to set the
	-- reasons for which namespaces have priority over protection reason.
	-- If we are dealing with one of those reasons, move the namespace table
	-- the end of the order table, i.e. give it highest priority. If not, the
	-- reason should have highest priority, so move that to the end of the ta
	-- instead.
	--]]
	table.insert(order, table.remove(order, self.reason and cfg.reasonsWithNa

	--[[
	-- Define the attempt order. Inactive subtables (subtables with nil "valu
	-- fields) are moved to the end, where they will later be given the key
	-- "all". This is to cut down on the number of table lookups in
	-- cfg.protectionCategories, which grows exponentially with the number of
	-- non-nil keys. We keep track of the number of active subtables with the
	-- noActive parameter.
	--]]
	local noActive, attemptOrder
	do
		local active, inactive = {}, {}
		for i, t in ipairs(order) do
			if t.val then
				active[#active + 1] = t
```

```
                        else
                                inactive[#inactive + 1] = t
                        end
                end
                noActive = #active
                attemptOrder = active
                for i, t in ipairs(inactive) do
                        attemptOrder[#attemptOrder + 1] = t
                end
        end

        --[[
        -- Check increasingly generic key combinations until we find a match. If
        -- specific category exists for the combination of key fragments we are
        -- given, that match will be found first. If not, we keep trying differer
        -- key fragment combinations until we match using the key
        -- "all-all-all-all-all".
        --
        -- To generate the keys, we index the key subtables using a binary matrix
        -- with indexes i and j. j is only calculated up to the number of active
        -- subtables. For example, if there were three active subtables, the matr
        -- would look like this, with 0 corresponding to the key fragment "all",
        -- 1 corresponding to other key fragments.
        --
        --   j 1  2  3
        -- i
        -- 1   1  1  1
        -- 2   0  1  1
        -- 3   1  0  1
        -- 4   0  0  1
        -- 5   1  1  0
        -- 6   0  1  0
        -- 7   1  0  0
        -- 8   0  0  0
        --
        -- Values of j higher than the number of active subtables are set
        -- to the string "all".
        --
        -- A key for cfg.protectionCategories is constructed for each value of i.
        -- The position of the value in the key is determined by the keypos field
        -- each subtable.
        --]]
        local cats = cfg.protectionCategories
        for i = 1, 2^noActive do
                local key = {}
                for j, t in ipairs(attemptOrder) do
                        if j > noActive then
                                key[t.keypos] = 'all'
                        else
                                local quotient = i / 2 ^ (j - 1)
                                quotient = math.ceil(quotient)
                                if quotient % 2 == 1 then
                                        key[t.keypos] = t.val
                                else
                                        key[t.keypos] = 'all'
                                end
                        end
                end
                key = table.concat(key, '|')
                local attempt = cats[key]
                if attempt then
                        return makeCategoryLink(attempt, title.text)
                end
        end
```

```
        return ''
end

function Protection:isIncorrect()
        local expiry = self.expiry
        return not self:shouldHaveProtectionCategory()
                or type(expiry) == 'number' and expiry < os.time()
end

function Protection:isTemplateProtectedNonTemplate()
        local action, namespace = self.action, self.title.namespace
        return self.level == 'templateeditor'
                and (
                        (action ~= 'edit' and action ~= 'move')
                        or (namespace ~= 10 and namespace ~= 828)
                )
end

function Protection:makeCategoryLinks()
        local msg = self._cfg.msg
        local ret = {self:makeProtectionCategory()}
        if self:isIncorrect() then
                ret[#ret + 1] = makeCategoryLink(
                        msg['tracking-category-incorrect'],
                        self.title.text
                )
        end
        if self:isTemplateProtectedNonTemplate() then
                ret[#ret + 1] = makeCategoryLink(
                        msg['tracking-category-template'],
                        self.title.text
                )
        end
        return table.concat(ret)
end

--------------------------------------------------------------------------------
-- Blurb class
--------------------------------------------------------------------------------

local Blurb = {}
Blurb.__index = Blurb

Blurb.bannerTextFields = {
        text = true,
        explanation = true,
        tooltip = true,
        alt = true,
        link = true
}

function Blurb.new(protectionObj, args, cfg)
        return setmetatable({
                _cfg = cfg,
                _protectionObj = protectionObj,
                _args = args
        }, Blurb)
end

-- Private methods --

function Blurb:_formatDate(num)
        -- Formats a Unix timestamp into dd Month, YYYY format.
        lang = lang or mw.language.getContentLanguage()
```

```
	local success, date = pcall(
		lang.formatDate,
		lang,
		self._cfg.msg['expiry-date-format'] or 'j F Y',
		'@' .. tostring(num)
	)
	if success then
		return date
	end
end

function Blurb:_getExpandedMessage(msgKey)
	return self:_substituteParameters(self._cfg.msg[msgKey])
end

function Blurb:_substituteParameters(msg)
	if not self._params then
		local parameterFuncs = {}

		parameterFuncs.CURRENTVERSION     = self._makeCurrentVersionParam
		parameterFuncs.EDITREQUEST        = self._makeEditRequestParamete
		parameterFuncs.EXPIRY             = self._makeExpiryParameter
		parameterFuncs.EXPLANATIONBLURB   = self._makeExplanationBlurbPar
		parameterFuncs.IMAGELINK          = self._makeImageLinkParameter
		parameterFuncs.INTROBLURB         = self._makeIntroBlurbParameter
		parameterFuncs.INTROFRAGMENT      = self._makeIntroFragmentParame
		parameterFuncs.PAGETYPE           = self._makePagetypeParameter
		parameterFuncs.PROTECTIONBLURB    = self._makeProtectionBlurbPara
		parameterFuncs.PROTECTIONDATE     = self._makeProtectionDateParam
		parameterFuncs.PROTECTIONLEVEL    = self._makeProtectionLevelPara
		parameterFuncs.PROTECTIONLOG      = self._makeProtectionLogParame
		parameterFuncs.TALKPAGE           = self._makeTalkPageParameter
		parameterFuncs.TOOLTIPBLURB       = self._makeTooltipBlurbParamet
		parameterFuncs.TOOLTIPFRAGMENT    = self._makeTooltipFragmentPara
		parameterFuncs.VANDAL             = self._makeVandalTemplateParam

		self._params = setmetatable({}, {
			__index = function (t, k)
				local param
				if parameterFuncs[k] then
					param = parameterFuncs[k](self)
				end
				param = param or ''
				t[k] = param
				return param
			end
		})
	end

	msg = msg:gsub('${(%u+)}', self._params)
	return msg
end

function Blurb:_makeCurrentVersionParameter()
	-- A link to the page history or the move log, depending on the kind of
	-- protection.
	local pagename = self._protectionObj.title.prefixedText
	if self._protectionObj.action == 'move' then
		-- We need the move log link.
		return makeFullUrl(
			'Special:Log',
			{type = 'move', page = pagename},
			self:_getExpandedMessage('current-version-move-display')
		)
```

```
        else
                -- We need the history link.
                return makeFullUrl(
                        pagename,
                        {action = 'history'},
                        self:_getExpandedMessage('current-version-edit-display')
                )
        end
end

function Blurb:_makeEditRequestParameter()
        local mEditRequest = require('Module:Submit an edit request')
        local action = self._protectionObj.action
        local level = self._protectionObj.level

        -- Get the edit request type.
        local requestType
        if action == 'edit' then
                if level == 'autoconfirmed' then
                        requestType = 'semi'
                elseif level == 'extendedconfirmed' then
                        requestType = 'extended'
                elseif level == 'templateeditor' then
                        requestType = 'template'
                end
        end
        requestType = requestType or 'full'

        -- Get the display value.
        local display = self:_getExpandedMessage('edit-request-display')

        return mEditRequest._link{type = requestType, display = display}
end

function Blurb:_makeExpiryParameter()
        local expiry = self._protectionObj.expiry
        if type(expiry) == 'number' then
                return self:_formatDate(expiry)
        else
                return expiry
        end
end

function Blurb:_makeExplanationBlurbParameter()
        -- Cover special cases first.
        if self._protectionObj.title.namespace == 8 then
                -- MediaWiki namespace
                return self:_getExpandedMessage('explanation-blurb-nounprotect')
        end

        -- Get explanation blurb table keys
        local action = self._protectionObj.action
        local level = self._protectionObj.level
        local talkKey = self._protectionObj.title.isTalkPage and 'talk' or 'subje

        -- Find the message in the explanation blurb table and substitute any
        -- parameters.
        local explanations = self._cfg.explanationBlurbs
        local msg
        if explanations[action][level] and explanations[action][level][talkKey] t
                msg = explanations[action][level][talkKey]
        elseif explanations[action][level] and explanations[action][level].defaul
                msg = explanations[action][level].default
        elseif explanations[action].default and explanations[action].default[talk
```

```
                msg = explanations[action].default[talkKey]
        elseif explanations[action].default and explanations[action].default.defa
                msg = explanations[action].default.default
        else
                error(string.format(
                        'could not find explanation blurb for action "%s", level
                        action,
                        level,
                        talkKey
                ), 8)
        end
        return self:_substituteParameters(msg)
end

function Blurb:_makeImageLinkParameter()
        local imageLinks = self._cfg.imageLinks
        local action = self._protectionObj.action
        local level = self._protectionObj.level
        local msg
        if imageLinks[action][level] then
                msg = imageLinks[action][level]
        elseif imageLinks[action].default then
                msg = imageLinks[action].default
        else
                msg = imageLinks.edit.default
        end
        return self:_substituteParameters(msg)
end

function Blurb:_makeIntroBlurbParameter()
        if self._protectionObj:isTemporary() then
                return self:_getExpandedMessage('intro-blurb-expiry')
        else
                return self:_getExpandedMessage('intro-blurb-noexpiry')
        end
end

function Blurb:_makeIntroFragmentParameter()
        if self._protectionObj:isTemporary() then
                return self:_getExpandedMessage('intro-fragment-expiry')
        else
                return self:_getExpandedMessage('intro-fragment-noexpiry')
        end
end

function Blurb:_makePagetypeParameter()
        local pagetypes = self._cfg.pagetypes
        return pagetypes[self._protectionObj.title.namespace]
                or pagetypes.default
                or error('no default pagetype defined', 8)
end

function Blurb:_makeProtectionBlurbParameter()
        local protectionBlurbs = self._cfg.protectionBlurbs
        local action = self._protectionObj.action
        local level = self._protectionObj.level
        local msg
        if protectionBlurbs[action][level] then
                msg = protectionBlurbs[action][level]
        elseif protectionBlurbs[action].default then
                msg = protectionBlurbs[action].default
        elseif protectionBlurbs.edit.default then
                msg = protectionBlurbs.edit.default
        else
```

```
                error('no protection blurb defined for protectionBlurbs.edit.defa
        end
        return self:_substituteParameters(msg)
end

function Blurb:_makeProtectionDateParameter()
        local protectionDate = self._protectionObj.protectionDate
        if type(protectionDate) == 'number' then
                return self:_formatDate(protectionDate)
        else
                return protectionDate
        end
end

function Blurb:_makeProtectionLevelParameter()
        local protectionLevels = self._cfg.protectionLevels
        local action = self._protectionObj.action
        local level = self._protectionObj.level
        local msg
        if protectionLevels[action][level] then
                msg = protectionLevels[action][level]
        elseif protectionLevels[action].default then
                msg = protectionLevels[action].default
        elseif protectionLevels.edit.default then
                msg = protectionLevels.edit.default
        else
                error('no protection level defined for protectionLevels.edit.defa
        end
        return self:_substituteParameters(msg)
end

function Blurb:_makeProtectionLogParameter()
        local pagename = self._protectionObj.title.prefixedText
        if self._protectionObj.action == 'autoreview' then
                -- We need the pending changes log.
                return makeFullUrl(
                        'Special:Log',
                        {type = 'stable', page = pagename},
                        self:_getExpandedMessage('pc-log-display')
                )
        else
                -- We need the protection log.
                return makeFullUrl(
                        'Special:Log',
                        {type = 'protect', page = pagename},
                        self:_getExpandedMessage('protection-log-display')
                )
        end
end

function Blurb:_makeTalkPageParameter()
        return string.format(
                '[[%s:%s#%s|%s]]',
                mw.site.namespaces[self._protectionObj.title.namespace].talk.name
                self._protectionObj.title.text,
                self._args.section or 'top',
                self:_getExpandedMessage('talk-page-link-display')
        )
end

function Blurb:_makeTooltipBlurbParameter()
        if self._protectionObj:isTemporary() then
                return self:_getExpandedMessage('tooltip-blurb-expiry')
        else
```

```
                return self:_getExpandedMessage('tooltip-blurb-noexpiry')
        end
end

function Blurb:_makeTooltipFragmentParameter()
        if self._protectionObj:isTemporary() then
                return self:_getExpandedMessage('tooltip-fragment-expiry')
        else
                return self:_getExpandedMessage('tooltip-fragment-noexpiry')
        end
end

function Blurb:_makeVandalTemplateParameter()
        return mw.getCurrentFrame():expandTemplate{
                title="vandal-m",
                args={self._args.user or self._protectionObj.title.baseText}
        }
end

-- Public methods --

function Blurb:makeBannerText(key)
        -- Validate input.
        if not key or not Blurb.bannerTextFields[key] then
                error(string.format(
                        '"%s" is not a valid banner config field',
                        tostring(key)
                ), 2)
        end

        -- Generate the text.
        local msg = self._protectionObj.bannerConfig[key]
        if type(msg) == 'string' then
                return self:_substituteParameters(msg)
        elseif type(msg) == 'function' then
                msg = msg(self._protectionObj, self._args)
                if type(msg) ~= 'string' then
                        error(string.format(
                                'bad output from banner config function with key
                                        .. ' (expected string, got %s)',
                                tostring(key),
                                type(msg)
                        ), 4)
                end
                return self:_substituteParameters(msg)
        end
end

--------------------------------------------------------------------------------
-- BannerTemplate class
--------------------------------------------------------------------------------

local BannerTemplate = {}
BannerTemplate.__index = BannerTemplate

function BannerTemplate.new(protectionObj, cfg)
        local obj = {}
        obj._cfg = cfg

        -- Set the image filename.
        local imageFilename = protectionObj.bannerConfig.image
        if imageFilename then
                obj._imageFilename = imageFilename
        else
```

```
                -- If an image filename isn't specified explicitly in the banner
                -- generate it from the protection status and the namespace.
                local action = protectionObj.action
                local level = protectionObj.level
                local namespace = protectionObj.title.namespace
                local reason = protectionObj.reason

                -- Deal with special cases first.
                if (
                        namespace == 10
                        or namespace == 828
                        or reason and obj._cfg.indefImageReasons[reason]
                        )
                        and action == 'edit'
                        and level == 'sysop'
                        and not protectionObj:isTemporary()
                then
                        -- Fully protected modules and templates get the special
                        -- padlock.
                        obj._imageFilename = obj._cfg.msg['image-filename-indef']
                else
                        -- Deal with regular protection types.
                        local images = obj._cfg.images
                        if images[action] then
                                if images[action][level] then
                                        obj._imageFilename = images[action][level]
                                elseif images[action].default then
                                        obj._imageFilename = images[action].defau
                                end
                        end
                end
        end
        return setmetatable(obj, BannerTemplate)
end

function BannerTemplate:renderImage()
        local filename = self._imageFilename
                or self._cfg.msg['image-filename-default']
                or 'Transparent.gif'
        return makeFileLink{
                file = filename,
                size = (self.imageWidth or 20) .. 'px',
                alt = self._imageAlt,
                link = self._imageLink,
                caption = self.imageCaption
        }
end

--------------------------------------------------------------------------------
-- Banner class
--------------------------------------------------------------------------------

local Banner = setmetatable({}, BannerTemplate)
Banner.__index = Banner

function Banner.new(protectionObj, blurbObj, cfg)
        local obj = BannerTemplate.new(protectionObj, cfg) -- This doesn't need t
        obj.imageWidth = 40
        obj.imageCaption = blurbObj:makeBannerText('alt') -- Large banners use th
        obj._reasonText = blurbObj:makeBannerText('text')
        obj._explanationText = blurbObj:makeBannerText('explanation')
        obj._page = protectionObj.title.prefixedText -- Only makes a difference i
        return setmetatable(obj, Banner)
end
```

```
function Banner:__tostring()
	-- Renders the banner.
	makeMessageBox = makeMessageBox or require('Module:Message box').main
	local reasonText = self._reasonText or error('no reason text set', 2)
	local explanationText = self._explanationText
	local mbargs = {
		page = self._page,
		type = 'protection',
		image = self:renderImage(),
		text = string.format(
			"'''%s'''%s",
			reasonText,
			explanationText and '<br />' .. explanationText or ''
		)
	}
	return makeMessageBox('mbox', mbargs)
end

--------------------------------------------------------------------------------
-- Padlock class
--------------------------------------------------------------------------------

local Padlock = setmetatable({}, BannerTemplate)
Padlock.__index = Padlock

function Padlock.new(protectionObj, blurbObj, cfg)
	local obj = BannerTemplate.new(protectionObj, cfg) -- This doesn't need t
	obj.imageWidth = 20
	obj.imageCaption = blurbObj:makeBannerText('tooltip')
	obj._imageAlt = blurbObj:makeBannerText('alt')
	obj._imageLink = blurbObj:makeBannerText('link')
	obj._indicatorName = cfg.padlockIndicatorNames[protectionObj.action]
		or cfg.padlockIndicatorNames.default
		or 'pp-default'
	return setmetatable(obj, Padlock)
end

function Padlock:__tostring()
	local frame = mw.getCurrentFrame()
	-- The nowiki tag helps prevent whitespace at the top of articles.
	return frame:extensionTag{name = 'nowiki'} .. frame:extensionTag{
		name = 'indicator',
		args = {name = self._indicatorName},
		content = self:renderImage()
	}
end

--------------------------------------------------------------------------------
-- Exports
--------------------------------------------------------------------------------

local p = {}

function p._exportClasses()
	-- This is used for testing purposes.
	return {
		Protection = Protection,
		Blurb = Blurb,
		BannerTemplate = BannerTemplate,
		Banner = Banner,
		Padlock = Padlock,
	}
end
```

```
function p._main(args, cfg, title)
	args = args or {}
	cfg = cfg or require(CONFIG_MODULE)

	local protectionObj = Protection.new(args, cfg, title)

	local ret = {}

	-- If a page's edit protection is equally or more restrictive than its
	-- protection from some other action, then don't bother displaying anythi
	-- for the other action (except categories).
	if not yesno(args.catonly) and (protectionObj.action == 'edit' or
		args.demolevel or
		not getReachableNodes(
			cfg.hierarchy,
			protectionObj.level
		)[effectiveProtectionLevel('edit', protectionObj.title)])
	then
		-- Initialise the blurb object
		local blurbObj = Blurb.new(protectionObj, args, cfg)

		-- Render the banner
		if protectionObj:shouldShowLock() then
			ret[#ret + 1] = tostring(
				(yesno(args.small) and Padlock or Banner)
				.new(protectionObj, blurbObj, cfg)
			)
		end
	end

	-- Render the categories
	if yesno(args.category) ~= false then
		ret[#ret + 1] = protectionObj:makeCategoryLinks()
	end

	return table.concat(ret)
end

function p.main(frame, cfg)
	cfg = cfg or require(CONFIG_MODULE)

	-- Find default args, if any.
	local parent = frame.getParent and frame:getParent()
	local defaultArgs = parent and cfg.wrappers[parent:getTitle():gsub('/sand

	-- Find user args, and use the parent frame if we are being called from a
	-- wrapper template.
	getArgs = getArgs or require('Module:Arguments').getArgs
	local userArgs = getArgs(frame, {
		parentOnly = defaultArgs,
		frameOnly = not defaultArgs
	})

	-- Build the args table. User-specified args overwrite default args.
	local args = {}
	for k, v in pairs(defaultArgs or {}) do
		args[k] = v
	end
	for k, v in pairs(userArgs) do
```

```
                args[k] = v
        end
        return p._main(args, cfg)
end

return p
```