

Module:Template translation

Documentation for this module may be created at Module:Template translation/doc

```
local this = {}

function this.checkLanguage(subpage, default)
--[[Check first if there's an any invalid character that would cause the
mw.language.isKnownLanguageTag function() to throw an exception:
- all ASCII controls in [000-031\127],
- double quote ("), sharp sign (#), ampersand (&), apostrophe ('),
- slash (/), colon (:), semicolon (;), lower than (<), greater than (>),
- brackets and braces ([, ], {, }), pipe (|), backslash (\)
All other characters are accepted, including space and all non-ASCII
characters (including \192, which is invalid in UTF-8).
--]]
if mw.language.isValidCode(subpage) and mw.language.isKnownLanguageTag(subpage)
--[[However "SupportedLanguages" are too restrictive, as they discard many
valid BCP47 script variants (only because MediaWiki still does not
define automatic transliterators for them, e.g. "en-dsrt" or
"fr-brai" for French transliteration in Braille), and country variants,
(useful in localized data, even if they are no longer used for
translations, such as zh-cn, also useful for legacy codes).
We want to avoid matching subpagenames containing any uppercase letter,
(even if they are considered valid in BCP 47, in which they are
case-insensitive; they are not "SupportedLanguages" for MediaWiki, so
they are not "KnownLanguageTags" for MediaWiki).
To be more restrictive, we exclude any character
* that is not ASCII and not a lowercase letter, minus-hyphen, or digit,
  or does not start by a letter or does not finish by a letter or digit;
* or that has more than 8 characters between hyphens;
* or that has two hyphens;
* or with specific uses in template subpages and unusable as languages.
--]]
or string.find(subpage, "^[%l][%-%d%l]*[%d%l]$" ) ~= nil
and string.find(subpage, "[%d%l][%d%l][%d%l][%d%l][%d%l][%d%l][%d%l][%d%l]" ) == nil
and string.find(subpage, "%-%-") == nil
and subpage ~= "doc"
and subpage ~= "layout"
and subpage ~= "sandbox"
and subpage ~= "testcases"
then
    return subpage
end
-- Otherwise there's currently no known language subpage
return default
end

--[[Get the last subpage of the current page if it is a translation.
]]
function this.getLanguageSubpage()
--[[This code does not work in all namespaces where the Translate tool works.
-- It works in the main namespace on Meta because it allows subpages there
-- It would not work in the main namespace of English Wikipedia (but the
-- articles are monolignual on that wiki).
-- On Meta-Wiki the main space uses subpages and its pages are translated.
-- The Translate tool allows translatng pages in all namespaces, even if
-- the namespace officially does not have subpages.
-- On Meta-Wiki the Category namespace still does not have subpages enabled,
-- even if they would be very useful for categorizing templates, that DO have
-- subpages (for documentatio and tstboxes pages). This is a misconfiguration
-- bug of Meta-Wiki. The work-around is to split the full title and then
-- get the last titlepart.
local subpage = mw.title.getCurrentTitle().subpageText
--]]
local titleparts = mw.text.split(mw.title.getCurrentTitle().fullText, '/')
local subpage = titleparts[#titleparts]
return this.checkLanguage(subpage, "")
end
```

```

--[[Get the first part of the language code of the subpage, before the '-'.
]]
function this.getMainLanguageSubpage()
parts = mw.text.split( this.getLanguageSubpage(), '-' )
return parts[1]
end

--[[Get the last subpage of the current frame if it is a translation.
Not used locally.
]]
function this.getFrameLanguageSubpage(frame)
local titleparts = mw.text.split(frame:getParent():getTitle(), '/')
local subpage = titleparts[#titleparts]
return this.checkLanguage(subpage, "")
end

--[[Get the language of the current page.
Not used locally.
]]
function this.getLanguage()
local subpage = mw.title.getCurrentTitle().subpageText
return this.checkLanguage(subpage, mw.language.getContentLanguage():getCode())
end

--[[Get the language of the current frame.
Not used locally.
]]
function this.getFrameLanguage(frame)
local titleparts = mw.text.split(frame:getParent():getTitle(), '/')
local subpage = titleparts[#titleparts]
return this.checkLanguage(subpage, mw.language.getContentLanguage():getCode())
end

function this.title(namespace, basepagename, subpage)
local message, title
local pagename = basepagename
if (subpage or "") ~= ""
then
pagename = pagename .. '/' .. subpage
end
local valid, title = xpcall(function()
return mw.title.new(pagename, namespace) -- costly
end, function(msg) -- catch undocumented exception (!?)
-- thrown when namespace does not exist. The doc still
-- says it should return a title, even in that case...
message = msg
end)
if valid and title ~= nil and (title.id or 0) ~= 0
then
return title
end
return { -- "pseudo" mw.title object with id = nil in case of error
prefixedText = pagename, -- the only property we need below
message = message -- only for debugging
}
end

--[[If on a translation subpage (like Foobar/de), this function returns
a given template in the same language, if the translation is available.
Otherwise, the template is returned in its default language, without
modification.
This is aimed at replacing the current implementation of Template:TNTN.

This version does not expand the returned template name: this solves the
problem of self-recursion in TNT when translatable templates need themselves
to transclude other translatable templates (such as Tnavbar).
]]
function this.getTranslatedTemplate(frame, withStatus)
local args = frame.args
local pagename = args['template']

--[[Check whether the pagename is actually in the Template namespace, or
if we're transcluding a main-namespace page.

```

```

    (added for backward compatibility of Template:TNT)
  ]]
local title
local namespace = args['tntns'] or ''
if (namespace ~= '') -- Checks for tntns parameter for custom ns.
then
    title = this.title(namespace, pagename) -- Costly
else -- Assumes that set page is in ns10.
    namespace = 'Template'
    title = this.title(namespace, pagename) -- Costly
    if title.id == nil
    then -- not found in the Template namespace, assume the main namespace (for backward compatibility)
        namespace = ''
        title = this.title(namespace, pagename) -- Costly
    end
end

-- Get the last subpage and check if it matches a known language code.
local subpage = args['uselang'] or ''
if (subpage == '')
then
    subpage = this.getLanguageSubpage()
end
if (subpage == '')
then
    -- Check if a translation of the pagename exists in English
    local newtitle = this.title(namespace, pagename, 'en') -- Costly
    -- Use the translation when it exists
    if newtitle.id ~= nil
    then
        title = newtitle
    end
else
    -- Check if a translation of the pagename exists in that language
    local newtitle = this.title(namespace, pagename, subpage) -- Costly
    if newtitle.id == nil
    then
        -- Check if a translation of the pagename exists in English
        newtitle = this.title(namespace, pagename, 'en') -- Costly
    end
    -- Use the translation when it exists
    if newtitle.id ~= nil
    then
        title = newtitle
    end
end
-- At this point the title should exist
if withStatus then
    -- status returned to Lua function below
    return title.prefixedText, title.id ~= nil
else
    -- returned directly to MediaWiki
    return title.prefixedText
end
end

--[[If on a translation subpage (like Foobar/de), this function renders
a given template in the same language, if the translation is available.
Otherwise, the template is rendered in its default language, without
modification.
This is aimed at replacing the current implementation of Template:TNT.

Note that translatable templates cannot transclude themselves other
translatable templates, as it will recurse on TNT. Use TNTN instead
to return only the effective template name to expand externally, with
template parameters also provided externally.
]]
function this.renderTranslatedTemplate(frame)
local title, found = this.getTranslatedTemplate(frame, true)
-- At this point the title should exist prior to performing the expansion
-- of the template, otherwise render a red link to the missing page
-- (resolved in its assumed namespace). If we don't test this here, a
-- script error would be thrown. Returning a red link is consistent with
-- MediaWiki behavior when attempting to transclude inexistant templates.

```

```

if not found then
return '[' .. title .. ']'
end

-- Copy args pseudo-table to a proper table so we can feed it to expandTemplate.
-- Then render the pagename.
local args = frame.args
local pargs = (frame:getParent() or {}).args
local arguments = {}
if (args['noshift'] or '') == ''
then
    for k, v in pairs(pargs) do
        -- numbered args >= 1 need to be shifted
        local n = tonumber(k) or 0
        if (n > 0)
        then
            if (n >= 2)
            then
                arguments[n - 1] = v
            end
        else
            arguments[k] = v
        end
    end
else -- special case where TNT is used as autotranslate
-- (don't shift again what is shifted in the invocation)
    for k, v in pairs(pargs) do
        arguments[k] = v
    end
end
arguments['template'] = title -- override the existing parameter of the base template name supplied with the full name of the actual template expanded
arguments['tntns'] = nil -- discard the specified namespace override
arguments['uselang'] = args['uselang'] -- argument forwarded into parent frame
arguments['noshift'] = args['noshift'] -- argument forwarded into parent frame

return frame:expandTemplate{title = '[' .. title, args = arguments}
end

--[[A helper for mocking TNT in Special:TemplateSandbox. TNT breaks
TemplateSandbox; mocking it with this method means templates won't be
localized but at least TemplateSandbox substitutions will work properly.
Won't work with complex uses.
]]
function this.mockTNT(frame)
local pargs = (frame:getParent() or {}).args
local arguments = {}
for k, v in pairs(pargs) do
    -- numbered args >= 1 need to be shifted
    local n = tonumber(k) or 0
    if (n > 0)
    then
        if (n >= 2)
        then
            arguments[n - 1] = v
        end
    else
        arguments[k] = v
    end
end
end
if not pargs[1]
then
    return ''
end
return frame:expandTemplate{title = 'Template:' .. pargs[1], args = arguments}
end

return this

```