



Inhaltsverzeichnis

1. Modul:TNTTools/Doku	2
2. Modul:TNT	3
3. Modul:TNTTools	8



Modul:TNTTools/Doku

Dies ist die Dokumentationsseite für Modul:TNTTools

Contains functions linked to [Module:TNT](#), which at the same time make calls to multilingual tables, located in Commons, for the creation of [modules](#) and [multilingual templates](#).

TNTTools has:

- **Question functions:** with boolean or numerical indexed return. To be called from other modules or from templates. With:
 - Case sensitive option.
 - Possibility of **more than one translated text value** (where each value is separated by "|").
- To put aside write, adding "l18n/" as a prefix and ".tab" extension as a suffix for the table names.

Modul:TNT

This module allows templates and modules to be easily translated as part of the [multilingual templates and modules project](#). Instead of storing English text in a module or a template, TNT module allows modules to be designed language-neutral, and store multilingual text in the [tabular data pages](#) on Commons. This way your module or template will use those translated strings (messages), or if the message has not yet been translated, will fallback to English. When someone updates the translation table, your page will automatically update (might take some time, or you can purge it), but no change in the template or module is needed on any of the wikis. This process is very similar to MediaWiki's [localisation](#), and supports all standard localization conventions such as `{{PLURAL|...}}` and [other parameters](#).

This module can be used from templates using `#invoke`, and from other modules. For a simple example, see [Data:I18n/Template:Graphs.tab](#) - a table with two messages, each message having a single parameter. By convention, all translation tables should have `"Data:I18n/..."` prefix to separate them from other types of data.

Inhaltsverzeichnis	
1 Using from Templates	3
2 Translating Template Parameters	4
3 Using from Modules	4
4 Using TNTTools	4

Using from Templates

Description	Wiki Markup
In a template, this command translates source_table message using Commons' Data:I18n/Template:Graphs.tab translation table.	<pre> {{#invoke:TNT msg I18n/Template:Graphs.tab source_table }}</pre>
If your message contains parameters, you can specify them after the message ID.	<pre> {{#invoke:TNT msg I18n/Template:My_Template.tab message-with-two-params param1 param2 }}</pre>



Translating Template Parameters

Template parameters are usually stored as a **JSON templatedata** block inside the template's /doc subpage. This makes it convenient to translate, but when a new parameter is added to a global template, all /doc pages need to be updated in every language. TNT helps with this by automatically generating the templatedata block from a table stored on Commons. Placing this line into every /doc sub-page will use [Data:Templatedata/Graph:Lines.tab](#) table to generate all the needed templatedata information in every language. Even if the local community has not translated the full template documentation, they will be able to see all template parameters, centrally updated.

```
{{#invoke:TNT | doc | Graph:Lines }}
```

Using from Modules

Just like templates, modules should also use this module for localization:

```
local TNT = require('Module:TNT')

-- format <messageId> string with two parameters using a translation table.
local text = TNT.format('I18n/My_module_messages', 'messageId', 'param1', 'param2')

-- Same, but translate to a specific language.
local text = TNT.formatInLanguage('fr', 'I18n/My_module_messages', 'messageId', 'param1', 'param2')
```

Using TNTTools

[Module:TNTTools](#) has:

- Question functions: with boolean or numerical indexed return. To be called from other modules or from templates. With:
 - Case sensitive option.
 - Possibility of more than one translated text value (where each value is separated by "|").
- To put aside write, adding "I18n/" as a prefix and ".tab" extension as a suffix for the table names.
- Several examples.

```
--
-- INTRO:  (!!! DO NOT RENAME THIS PAGE !!!)
--   This module allows any template or module to be copy/pasted between
--   wikis without any translation changes. All translation text is stored
--   in the global Data:*.tab pages on Commons, and used everywhere.
--
-- SEE:    https://www.mediawiki.org/wiki/Multilingual_Templates_and_Modules
--
-- ATTENTION:
--   Please do NOT rename this module - it has to be identical on all wikis.
--   This code is maintained at https://www.mediawiki.org/wiki/Module:TNT
```

```
-- Please do not modify it anywhere else, as it may get copied and override yo
-- Suggestions can be made at https://www.mediawiki.org/wiki/Module_talk:TNT
--
-- DESCRIPTION:
-- The "msg" function uses a Commons dataset to translate a message
-- with a given key (e.g. source-table), plus optional arguments
-- to the wiki markup in the current content language.
-- Use lang=xx to set language. Example:
--
-- {{#invoke:TNT | msg
-- | I18n/Template:Graphs.tab <!-- https://commons.wikimedia.org/wiki/Data:
-- | source-table <!-- uses a translation message with id = "so
-- | param1 }} <!-- optional parameter -->
--
--
-- The "doc" function will generate the <templatedata> parameter documentation
-- This way all template parameters can be stored and localized in a single C
-- NOTE: "doc" assumes that all documentation is located in Data:Templatedata/
--
-- {{#invoke:TNT | doc | Graph:Lines }}
-- uses https://commons.wikimedia.org/wiki/Data:Templatedata/Graph:Lines.t
-- if the current page is Template:Graph:Lines/doc
--
--
local p = {}
local i18nDataset = 'I18n/Module:TNT.tab'

-- Forward declaration of the local functions
local sanitizeDataset, loadData, link, formatMessage

function p.msg(frame)
    local dataset, id
    local params = {}
    local lang = nil
    for k, v in pairs(frame.args) do
        if k == 1 then
            dataset = mw.text.trim(v)
        elseif k == 2 then
            id = mw.text.trim(v)
        elseif type(k) == 'number' then
            table.insert(params, mw.text.trim(v))
        elseif k == 'lang' and v ~= '' then
            lang = mw.text.trim(v)
        end
    end
    return formatMessage(dataset, id, params, lang)
end

-- Identical to p.msg() above, but used from other lua modules
-- Parameters: name of dataset, message key, optional arguments
-- Example with 2 params: format('I18n/Module:TNT', 'error_bad_msgkey', 'my-key
function p.format(dataset, key, ...)
    local checkType = require('libraryUtil').checkType
    checkType('format', 1, dataset, 'string')
    checkType('format', 2, key, 'string')
    return formatMessage(dataset, key, {...})
end

-- Identical to p.msg() above, but used from other lua modules with the language
-- Parameters: language code, name of dataset, message key, optional arguments
-- Example with 2 params: formatInLanguage('es', I18n/Module:TNT', 'error_bad_ms
function p.formatInLanguage(lang, dataset, key, ...)
    local checkType = require('libraryUtil').checkType
```



```
        checkType('formatInLanguage', 1, lang, 'string')
        checkType('formatInLanguage', 2, dataset, 'string')
        checkType('formatInLanguage', 3, key, 'string')
        return formatMessage(dataset, key, {...}, lang)
end

-- Obsolete function that adds a 'c:' prefix to the first param.
-- "Sandbox/Sample.tab" -> 'c:Data:Sandbox/Sample.tab'
function p.link(frame)
    return link(frame.args[1])
end

function p.doc(frame)
    local dataset = 'Templatedata/' .. sanitizeDataset(frame.args[1])
    return frame:extensionTag('templatedata', p.getTemplateData(dataset)) ..
        formatMessage(i18nDataset, 'edit_doc', {link(dataset)})
end

function p.getTemplateData(dataset)
    -- TODO: add '_' parameter once lua starts reindexing properly for "all"
    local data = loadData(dataset)
    local names = {}
    for _, field in pairs(data.schema.fields) do
        table.insert(names, field.name)
    end

    local params = {}
    local paramOrder = {}
    for _, row in pairs(data.data) do
        local newVal = {}
        local name = nil
        for pos, val in pairs(row) do
            local columnName = names[pos]
            if columnName == 'name' then
                name = val
            else
                newVal[columnName] = val
            end
        end
        if name then
            params[name] = newVal
            table.insert(paramOrder, name)
        end
    end

    -- Work around json encoding treating {"1":{...}} as an [{...}]
    params['zzz123']=''

    local json = mw.text.jsonEncode({
        params=params,
        paramOrder=paramOrder,
        description=data.description
    })

    json = string.gsub(json, '"zzz123":",?', "")

    return json
end

-- Local functions

sanitizeDataset = function(dataset)
    if not dataset then
        return nil
    end
end
```



```
        end
        dataset = mw.text.trim(dataset)
        if dataset == '' then
            return nil
        elseif string.sub(dataset,-4) ~= '.tab' then
            return dataset .. '.tab'
        else
            return dataset
        end
    end
end

loadData = function(dataset, lang)
    dataset = sanitizeDataset(dataset)
    if not dataset then
        error(formatMessage(i18nDataset, 'error_no_dataset', {}))
    end

    -- Give helpful error to thirdparties who try and copy this module.
    if not mw.ext or not mw.ext.data or not mw.ext.data.get then
        error('Missing JsonConfig extension; Cannot load https://commons
    end

    local data = mw.ext.data.get(dataset, lang)

    if data == false then
        if dataset == i18nDataset then
            -- Prevent cyclical calls
            error('Missing Commons dataset ' .. i18nDataset)
        else
            error(formatMessage(i18nDataset, 'error_bad_dataset', {l
        end
    end
    return data
end

-- Given a dataset name, convert it to a title with the 'commons:data:' prefix
link = function(dataset)
    return 'c:Data:' .. mw.text.trim(dataset or '')
end

formatMessage = function(dataset, key, params, lang)
    for _, row in pairs(loadData(dataset, lang).data) do
        local id, msg = unpack(row)
        if id == key then
            local result = mw.message.newRawMessage(msg, unpack(para
            return result:plain()
        end
    end
    if dataset == i18nDataset then
        -- Prevent cyclical calls
        error('Invalid message key "' .. key .. "'')
    else
        error(formatMessage(i18nDataset, 'error_bad_msgkey', {key, link(c
    end
end

return p
```

Modul:TNTTools

Contains functions linked to [Module:TNT](#), which at the same time make calls to multilingual tables, located in Commons, for the creation of [modules and multilingual templates](#).

TNTTools has:

- **Question functions:** with boolean or numerical indexed return. To be called from other modules or from templates. With:
 - Case sensitive option.
 - Possibility of **more than one translated text value** (where each value is separated by "|").
- To put aside write, adding "I18n/" as a prefix and ".tab" extension as a suffix for the table names.

```
local p = {}

local TNT = require('Module:TNT')
--local SD = require('Module:SimpleDebug')

function p.TNTTabFull (TNTTab)
    if (string.sub(TNTTab, 1, 5)) ~= 'I18n/' then
        TNTTab = 'I18n/'..TNTTab
    end
    if (string.sub(TNTTab, string.len(TNTTab)-3)) ~= '.tab' then
        TNTTab = TNTTab..'tab'
    end
    return TNTTab
end --TNTTabFull

function p.TNTTabCommons (TNTTab)
    return 'Commons:Data: '..p.TNTTabFull(TNTTab)
end

function p.LnkTNTTab (TNTTab)
    return '['..p.TNTTabCommons(TNTTab)..']'
end

function I18nStr (TNTTab, S, IsMsg, params)
    TNTTab = p.TNTTabFull (TNTTab)
    local SEnd = TNT.format(TNTTab, S, unpack(params)) or ''
    if SEnd == '' then
        SEnd = TNT.formatInLanguage('en', TNTTab, S, unpack(params))
        if IsMsg then
            local icon = '[[File:Arbcom ru editing.svg|12px|Not found]]'
            SEnd = SEnd..icon
        end
    end
    return SEnd
end --I18nStr

function p.GetMsgP (TNTTab, S, ...)
    return I18nStr (TNTTab, S, true, {...})
end
```



```
function p.GetStrP (TNNTab, S, ...)
    return I18nStr (TNNTab, S, false, {...})
end

function p.TabTransCS (TNNTab, S, CaseSensitive)
    CaseSensitive = ((CaseSensitive ~= nil) and (CaseSensitive == true)) or false
    local Wds = TNT.format (p.TNNTabFull(TNNTab), S)
    if not CaseSensitive then
        Wds = string.lower (Wds)
    end
    return mw.text.split (Wds, '|')
end --TabTransCS

function p.TabTransMT (TNNTab, S, MaxTrans)
    local FN = p.TNNTabFull(TNNTab)
    local tab = mw.text.split (TNT.format (FN, S), '|')
    if #tab > MaxTrans then
        error (string.format('Found %s translations for "%s". Search in |
                                -- Translation not required
        end
    end
    return tab
end --TabTransMT

function p.SFoundInTNTArr (TNNTab, val, CaseSensitive, S)
    if (S == nil) or (S == '') then
        error('Not arguments trying to find "'.val..'"') --It doesn't re
    end
    local Arr = p.TabTransCS (TNNTab, S, CaseSensitive)
    if not CaseSensitive then
        val = string.lower (val)
    end
    for I, W in ipairs(Arr) do
        if W == val then
            return true
        end
    end
    return false
end --SFoundInTNTArr

function p.IdxFromTabTrans (TNNTab, val, CaseSensitive, ...)
    local Arr = unpack(arg)
    if Arr == nil then
        error('Not arguments trying to find "'.val..'"') --It doesn't re
    end
    local Idx = 0
    for I, W in ipairs(Arr) do
        if p.SFoundInTNTArr (TNNTab, val, CaseSensitive, W) then
            Idx = I
            break
        end
    end
    return Idx
end --IdxFromTabTrans

return p
```