

Modul:Format TemplateData

Ausgabe: 17.04.2026

Letzte Änderung: 24.08.2022

Modul:Format TemplateData

[Vorlage:Nutshell](#) [Vorlage:Lua](#)

TemplateData – Module with auxiliary functions for template documentation, especially by [TemplateData](#).

Core functionality is improved presentation on documentation pages.

Inhaltsverzeichnis	
1 Vorlage:Anchor Improve template documentation page – MediaWiki disappointing	2
1.1 Vorlage:Anchor Improved presentation	2
1.2 Eliminate disadvantages	4
2 General workflow	4
3 Vorlage:Anchor Functions for templates	4
3.1 Details	4
3.2 Examples (test page)	5
4 Vorlage:Anchor Functions for Lua modules (API)	5
5 Usage	6
6 Dependencies	6

[Vorlage:Anchor Improve template documentation page – MediaWiki disappointing](#)

For presentation of template depiction in VisualEditor agreement was made to abandon all markup and clickable links, permitting all tooltips in all environments. Basically this is reasonable, albeit tooltips with markup and clickable links are supported as HTML application for decades now and JavaScript is present anyway when VisualEditor is used.

- In consequence it was decided, that also presentation on template documentation views never ever is permitted to contain effective links or markup.
- That involved, that on many template documentation pages two separated parameter documentation tables are needed and required to be maintained simultaneously: One plain text version for VisualEditor, and a useful one for complex circumstances, with links and markup and lists and tables. – BTW, VisualEditor has not only tooltips, but also a static GUI view, where the lack of deepening links in parameter description is painful.

This state is indefensible.

[Vorlage:Anchor Improved presentation](#)

In addition to the simple syntax supported by MediaWiki and presented in the VisualEditor, the following features can be added to the JSON code for the template documentation page. They affect the elements classified as *InterfaceText*, but are only useful for description fields.

Wikilinks (internal format)

- Using double square brackets pages can be linked as common.
- In VisualEditor, only link title is visible, as it is displayed otherwise.

External links (URL format)

- Open URL are linked as usual by themselves. In VisualEditor they appear as normal text.
- External links enclosed in simple square brackets are displayed normally on the template documentation page. In VisualEditor the title is omitted and the URL is displayed so that the users can c&p it and transfer it to the address field of the browser. There is no other way.

Apostrophs ' for italic and bold

- They can be used to emphasize on the documentation page and are missing in VisualEditor (regular script).

HTML entities

- The following entities can be used: < > & " and all numeric formats.

HTML tags

- HTML tags (and the MediaWiki elements that are not replaced in advance) are removed for the VisualEditor. Otherwise, they remain effective.
- Attributes are often included in ", which conflicts with the JSON syntax. It is important to make sure that ' is used, which can be a problem with template transclusions.

<noexport> ... </noexport>

- The enclosed areas are not exported to the VisualEditor.
- More complex wiki syntax and extensive explanations can be restricted to the documentation page.
- Within a *noexport* area, the line structure of the source text is considered. Otherwise everything is running in a single line, as it would also be represented in the VisualEditor.

Templates

- In particular when the template parameter `JSON=` is used, templates can be distributed anywhere in the JSON code. However, the expanded syntax might collide with the JSON syntax.

More effects

- According to the state (required, suggested, optional, deprecated) the table rows are highlighted in light blue, white, gray and pale red.
- When sorting by state, this importance is taken into account and not the alphabetical sequence of the keywords.
- Each parameter can be addressed as a jump destination. The fragment is `#templatedata:parameter-name`.
- Missing labels are highlighted as errors.
- A maintenance category is triggered if errors occur.
- If there are no parameters, the element `params: { }` is not required.

Eliminate disadvantages

Two aspects were found to be particularly disturbing in 2013–2017:

1. Even if no parameters at all were defined, a table head is always displayed for a table without content. Even more, this is sortable.
 - A reduction was rejected with [Vorlage:Phab](#). A sortable table of the parameters would be always necessary, even if the table has no rows at all and consists only of the header row.
 - This ridiculous statement led to the development of this module in 2016.
2. Even if the context does not permit that default values or even AutoValue specifications will be defined ever, a content-free six-line definition list is output for each individual parameter value.
 - [Vorlage:Phab](#) / [Vorlage:Phab](#) / [Vorlage:Phab](#) / [Vorlage:Phab](#)
 - MediaWiki did not even deign to answer the disastrous documentation page situation.

The general comments show that MediaWiki only regards the presentation of TemplateData specifications in the VisualEditor as important. However, someone has to program and maintain the templates and someone needs to generate the template description and make it manageable beside the functionality in the VisualEditor form, but that is beyond ken.

General workflow

- An attempt is made to read the JSON object (string) from passed template parameters.
- If this failed, the source code of the current and the documentation page is searched for `<templatedata>` elements.
- Two representations are obtained from the JSON object input:
 1. A localized version, markup etc. stripped off, in JSON format.
 2. An HTML structure, basically similar to the MediaWiki representation, possibly with table of the parameters, with enhanced features.
- The result of the template is a visible documentation with markup, followed by a hidden `<templatedata>` element. This is done for the export and corresponds to the MediaWiki guidelines.
 - If current page has been identified as documentation page the hidden `<templatedata>` is suppressed, and those pages do not appear separately in [Special:PagesWithProp/templatedata](#).

[Vorlage:Anchor](#) Functions for templates

Details

f [Vorlage:Anchor](#)

Improve TemplateData-presentation; used in [Template:Format TemplateData](#)

Parameters of template transclusion environment (all optional):

1

JSON string or `<templatedata>` object

JSON

JSON string

(precedes **1**)

Transition from `<templatedata>` objects with pipe symbols needs special attention: Pipes are to be represented as `{{!}}`, on double curly brackets one should be encoded by HTML entity.

TOC

1 – Insert table of contents after general purpose descriptions; but before parameter list, if present

Example

lazy

1 – Presentation only, do not generate an effective data block

For general method descriptions.

debug

1 – developer mode

Parameters of #invoke for particular project adaption (all optional):

cat

Title of a maintenance category on invalid parameter value etc.

debug

Development mode, if provided and not equal 0

docpageCreate

Pattern for creation of subpage names; %s/Doku

docpageDetect

Pattern for recognition of subpage names; /Doku\$

msgDescMiss

Localisation: complaint text on missing description

Returns: HTML code; and/or error message, probably with class="error"

failsafe **Vorlage:Anchor**

Version identification: 2017-11-06

Optional additional parameter 1 – requested minimal version identification

Returns: (empty), if minimal version condition not matched

Examples (test page)

A [test page](#) illustrates practical use.

Vorlage:Anchor Functions for Lua modules (API)

Some functions described above can be used by other modules:

```
local lucky, TemplateData = pcall( require, "Module:Format TemplateData" )
if type( TemplateData ) == "table" then
    TemplateData = TemplateData.TemplateData()
else
    -- failure; TemplateData is the error message
    return "<span class='error'>" .. TemplateData .. "</span>"
end
```

TemplateData.[failsafe](#)(atleast)

1. atleast
optional
nil or minimal version request

Returns: *string* or *false*

TemplateData.getPlainJSON(adapt)

Reduce enhanced JSON information to MediaWiki JSON

1. adapt
string, with JSON (enhanced)

Returns: *string*, with JSON (MediaWiki)

TemplateData.test(adapt, arglist)

Simulation of template functionality

1. adapt
table, #invoke parameters
2. arglist
table, template parameters

Returns: *string*

Usage

Currently focusing on one template only:

- [Template:Format TemplateData](#)

Dependencies

- [Module:Plain text](#)
- [Module:TNT](#) (for the [Vorlage:Para](#) parameter)

```
local TemplateData = { serial = "2017-11-06",
                       suite = "TemplateData" }
local plaintext = require("Module:Plain text")
--[=[
improve template:TemplateData
]=]
local Config = {
  -- multiple #invoke option names mapped into unique internal fields
  cat          = "strange",
  classNoNumTOC = "suppressTOCnum",
  -- classParams = "classTable",
  cssParams    = "stylesTable",
  cssParWrap   = "stylesTabWrap",
  debug        = false,
  docpageCreate = "suffix",
  docpageDetect = "subpage",
  msgDescMiss  = "solo",
  -- classTable = false, -- class for params table
  loudly       = false, -- show exported element, etc.
  solo         = false, -- complaint on missing description
  strange      = false, -- title of maintenance category
```

```

stylesTable = false, -- styles for params table
stylesTabWrap = false, -- styles for params table wrapper
subpage = false, -- pattern to identify subpage
suffix = false, -- subpage creation scheme
suppressTOCnum = false -- class for TOC number suppression
}
local Data = {
  div = false, -- <div class="mw-templatedata-doc-wrap">
  got = false, -- table, initial templatedata object
  heirs = false, -- table, params that are inherited
  less = false, -- main description missing
  lasting = false, -- old syntax encountered
  lazy = false, -- doc mode; do not generate effective <templatedata>
  leading = false, -- show TOC
-- low = false, -- 1= mode
  order = false, -- parameter sequence
  params = false, -- table, exported parameters
  scream = false, -- error messages
  slang = false, -- project language code
  slim = false, -- JSON reduced to plain
  source = false, -- JSON input
  strip = false, -- <templatedata> evaluation
  tag = false, -- table, exported root element
  title = false, -- page
  tree = false -- table, rewritten templatedata object
}
local Permit = {
  styles = { required = "border-left: 3px solid black;",
    suggested = "border-left: 3px solid #888;",
    optional = "border-left: 3px solid #ccc",
    deprecated = "border-left: 3px dotted red; background-color: #FDD;",
    tableheadbg = "background-color: #B3B7FF;" },
  params = { aliases = "table",
    autovalue = "string",
    default = "string table I18N nowiki",
    deprecated = "boolean string",
    description = "string table I18N",
    example = "string table I18N nowiki",
    label = "string table I18N",
    inherits = "string",
    required = "boolean",
    suggested = "boolean",
    suggestedvalues = "table",
    type = "string" },
  root = { description = "string table I18N",
    format = "string",
    maps = "table",
    params = "table",
    paramOrder = "table",
    sets = "table" },
  search = "[{,}%s*(["\"])%s%%1%%s*:%%s*%{",
  types = { boolean = true,
    content = true,
    date = true,
    line = true,
    number = true,
    string = true,
    unknown = true,
    url = true,
    ["wiki-file-name"] = true,

```

```

    ["wiki-page-name"]    = true,
    ["wiki-template-name"] = true,
    ["wiki-user-name"]    = true,
    ["unbalanced-wikitext"] = true,
    ["string/line"]       = "line",
    ["string/wiki-page-name"] = "wiki-page-name",
    ["string/wiki-user-name"] = "wiki-user-name" }
}

--
-- Generic utility functions
--

local function _ne( value )
    -- Is string not empty?
    -- Parameter:
    --   value -- the value to test
    -- Return:
    --   boolean -- whether `value` is truthy and not the empty string
    return value and value ~= ""
end -- _ne

local function Fault( alert )
    -- Memorize error message
    -- Parameter:
    --   alert -- string, error message
    if Data.scream then
        Data.scream = string.format( "%s *** %s", Data.scream, alert )
    else
        Data.scream = alert
    end
end -- Fault()

local function collapseWhitespace ( a )
    -- Collapses whitespace, HTML style.
    return a:gsub( "%s*\n%s*", " " )
        :gsub( "%s%s+", " " )
end -- collapseWhitespace

-----
--

local function facet( ask, at )
    -- Find physical position of parameter definition in JSON
    -- Parameter:
    --   ask -- string, parameter name
    --   at  -- number, physical position within definition
    -- Returns number, or nil
    local seek = string.format( Permit.search,
        ask:gsub( "%%", "%%%" )
        :gsub( "([%-.()]+*?^$%[%]])",
            "%% 1" ) )
    local i, k = Data.source:find( seek, at )
    local r, slice, source
    while i and not r do
        source = Data.source:sub( k + 1 )
        slice = source:match( "^%s*\"([^\"]+)\"%s*:" )
        if not slice then
            slice = source:match( "^%s*\"([^\"]+)\"%s*:" )
        end
    end

```

```

    if ( slice and Permit.params[ slice ] ) or
        source:match( "^%s*%" ) then
        r = k
    else
        i, k = Data.source:find( seek, k )
    end
end -- while i
return r
end -- facet()

local function getLocalizedText( adapt )
    -- Retrieve localized text from system message
    -- Parameter:
    --   adapt -- string, message ID after "templatedata-"
    -- Returns string, with localized text
    return mw.message.new( "templatedata-" .. adapt ):plain()
end -- getLocalizedText()

local function faculty( adjust )
    -- Test template arg for boolean
    --   adjust -- string or nil
    -- Returns boolean
    local s = type( adjust )
    local r
    if s == "string" then
        r = mw.text.trim( adjust )
        r = ( r ~= "" and r ~= "0" )
    elseif s == "boolean" then
        r = adjust
    else
        r = false
    end
    return r
end -- faculty()

local function failures()
    -- Retrieve error collection and category
    -- Returns string
    local r
    if Data.scream then
        local e = mw.html.create( "span" )
            :addClass( "error" )
            :wikitext( Data.scream )
        r = tostring( e )
        mw.addWarning( ""TemplateData"<br />" .. Data.scream )
        if Config.strange then
            r = string.format( "%s[[category:%s]]",
                r,
                Config.strange )
        end
    else
        r = ""
    end
    return r
end

```

```

end -- failures()

local function handleNoexportWhitespace( adjust )
  -- Reduces runs of spaces, including newlines, to a single space, so the
  -- whole string is on one line. <noexport> blocks are left alone, but the
  -- <noexport> tags themselves are removed.
  -- adjust -- string
  -- Returns string, with adjusted text
  local r
  if adjust:find( "<noexport>", 1, true ) then
    local i = 1
    local j, k = adjust:find( "<noexport>", i, true )
    r = ""
    while j do
      if j > 1 then
        r = r .. collapseWhitespace( adjust:sub( i, j - 1 ) )
        end
        i = k + 1
        j, k = adjust:find( "</noexport>", i, true )
        if j then
          r = r .. adjust:sub( i, j - 1 )
          i = k + 1
          j, k = adjust:find( "<noexport>", i, true )
        else
          Fault( "missing </noexport>" )
        end
      end -- while j
      r = r .. adjust:sub( i )
    else
      r = collapseWhitespace( adjust )
    end
    return r
  end -- handleNoexportWhitespace()

local function faraway( alternatives )
  -- Retrieve project language version from multilingual text
  -- Parameter:
  -- alternatives -- table, to be evaluated
  -- Returns
  -- 1 -- string, with best match
  -- 2 -- table of other versions, if any
  local n = 0
  local variants = { }
  local r1, r2
  if not Data.slang then
    Data.slang = mw.language.getContentLanguage():getCode()
  end
  for k, v in pairs( alternatives ) do
    if type( v ) == "string" then
      v = mw.text.trim( v )
      if v ~= "" then
        variants[ k ] = v
        n = n + 1
      end
    end
  end
  end -- for k, v

```

```

if n > 0 then
  for k, v in pairs( variants ) do
    if v then
      if n == 1 then
        r1 = v
      elseif k:lower() == Data.slang then
        variants[ k ] = nil
        r1 = v
        r2 = variants
        break -- for k, v
      end
    end
  end -- for k, v
  if not r1 then
    local seek = string.format( "^%s-", Data.slang )
    for k, v in pairs( variants ) do
      if v and k:lower():match( seek ) then
        variants[ k ] = nil
        r1 = v
        r2 = variants
        break -- for k, v
      end
    end -- for k, v
    if not r1 then
      local others = mw.language.getFallbacksFor( slang )
      table.insert( others, "en" )
      for i = 1, #others do
        seek = others[ i ]
        if variants[ seek ] then
          r1 = variants[ seek ]
          variants[ seek ] = nil
          r2 = variants
          break -- for i
        end
      end -- i = 1, #others
    end
    if not r1 then
      for k, v in pairs( variants ) do
        if v then
          variants[ k ] = nil
          r1 = v
          r2 = variants
          break -- for k, v
        end
      end -- for k, v
    end
    if r2 then
      for k, v in pairs( r2 ) do
        if v then
          local baseCode = k:match( "^%s*(%a%a%a%a?)-?%a*%s*$" )
          if not baseCode or not mw.language.isKnownLanguageTag( baseCode ) then
            Fault( string.format( "Invalid <code>lang=%s</code>", k ) )
          end
        end
      end -- for k, v
    end
  end
  return r1, r2
end -- faraway()

```

```

local function fathers()
  -- Merge params with inherited values
  local n = 0
  local p = Data.params
  local t = Data.tree.params
  local p2, t2
  for k, v in pairs( Data.heirs ) do
    n = n + 1
  end -- for k, v
  for i = 1, n do
    for k, v in pairs( Data.heirs ) do
      if v and not Data.heirs[ v ] then
        n = n - 1
        t[ k ].inherits = nil
        Data.heirs[ k ] = nil
        p2 = { }
        t2 = { }
        for k2, v2 in pairs( p[ v ] ) do
          p2[ k2 ] = v2
        end -- for k2, v2
        if p[ k ] then
          for k2, v2 in pairs( p[ k ] ) do
            if type( v2 ) ~= "nil" then
              p2[ k2 ] = v2
            end
          end -- for k2, v2
        end
        p[ k ] = p2
        for k2, v2 in pairs( t[ v ] ) do
          t2[ k2 ] = v2
        end -- for k2, v2
        for k2, v2 in pairs( t[ k ] ) do
          if type( v2 ) ~= "nil" then
            t2[ k2 ] = v2
          end
        end -- for k2, v2
        t[ k ] = t2
      end
    end -- for k, v
  end -- i = 1, n
  if n > 0 then
    local s
    for k, v in pairs( Data.heirs ) do
      if v then
        if s then
          s = string.format( "%s &#124; %s", s, k )
        else
          s = "Circular inherits: " .. k
        end
      end
    end
    Fault( s )
  end
end -- fathers()

```

```

local function feasible( about, asked )
  -- Create description head
  -- Parameter:
  -- about -- table, supposed to contain description
  -- asked -- true, if mandatory description
  -- Returns <block>, with head, or nil
  local para = mw.html.create( "div" )
  local plus, r
  if about and about.description then
    if type( about.description ) == "string" then
      para:wikitext( about.description )
    else
      para:wikitext( about.description[ 1 ] )
      plus = mw.html.create( "ul" )
      if not Config.loudly then
        plus:addClass( "templatedata-maintain" )
        :css( "display", "none" )
      end
      for k, v in pairs( about.description[ 2 ] ) do
        plus:node( mw.html.create( "li" )
          :node( mw.html.create( "code" )
            :wikitext( k ) )
          :node( mw.html.create( "br" ) )
          :wikitext( handleNoexportWhitespace( v ) ) )
        end -- for k, v
      end
    elseif Config.solo and asked then
      para:addClass( "error" )
      :wikitext( Config.solo )
      Data.less = true
    else
      para = false
    end
    if para then
      if plus then
        r = mw.html.create( "div" )
          :node( para )
          :node( plus )
      else
        r = para
      end
    end
  end
  return r
end -- feasible()

```

```

local function feat()
  -- Check and store parameter sequence
  if Data.source then
    local i = 0
    local s
    for k, v in pairs( Data.tree.params ) do
      if i == 0 then
        Data.order = { }
        i = 1
        s = k
      else
        i = 2
        break -- for k, v
      end
    end
  end

```

```

end
end -- for k, v
if i > 1 then
  local pointers = { }
  local points = { }
  for k, v in pairs( Data.tree.params ) do
    i = facet( k, 1 )
    if i then
      table.insert( points, i )
      pointers[ i ] = k
      i = facet( k, i )
      if i then
        s = "Parameter '%s' detected twice"
        Fault( string.format( s, k ) )
      end
    else
      s = "Parameter '%s' not detected"
      Fault( string.format( s, k ) )
    end
  end -- for k, v
  table.sort( points )
  for i = 1, #points do
    table.insert( Data.order, pointers[ points[ i ] ] )
  end -- i = 1, #points
elseif s then
  table.insert( Data.order, s )
end
end
end -- feat()

```

```

local function feature( access )
  -- Create table row for parameter, check and display violations
  -- Parameter:
  -- access -- string, with name
  -- Returns <tr>
  local mode, s, status
  local fine = function ( a )
    s = mw.text.trim( a )
    return a == s and
      a ~= "" and
      not a:find( "%|=\n" ) and
      not a:find( "%s%s" )
  end
  local begin = mw.html.create( "td" )
  local code = mw.html.create( "code" )
  local desc = mw.html.create( "td" )
  local legal = true
  local param = Data.tree.params[ access ]
  local ranking = { "required", "suggested", "optional", "deprecated" }
  local r = mw.html.create( "tr" )
  local sort, typed

  for k, v in pairs( param ) do
    if v == "" then
      param[ k ] = false
    end
  end
end -- for k, v

```

```

-- label
sort = param.label or access
if sort:match( "^%d+$" ) then
    begin:attr( "data-sort-value",
        string.format( "%05d", tonumber( sort ) ) )
end
begin:css( "font-weight", "bold" )
    :wikitext( sort )

-- name and aliases
code:css( "font-size", "92%" )
    :css( "white-space", "nowrap" )
    :wikitext( access )
if not fine( access ) then
    code:addClass( "error" )
    Fault( string.format( "Bad ID params.<code>%s</code>", access ) )
    legal = false
    begin:attr( "data-sort-value", " " .. sort )
end
code = mw.html.create( "td" )
    :node( code )
if access:match( "^%d+$" ) then
    code:attr( "data-sort-value",
        string.format( "%05d", tonumber( access ) ) )
end
if type( param.aliases ) == "table" then
    local lapsus
    for k, v in pairs( param.aliases ) do
        code:tag( "br" )
        if type( v ) == "string" then
            if not fine( v ) then
                lapsus = true
                code:node( mw.html.create( "span" )
                    :addClass( "error" )
                    :css( "font-style", "italic" )
                    :wikitext( "string" ) )
            end
            code:wikitext( s )
        else
            lapsus = true
            code:node( mw.html.create( "code" )
                :addClass( "error" )
                :wikitext( type( v ) ) )
        end
    end -- for k, v
    if lapsus then
        s = string.format( "params.<code>%s</code>.aliases", access )
        Fault( getLocalizedText( "invalid-value" ):gsub( "$1", s ) )
        legal = false
    end
end

-- description etc.
s = feasible( param )
if s then
    desc:node( s )
end
if param.suggestedvalues or param.default or param.example or param.autovalue then
    local details = { "suggestedvalues", "default", "example", "autovalue" }
    local dl = mw.html.create( "dl" )

```

```

local dd, section, show, sv
for i = 1, #details do
  s = details[ i ]
  show = param[ s ]
  if show then
    section = getLocalizedText( "doc-param-" .. s )
    dt = mw.html.create( "dt" ):wikitext( section )
    dd = mw.html.create( "dd" )
    if type( show ) == "string" and (string.len(show) < 80) then
      dt.cssText("float: left;margin-right: 1.6em;")
    end
    if param.type == "boolean" then
      if (type( show ) == "table") then
        -- "suggestedvalues"
        for i = 1, #show do
          sv = show[ i ]
          if i > 1 then
            dd:wikitext("&#10;")
          end
          if sv == "0" then
            dd:wikitext("<span style=\"color: #610; font-weight: bold;\">False</span> (<code>0</code>)")
          elseif sv == "1" then
            dd:wikitext("<span style=\"color: #050; font-weight: bold;\">True</span> (<code>1</code>)")
          else
            dd:tag("code"):wikitext( sv )
          end
        end
      elseif show == "0" then
        dd:wikitext("<span style=\"color: #610; font-weight: bold;\">False</span> (<code>0</code>)")
      elseif show == "1" then
        dd:wikitext("<span style=\"color: #050; font-weight: bold;\">True</span> (<code>1</code>)")
      else
        dd:wikitext( show )
      end
    elseif type( show ) == "table" then
      -- "suggestedvalues"
      for i = 1, #show do
        sv = show[ i ]
        if i > 1 then
          dd:wikitext("&#10;")
        end
        dd:tag("code"):wikitext( sv )
      end
    else
      dd:wikitext( show )
    end
    dl:node( dt )
    :node( dd )
  end
end -- i = 1, #details
desc:node( dl )
end

-- type
if param.type then
  s = Permit.types[ param.type ]
  typed = mw.html.create( "td" )
  if s then
    if type( s ) == "string" then
      Data.params[ access ].type = s
    end
  end
end

```

```

typed:wikitext( getLocalizedText( "doc-param-type-" .. s ) )
  :tag( "br" )
typed:node( mw.html.create( "span" )
  :addClass( "error" )
  :wikitext( param.type ) )
Data.lasting = true
else
s = getLocalizedText( "doc-param-type-" .. param.type )
typed:wikitext( s )
end
else
Data.params[ access ].type = "unknown"
typed:addClass( "error" )
  :wikitext( "INVALID" )
s = string.format( "params.<code>%s</code>.type", access )
Fault( getLocalizedText( "invalid-value" ):gsub( "$1", s ) )
legal = false
end
else
typed = mw.html.create( "td" )
  :wikitext( getLocalizedText( "doc-param-type-unknown" ) )
end

-- status
if param.required then
mode = 1
if param.deprecated then
Fault( string.format( "Required deprecated <code>%s</code>",
access ) )
legal = false
end
elseif param.deprecated then
mode = 4
elseif param.suggested then
mode = 2
else
mode = 3
end
status = ranking[ mode ]
ranking = getLocalizedText( "doc-param-status-" .. status )
if mode == 1 or mode == 4 then
ranking = mw.html.create( "span" )
  :css( "font-weight", "bold" )
  :wikitext( ranking )
if type( param.deprecated ) == "string" then
ranking:tag( "br" )
ranking:wikitext( param.deprecated )
end
end

-- <tr>
r:attr( "id", "templatedata:" .. mw.uri.anchorEncode( access ) )
:cssText( Permit.styles[ status ] )
:node( begin )
:node( code )
:node( desc )
:node( typed )
:node( mw.html.create( "td" )
  :attr( "data-sort-value", tostring( mode ) )
  :node( ranking ) )

```

```

:newline()
if not legal then
  r:css( "border", "#FF0000 3px solid" )
end
return r
end -- feature()

local function features()
  -- Create <table> for parameters
  -- Returns <table>, or nil
  local r
  if Data.tree and Data.tree.params then
    local style = Permit.styles.tableheadbg
    local tbl = mw.html.create( "table" )
      :addClass( "wikitable" )
    local tr = mw.html.create( "tr" )
    feat()
    if Data.order and #Data.order > 1 then
      tbl:addClass( "sortable" )
    end
    -- if Config.classTable then
    --   tbl:addClass( Config.classTable )
    -- end
    if Config.stylesTable then
      tbl:cssText( Config.stylesTable )
    end
    tr:node( mw.html.create( "th" )
      :attr( "colspan", "2" )
      :cssText( style )
      :wikitext( getLocalizedText( "doc-param-name" ) ) )
    :node( mw.html.create( "th" )
      :cssText( style )
      :wikitext( getLocalizedText( "doc-param-desc" ) ) )
    :node( mw.html.create( "th" )
      :cssText( style )
      :wikitext( getLocalizedText( "doc-param-type" ) ) )
    :node( mw.html.create( "th" )
      :cssText( style )
      :wikitext( getLocalizedText( "doc-param-status" ) ) )
    tbl:newline()
    --   :node( mw.html.create( "thead" )
    --     :node( tr )
    --   )
    :newline()
    if Data.order then
      for i = 1, #Data.order do
        tbl:node( feature( Data.order[ i ] ) )
      end -- for i = 1, #Data.order
    end
    if Config.stylesTabWrap then
      r = mw.html.create( "div" )
        :cssText( Config.stylesTabWrap )
        :node( tbl )
    else
      r = tbl
    end
  end
  return r
end

```

```
end -- features()
```

```
local function finalize()  
  -- Wrap presentation into frame  
  -- Returns string  
  local r  
  if Data.div then  
    r = tostring( Data.div )  
  elseif Data.strip then  
    r = Data.strip  
  else  
    r = ""  
  end  
  return r .. failures()  
end -- finalize()
```

```
local function find()  
  -- Find JSON data within page source (title)  
  -- Returns string, or nil  
  local s = Data.title:getContent()  
  local i, j = s:find( "<templatedata>", 1, true )  
  local r  
  if i then  
    local k = s:find( "</templatedata>", j, true )  
    if k then  
      r = mw.text.trim( s:sub( j + 1, k - 1 ) )  
    end  
  end  
  return r  
end -- find()
```

```
local function flat( adjust )  
  -- Remove formatting from text string  
  -- Parameter:  
  -- arglist -- string, to be stripped, or nil  
  -- Returns string, or nil  
  local r  
  if adjust then  
    r = adjust:gsub( "\\n", " " )  
    if r:find( "<noexport>", 1, true ) then  
      r = r:gsub( "<noexport>(.*?)</noexport>", "" )  
    end  
    r = plaintext._main(r)  
    if r:find( "&", 1, true ) then  
      r = mw.text.decode( r )  
    end  
  end  
  return r  
end -- flat()
```

```
local function flush()  
  -- JSON encode narrowed input; obey unnamed (numerical) parameters
```

```

-- Returns <templatedata> JSON string
local r
if Data.tag then
  r = mw.text.jsonEncode( Data.tag ):gsub( "% }$", "," )
else
  r = "{"
end
r = r .. "\n\"params\":{"
if Data.order then
  local sep = ""
  local s
  for i = 1, #Data.order do
    s = Data.order[ i ]
    r = string.format( "%s%s\n%s:%s",
                      r,
                      sep,
                      mw.text.jsonEncode( s ),
                      mw.text.jsonEncode( Data.params[ s ] ) )
    sep = ",\n"
  end -- for i = 1, #Data.order
end
r = r .. "\n}\n}"
return r
end -- flush()

local function focus( access )
-- Check components; focus multilingual description, build trees
-- Parameter:
-- access -- string, name of parameter, nil for root
local f = function ( a, at )
  local r
  if at then
    r = string.format( "<code>params.%s</code>", at )
  else
    r = ""root""
  end
  if a then
    r = string.format( "%s<code>.%s</code>", r, a )
  end
  return r
end
local parent
if access then
  parent = Data.got.params[ access ]
else
  parent = Data.got
end
if type( parent ) == "table" then
  local elem, got, permit, s, scope, slot, tag, target
  if access then
    permit = Permit.params
    if type( access ) == "number" then
      slot = tostring( access )
    else
      slot = access
    end
  else
    permit = Permit.root
  end
end

```

```

end
for k, v in pairs( parent ) do
  scope = permit[ k ]
  if scope then
    s = type( v )
    if s == "string" then
      v = mw.text.trim( v )
    end
    if scope:find( s, 1, true ) then
      if scope:find( "I18N", 1, true ) then
        if s == "string" then
          elem = handleNoexportWhitespace( v )
        else
          local translated
          v, translated = faraway( v )
          if v then
            if translated and
              k == "description" then
              elem = { [ 1 ] = handleNoexportWhitespace( v ),
                [ 2 ] = translated }
            else
              elem = handleNoexportWhitespace( v )
            end
          end
          else
            elem = false
          end
        end
      end
      if v then
        if scope:find( "nowiki", 1, true ) then
          elem = mw.text.nowiki( v )
        else
          v = flat( v )
        end
      end
    end
  else
    if k == "params" and not access then
      v = nil
      elem = nil
    elseif k == "format" and not access then
      v = mw.text.decode( v )
      elem = v
    elseif k == "inherits" then
      elem = v
      if not Data.heirs then
        Data.heirs = { }
      end
      Data.heirs[ slot ] = v
      v = nil
    elseif s == "string" then
      v = mw.text.nowiki( v )
      elem = v
    else
      elem = v
    end
  end
end
if type( elem ) ~= "nil" then
  if not target then
    if access then
      if not Data.tree.params then
        Data.tree.params = { }
      end
    end
  end
end

```

```

        end
        Data.tree.params[ slot ] = { }
        target = Data.tree.params[ slot ]
    else
        Data.tree = { }
        target = Data.tree
    end
end
end
target[ k ] = elem
elem = false
end
if type( v ) ~= "nil" then
    if not tag then
        if access then
            if not Data.params then
                Data.params = { }
            end
            Data.params[ slot ] = { }
            tag = Data.params[ slot ]
        else
            Data.tag = { }
            tag = Data.tag
        end
    end
    tag[ k ] = v
end
else
    s = string.format( "Type <code>%s</code> bad for %s",
        scope, f( k, slot ) )
    Fault( s )
end
else
    Fault( "Unknown component " .. f( k, slot ) )
end
end -- for k, v
else
    Fault( f() .. " needs to be of <code>object</code> type" )
end
end -- focus()

```

```

local function format()
    -- Build presented documentation
    -- Returns <div>
    local r = mw.html.create( "div" )
    local s = feasible( Data.tree, true )
    if s then
        r:node( s )
    end
    if Data.leading then
        local toc = mw.html.create( "div" )
        if Config.suppressTOCnum then
            toc.addClass( Config.suppressTOCnum )
        end
        toc:css( "margin-top", "0.5em" )
        :wikitext( "__TOC__" )
        r:newline()
        :node( toc )
        :newline()
    end
end

```

```

end
s = features()
if s then
  if Data.leading then
    r:node( mw.html.create( "h2" )
            :wikitext( getLocalizedText( "doc-params" ) ) )
    :newline()
  end
  r:node( s )
end
if Data.tree and Data.tree.format then
  local e, style
  s = Data.tree.format:lower( Data.tree.format )
  if s == "inline" or s == "block" then
    style = "i"
  else
    style = "code"
  end
  r:node( mw.html.create( "p" )
          :wikitext( "Format: " )
          :node( mw.html.create( style )
                :wikitext( s ) ) )
end
return r
end -- format()

```

```

local function free()
  -- Remove JSON comment lines
  Data.source:gsub( "( [{,}] (%s*\n%s*//.*\n%s*)( [},\"])",
                  "%1%3" )
end -- free()

```

```

local function full()
  -- Build HTML table for display from JSON data, and append an invisible
  -- <templatedata> block.
  Data.div = mw.html.create( "div" )
    :addClass( "mw-templatedata-doc-wrap" )
  focus()
  if Data.tag then
    if type( Data.got.params ) == "table" then
      for k, v in pairs( Data.got.params ) do
        focus( k )
      end -- for k, v
      if Data.heirs then
        fathers()
      end
    end
  end
  Data.div:node( format() )
  if not Data.lazy then
    Data.slim = flush()
    if TemplateData.frame then
      local div = mw.html.create( "div" )
      local tdata = { [ 1 ] = "templatedata",
                    [ 2 ] = Data.slim }
      Data.strip = TemplateData.frame:callParserFunction( "#tag",

```

```

                                tdata )
div:wikitext( Data.strip )
if Config.loudly then
    -- Display raw templatedata table all the time.
    Data.div:node( mw.html.create( "hr" ) )
    Data.div:node( div )
else
    -- Creates an expand link to check raw templatedata table.
    local wrapper = mw.html.create( "div" )
    wrapper:addClass( "mw-collapsible" )
    wrapper:addClass( "mw-collapsed" )
    wrapper:css( "font-size", "85%" )
    div:addClass( "mw-collapsible-content" )
    wrapper:wikitext( ""Test of raw TemplateData output": " )
    wrapper:node( div )
    Data.div:node( wrapper )
end
end
end
end -- full()

local function furnish( adapt, arglist )
    -- Called by f, this function is the first to do any real work when the
    -- module is invoked.
    -- Parameter:
    --   adapt -- table, #invoke parameters
    --   arglist -- table, template parameters
    -- Returns string
--local spy=""
    local source
    for k, v in pairs( Config ) do
        if adapt[ k ] and adapt[ k ] ~= "" then
            Config[ v ] = adapt[ k ]
        end
    end -- for k, v
    Config.loudly = faculty( arglist.debug or adapt.debug )
--if mw.site.server:find( "//de.wikipedia.beta.wmflabs.org", 1, true ) then
--   Config.loudly = true
--end
    Data.lazy = faculty( arglist.lazy ) and not Config.loudly
    Data.leading = faculty( arglist.TOC )
    if arglist.JSON then
        source = arglist.JSON
    elseif _ne( arglist.TNT ) then
        local tnt = require( "Module:TNT" )
        source = tnt.getTemplateData( "Templatedata/"
            .. mw.text.trim( arglist.TNT ) )
    elseif arglist[ 1 ] then
        local s = mw.text.trim( arglist[ 1 ] )
        local start = s:sub( 1, 1 )
        if start == "<" then
            Data.strip = s
        elseif start == "{" then
            source = s
        elseif mw.ustring.sub( s, 1, 8 ) ==
            mw.ustring.char( 127, 39, 34, 96, 85, 78, 73, 81 ) then -- <DEL> ' ` U N I Q
            Data.strip = s
        end
    end

```

```

end
if not source then
  Data.title = mw.title.getCurrentTitle()
  source = find()
  if not source and
    Config.subpage and Config.suffix and
    not Data.title.text:match( Config.subpage ) then
    local s = string.format( Config.suffix,
      Data.title.prefixedText )
    Data.title = mw.title.new( s )
    if Data.title.exists then
      source = find()
    end
  end
end
--if source and
--  ( source:find( "|", 1, true ) or
--  source:find( "}" , 1, true ) ) then
--  -- <ref
--spy=string.format( "[[category:%s]]", Config.strange )
--end
end
if not Data.lazy and Config.subpage then
  if not Data.title then
    Data.title = mw.title.getCurrentTitle()
  end
  Data.lazy = Data.title.text:match( Config.subpage )
end
TemplateData.getPlainJSON( source )
return finalize()
--return spy .. finalize()
end -- furnish()

```

```

TemplateData.failsafe = function ( assert )
  -- Checks the age of this implementation against some minimum ("assert").
  local r
  if not assert or assert <= TemplateData.serial then
    r = TemplateData.serial
  else
    r = false
  end
  return r
end -- TemplateData.failsafe()

```

```

TemplateData.getPlainJSON = function ( adapt )
  -- Reduce enhanced JSON data to plain text localized JSON
  -- Parameter:
  --  adapt -- string, with enhanced JSON
  -- Returns string, or not
  if type( adapt ) == "string" then
    Data.source = adapt
    free()
    Data.got = mw.text.jsonDecode( Data.source )
    if Data.got then
      full()
      if Data.lasting then
        Fault( "deprecated type syntax" )
      end
    end
  end
end

```

```

    end
    if Data.less then
      Fault( Config.solo )
    end
    elseif not Data.strip then
      Fault( "fatal JSON error" )
    end
  end
  end
  return Data.slim
end -- TemplateData.getPlainJSON()

```

```

TemplateData.test = function ( adapt, arglist )
  TemplateData.frame = mw.getCurrentFrame()
  return furnish( adapt, arglist )
end -- TemplateData.test()

```

```

-- Export
local p = { }

```

```

p.f = function ( frame )
  -- The entry point for templates invoking the module.
  -- Just wraps furnish in an exception handler.
  local lucky, result
  TemplateData.frame = frame
  lucky, result = pcall( furnish, frame.args, frame:getParent().args )
  if not lucky then
    Fault( "INTERNAL: " .. result )
    result = failures()
  end
  return result
end -- p.f()

```

```

p.failsafe = function ( frame )
  -- Versioning interface
  local s = type( frame )
  local since
  if s == "table" then
    since = frame.args[ 1 ]
  elseif s == "string" then
    since = frame
  end
  if since then
    since = mw.text.trim( since )
    if since == "" then
      since = false
    end
  end
  return TemplateData.failsafe( since ) or ""
end -- p.failsafe()

```

```

p.TemplateData = function ()
  -- Module interface
  return TemplateData
end

```

```

return p

```
