Inhaltsverzeichnis

Modul:Template wrapper

This module is to be used in wrapper templates to allow those templates to provide default parameter values and allow editors to pass additional parameters to the underlying working template.

When writing a wrapper template, give this module all of the normally required default parameters necessary to use the wrapper template in its base form. Editors then use the wrapper template as-is or may supply additional wrapper and canonical parameters. Any of the canonical parameters supported by the working template may be added to the wrapper template or supplied by editors in article space. When an editor supplies a parameter that has a default value in the wrapper template, the editor-supplied value overrides the default. When it is necessary to remove a default parameter, editors may set the parameter value to the special keyword `unset` which will cause this wrapper module to erase the wrapper template's default value for that parameter. This module discards empty named parameters.

Positional parameters are not normally passed on to the working template. Setting Vorlage:Para will pass all positional parameters to the working template. Positional parameters cannot be excluded; positional parameters may be `unset`.

Parameters that are used only by the wrapper should be either positional (Vorlage:Param) or listed in Vorlage:Para (a comma-separated list of named parameters). This module will not pass `_excluded` parameters to the working template.

Inhaltsverzeichnis

1 Usage 2
2 Parameter details 4
2.1 _template 4
2.2 _alias-map 4
2.3 _reuse 5
2.4 _exclude 5
2.5 _include-positional 6
2.6 Overriding default parameters 6
3 Debugging/documentation mode 6

Usage

```
{{#invoke:Template wrapper|wrap|_template=Vorlage:Var|_exclude=Vorlage:Var, Vorlage:Var, ...|_reuse=Vorlage:Var, Vorlage:Var, ...|_alias-map=Vorlage:Var:Vorlage:Var|_include-positional=yes|<Vorlage:Var>|<Vorlage:Var>|...}}
```

Control parameters

Vorlage:Para – (required) the name, without namespace, of the working template (the template that is wrapped); see §_template below

Vorlage:Para – comma-separated list of parameter names used by the wrapper template that are not to be passed to the working template; see §_exclude below

Vorlage:Para – comma-separated list of canonical names that have meaning to both the wrapper template and to the working template; see §_reuse below

Vorlage:Para – comma-separated list of wrapper-template parameter names that are to be treated as aliases of specified working template canonical parameters; see §_alias-map below

Vorlage:Para – pass all positional parameters to the working template; see §_include-positional below

Definitions

canonical parameter – a parameter supported and used by the working template

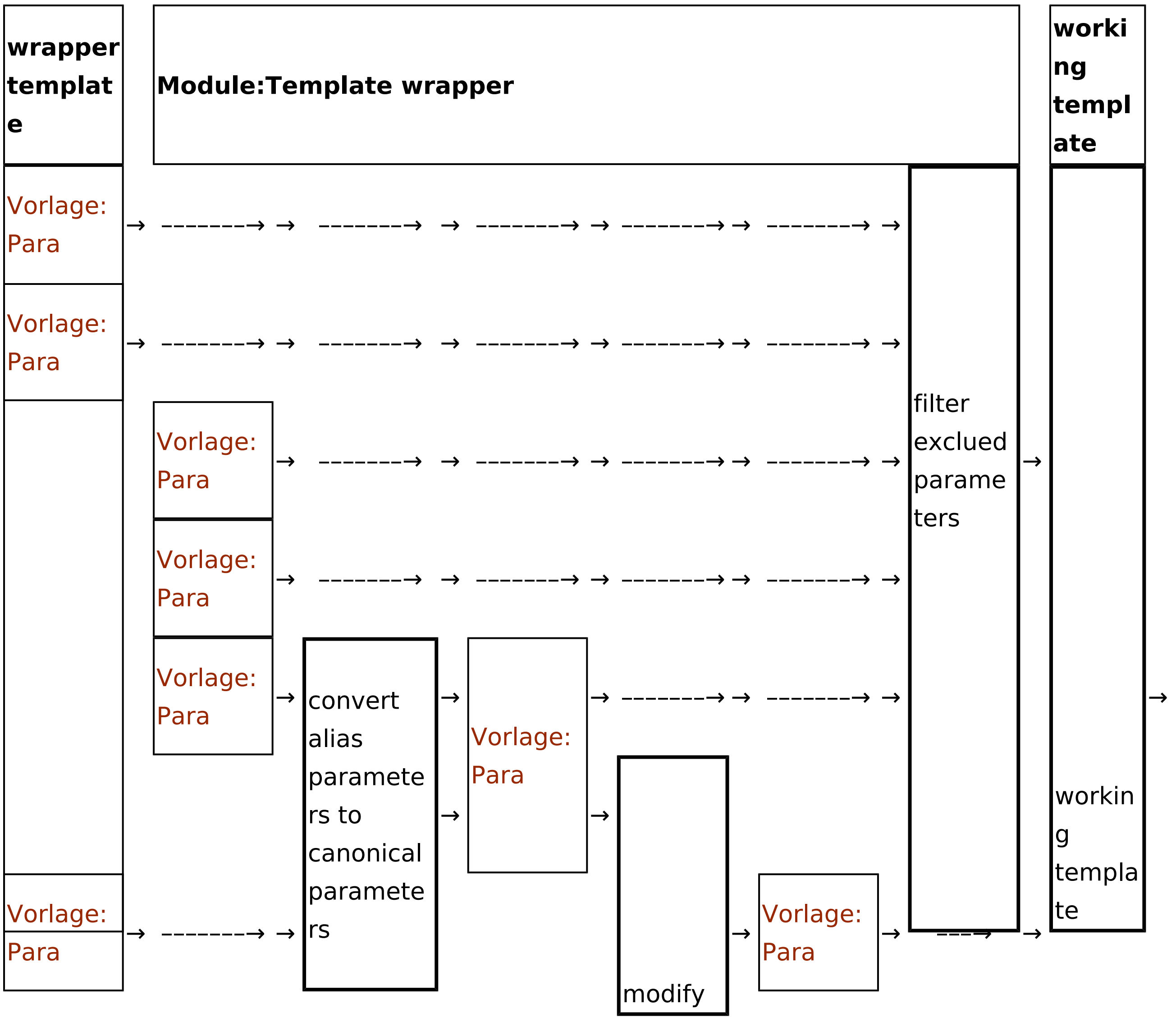
wrapper parameter – a parameter used by the wrapper template; may provide data for canonical parameters or control other aspects of the wrapper template

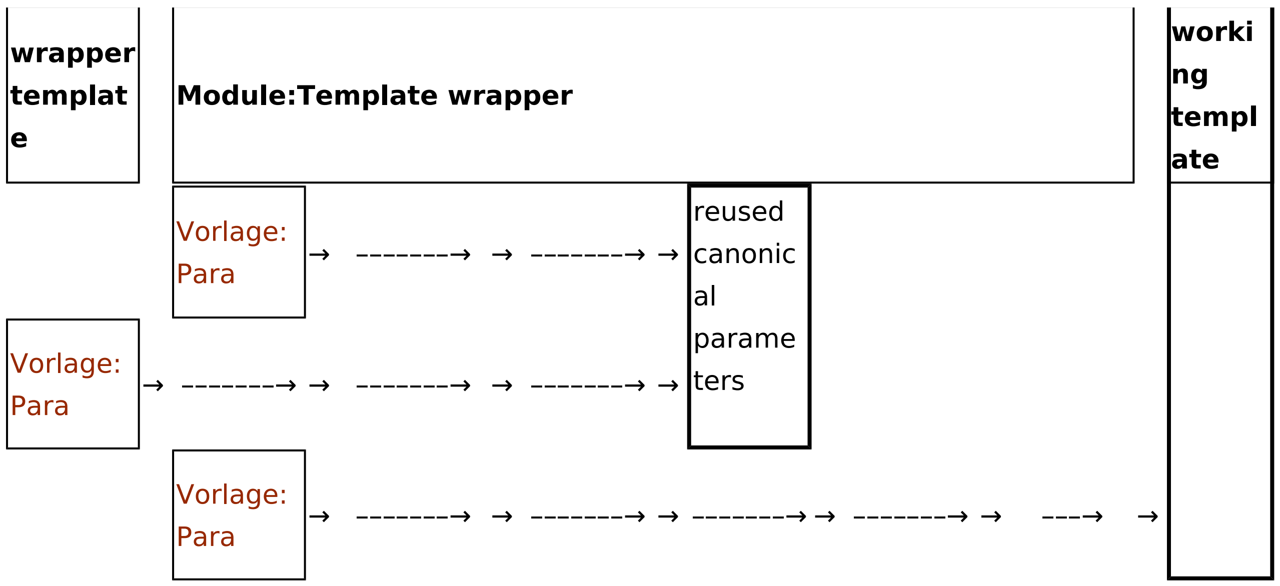
alias parameter – a wrapper parameter that is contextually meaningful to the wrapper template but must be renamed to a canonical parameter for use by the working template

reused parameter – a parameter that is shared by both wrapper and working templates and has been modified by wrapper template

default parameter – a canonical parameter given a default value in the wrapper template

parameter processing





Parameter details

_template

The only required parameter, Vorlage:Para supplies the name, without namespace, of the working template (the template that is wrapped). If this parameter is omitted, Module:Template wrapper will emit the error message:

> `|_template=` missing or empty

_alias-map

Vorlage:Para takes a comma-separated list of wrapper-template parameters that are to be treated as aliases of specified working template canonical parameters. Each mapping element of the list has the form:

> `<Vorlage:Var>:<Vorlage:Var>` – where: `<Vorlage:Var>` is a wrapper parameter name and `<Vorlage:Var>` is a canonical parameter name

In this example, it may be preferable for a wrapper template to use Vorlage:Para which may be unknown to the working template but the working template may have an equivalent Vorlage:Para so in the `{{#invoke:}}` we would write:

> Vorlage:Para

Positional parameters may also be mapped to canonical parameters:

> Vorlage:Para

Enumerated wrapper parameters may be mapped to enumerated canonical parameters using the `#` enumerator specifier:

> Vorlage:Para

Given the above example, Vorlage:Para will map to Vorlage:Para; also, Vorlage:Para and Vorlage:Para will map to Vorlage:Para

Multiple wrapper parameters can map to a single canonical parameter:

Vorlage:Para

Wrapper parameters listed in Vorlage:Para are not passed to the working template. Mapping positional parameters when Vorlage:Para may give undesirable results. Vorlage:Para and Vorlage:Para will cause all other positional parameters to be passed to the working template as is: wrapper template `{{{2}}}` becomes working template `{{{2}}}`, etc; working template will not get `{{{1}}}` though it will get Vorlage:Para.

_reuse

Vorlage:Para takes a comma-separated list of canonical parameters that have meaning to both the wrapper template and to the working template

In the simplest cases, a canonical parameter passed into the wrapper template overrides a default parameter provided in the wrapper template. Sometimes a wrapper parameter is the same as a canonical parameter and the wrapper template needs to modify the parameter value before it is passed to the working template. In this example, Vorlage:Para is both a wrapper parameter and a canonical parameter that the wrapper template needs to modify before passing to the working template. To do this we first write:

Vorlage:Para

then, in the wrapper template's `{{#invoke:Template wrapper|wrap|_template=...|...}}` we write:

Vorlage:Para

_reused parameters cannot be overridden.

_exclude

Vorlage:Para takes a comma-separated list of parameters used by the wrapper template that are not to be passed to the working template. This list applies to all wrapper and canonical parameters (including those canonical parameters that are renamed alias parameters) received from the wrapper template.

As an example, a wrapper template might use Vorlage:Para to supply a portion of the value assigned to default parameter Vorlage:Para so we would write:

Vorlage:Para

then, in the wrapper template's `{{#invoke:Template wrapper|wrap|_template=...|...}}` we write:

Vorlage:Para

The modified Vorlage:Para value is passed on to working template but Vorlage:Para and its value is not.

_reused and default parameters cannot be excluded.

_include-positional

Vorlage:Para is a boolean parameter that takes only one value: yes; the default (empty, missing) is no (positional parameters normally excluded). When set to yes, Module:Template wrapper will pass all positional parameters to the working template.

See also §_alias-map.

Overriding default parameters

Editors may override default parameters by simply setting the default parameter to the desired value in the wrapper template. This module ignores empty parameters (those parameters that are named but which do not have an assigned value). When it is desirable to override a default parameter to no value, use the special keyword unset. Default parameters with this value are passed to the working template as empty (no assigned value) parameters.

_reused parameters cannot be unset or overridden.

Debugging/documentation mode

This module has two entry points. A wrapper template might use a module {{#invoke:}} written like this:

`{{#invoke:Template wrapper|{{#if:{{{_debug|}}}|list|wrap}}|_template=<Vorlage:Var>|_exclude=_debug, ...|...}}`

where the Vorlage:Para wrapper parameter, set to any value, will cause the module to render the call to the working template without actually calling the working template.

As an example, Vorlage:Tlx is a wrapper template that uses Vorlage:Tlx as its working template. Vorlage:Tld accepts positional parameters but Vorlage:Tld does not so the wrapper template must convert the positional parameters to named parameters which it does using the Vorlage:Para parameter:

```
{{#invoke:template wrapper|{{#if:{{{_debug|}}}|list|wrap}}|_template=citation
  |_exclude=..., _debug <!-- unnecessary detail omitted -->
  |_alias-map=1:title, 2:author, 3:language
```

This example uses positional parameters and sets Vorlage:Para to show that the Vorlage:Tld template is correctly formed:

`{{cite wikisource|Sentido y sensibilidad|Jane Austen|es|_debug=yes}}`

Vorlage:Cite wikisource

and, with Vorlage:Para unset:

`{{cite wikisource|Sentido y sensibilidad|Jane Austen|es|_debug=}}`

Vorlage:Cite wikisource

The Vorlage:Para name is chosen here for convenience but may be anything so long as it matches the {{#if:}} in the {{#invoke:}}.

You may also call the `link` function to get something like the left-hand side of Template:yy. This is essentially the `list` function with the template name turned into a link. Vorlage:Yytop Vorlage:Yy Vorlage:Yybottom

---

```
require('Module:No globals');

local error_msg = '<span style=\"font-size:100%\" class=\"error\"><code style=\"c


--[[--------------------------< I S _ I N _ T A B L E >-------------------------

scan through tbl looking for value; return true if found, false else

]]

local function is_in_table (tbl, value)
    for k, v in pairs (tbl) do
        if v == value then return true end
    end
    return false;
end


--[[--------------------------< A D D _ P A R A M E T E R >---------------------

adds parameter name and its value to args table according to the state of boolear
template execution; k=v string for template listing.

]]

local function add_parameter (k, v, args, list)
        if list then
                table.insert( args, table.concat ({k, '=', v}));
        else
                args[k] = v;
        end
end


--[[--------------------------< A L I A S _ M A P _ G E T >---------------------

returns a table of local template (parent frame) parameter names and the target t
in [key]=<value> pairs where:
        [key] is local template parameter name (an alias)
        <value> is target template parameter name (the canonical parameter name u

The parameter |_alias-map= has the form:
        |_alias-map=<list>
where <list> is a comma-separated list of alias / canonical parameter name pairs
        <from> : <to>
where:
        <from> is the local template's parameter name (alias)
        <to> is the target template's parameter name (canonical)
        for enumerated parameters place an octothorp (#) where the enumerator dig
                <from#> : <to#>

]]

local function alias_map_get (_alias_map)
```

```
	local T = mw.text.split (_alias_map, '%s*,%s*');
	local mapped_aliases = {};
	local l_name, t_name;

	for _, alias_pair in ipairs (T) do
		l_name, t_name = alias_pair:match ('(.-)%s*:%s*(.+)');
		if l_name and t_name then
			if tonumber (l_name) then
				l_name = tonumber (l_name);
			end
			mapped_aliases[l_name] = t_name;
		end
	end

	return mapped_aliases;
end


--[[--------------------------< F R A M E _ A R G S _ G E T >--------------------

Fetch the wrapper template's 'default' and control parameters; adds default param

returns content of |_template= parameter (name of the working template); nil else

]]

local function frame_args_get (frame_args, args, list)
	local template;

	for k, v in pairs (frame_args) do
		if 'string' == type (k) and (v and ('' ~= v)) then
			if '_template' == k then
				template = v;
			elseif '_exclude' ~= k and '_reuse' ~= k and '_include-po
				add_parameter (k, v, args, list);
			end
		end
	end

	return template;
end


--[=[--------------------------< P F R A M E _ A R G S _ G E T >----------------

Fetches the wrapper template's 'live' parameters; adds live parameters that aren'
args table; positional parameters may not be excluded

no return value

]=]

local function pframe_args_get (pframe_args, args, exclude, _include_positional,
	for k, v in pairs (pframe_args) do
		if 'string' == type (k) and not is_in_table (exclude, k) then
			if v and ('' ~= v) then
				if 'unset' == v:lower() then
					v = '';
				end
				add_parameter (k, v, args, list)
			end
		end
	end
```

```
        if _include_positional then
                for i, v in ipairs (pframe_args) do
                        if 'unset' == v:lower() then
                                v = '';
                        end
                        add_parameter (i, v, args, list);
                end
        end
end


--[[--------------------------< _ M A I N >--------------------------------------

Collect the various default and live parameters into args styled according to boo

returns name of the working or listed template or nil for an error message

]]

local function _main (frame, args, list)
        local template;
        local exclude = {};
        local reuse_list = {};
        local alias_map = {};
        local _include_positional;

        if frame.args._exclude and ('' ~= frame.args._exclude) then
                exclude = mw.text.split (frame.args._exclude, "%s*,%s*");
        end

        if frame.args._reuse and ('' ~= frame.args._reuse) then
                reuse_list = mw.text.split (frame.args._reuse, "%s*,%s*");
        end

        if frame.args['_alias-map'] and ('' ~= frame.args['_alias-map']) then
                alias_map = alias_map_get (frame.args['_alias-map']);
        end

        template = frame_args_get (frame.args, args, list);
        if nil == template or '' == template then
                return nil;
        end

        _include_positional = 'yes' == frame.args['_include-positional'];


        local _pframe_args = frame:getParent().args;
        local pframe_args = {};

        for k, v in pairs (_pframe_args) do
                pframe_args[k] = v;
        end

-- here we look for pframe parameters that are aliases of canonical parameter nam
-- we replace the alias with the canonical.  We do this here because the reuse_li
-- canonical parameter names so first we convert alias parameter names to canonic
-- we remove those canonical names from the pframe table that are reused (provide
-- template through the frame args table)

        for k, v in pairs (alias_map) do
                if pframe_args[k] then
                        pframe_args[v] = _pframe_args[k];
                        pframe_args[k] = nil;
                end
```

```
	end

	for k, v in pairs (pframe_args) do
		if 'string' == type (k) then
			if alias_map[k..'#'] then
				pframe_args[alias_map[k..'#']:gsub('#', '')] = v;
				pframe_args[k] = nil;
			elseif k:match ('%d+') then
				local temp = k:gsub ('%d+', '#');
				local enum = k:match ('%d+');

				if alias_map[temp] then
					pframe_args[alias_map[temp]:gsub('#', enu
					pframe_args[k] = nil;
				end
			end
		end
	end

-- pframe parameters that are _reused are 'reused' have the form something like t
--	|chapter=[[wikisource:{{{chapter}}}|{{{chapter}}}]]
-- where a parameter in the wrapping template is modified and then passed to the
-- using the same parameter name (in this example |chapter=)


	for k, v in ipairs (reuse_list) do
		if pframe_args[v] then
			pframe_args[v] = nil;
		end
	end

	pframe_args_get (pframe_args, args, exclude, _include_positional, list);

	return template;
end


--[[--------------------------< W R A P >-----------------------------------------

Template entry point.  Call this function to 'execute' the working template

]]

local function wrap (frame)
	local args = {};
	local template;

	template = _main (frame, args, false);
	if not template then
		return error_msg;
	end

	return frame:expandTemplate {title=template, args=args};
end


--[[--------------------------< L I S T >-----------------------------------------

Template entry point.  Call this function to 'display' the source for the workin
as a result of a TfD here: Wikipedia:Templates_for_discussion/Log/2018_April_28#N

This function replaces a similarly named function which was used in {{cite compa

Values in the args table are numerically indexed strings in the form 'name=value'
```

```
]]

local function list(frame, do_link)
	local args = {};                                                     -- table
	local template;                                                          --

	template = _main (frame, args, true);        -- get default and live para
	if not template then                                             -- template r
		return error_msg;                                                 -- emit
	end
	if do_link then
		template = ('[[%s|%s]]'):format(frame:expandTemplate{ title='Tran
	end
	table.sort(args)
	for i = 1, #args do
		local stripped = args[i]:match('^' .. i .. '=([^=]*)$')
		if stripped then args[i] = stripped else break end
	end
	return frame:preprocess(table.concat({
		'<code style="color:inherit; background:inherit; border:none;">&#
		template,
		('<wbr><nowiki>|%s</nowiki>'):rep(#args):format(unpack(args)), '&
end

local function link (frame)
	return list(frame, true)
end

--[[--------------------------< E X P O R T E D   F U N C T I O N S >-----------
]]

return {
	link = link,
	list = list,
	wrap = wrap,
	};
```