Module:Template wrapper

From Blue Gold Program Wiki
The printable version is no longer supported and may have rendering errors. Please update your browser bookmarks and please use the default browser print function instead.

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

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

local error_msg = '<span style=\"font-size:100%\" class=\"error\"><code
style=\"color:inherit; border:inherit;
padding:inherit;\">&#124;_template=</code> missing or empty</span>';


--[[--------------------------< 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
boolean list argument; kv pair for
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}));
-- write parameter names and values to args table as string
        else
                args[k] = v;
-- copy parameters to args table
        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 template names that match where
in [key]=<value> pairs where:
        [key] is local template parameter name (an alias)
        <value> is target template parameter name (the canonical parameter
name used in the working template)
```

```
The parameter |_alias-map= has the form:
        |_alias-map=<list>
where <list> is a comma-separated list of alias / canonical parameter name
pairs in the form
        <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
digits are placed in the parameter names:
                <from#> : <to#>

]]

local function alias_map_get (_alias_map)
        local T = mw.text.split (_alias_map, '%s*,%s*');
-- convert the comma-separated list into a table of alias pairs
        local mapped_aliases = {};
-- mapped aliases will go here
        local l_name, t_name;
-- parameter names
        for _, alias_pair in ipairs (T) do
-- loop through the table of alias pairs
                l_name, t_name = alias_pair:match ('(.-)%s*:%s*(.+)');
-- from each pair, get local and target parameter names
                if l_name and t_name then
-- if both are set
                        if tonumber (l_name) then
                                l_name = tonumber (l_name);
-- convert number-as-text to a number
                        end
                        mapped_aliases[l_name] = t_name;
-- add them to the map table
                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
parameters to args

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
-- here we get the wrapper template's 'default' parameters
                if 'string' == type (k) and (v and ('' ~= v)) then
-- do not pass along positional or empty parameters
                        if '_template' == k then
                                template = v;
-- save the name of template that we are wrapping
                        elseif '_exclude' ~= k and '_reuse' ~= k and
'_include-positional' ~= k  and '_alias-map' ~= k then        -- these
already handled so ignore here;
                                add_parameter (k, v, args, list);
-- add all other parameters to args in the style dictated by list
                        end
                end
        end

        return template;
-- return contents of |_template= parameter
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't members of the exclude table to
args table; positional parameters may not be excluded

no return value

]=]

local function pframe_args_get (pframe_args, args, exclude,
_include_positional, list)
        for k, v in pairs (pframe_args) do
                if 'string' == type (k) and not is_in_table (exclude, k) then
-- do not pass along excluded parameters
                        if v and ('' ~= v) then
-- pass along only those parameters that have assigned values
                                if 'unset' == v:lower() then
-- special keyword to unset 'default' parameters set in the wrapper template
                                        v = '';
-- unset the value in the args table
                                end
                                add_parameter (k, v, args, list)
-- add all other parameters to args in the style dictated by list; alias map
only supported for local-template parameters
                        end
```

```
                end
        end

        if _include_positional then
                for i, v in ipairs (pframe_args) do
-- pass along positional parameters
                        if 'unset' == v:lower() then
-- special keyword to unset 'default' parameters set in the wrapper template
                                v = '';
-- unset the value in the args table
                        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
boolean list.

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

]]

local function _main (frame, args, list)
        local template;
        local exclude = {};
-- table of parameter names for parameters that are not passed to the working
template
        local reuse_list = {};
-- table of pframe parameter names whose values are modified before they are
passed to the working template as the same name
        local alias_map = {};
-- table that maps parameter aliases to working template canonical parameter
names
        local _include_positional;
        if frame.args._exclude and ('' ~= frame.args._exclude) then
-- if there is |_exclude= and it's not empty
                exclude = mw.text.split (frame.args._exclude, "%s*,%s*");
-- make a table from its contents
        end
-- TODO: |_reuse= needs a better name (|_reuse=)
        if frame.args._reuse and ('' ~= frame.args._reuse) then
-- if there is |_reuse= and it's not empty
                reuse_list = mw.text.split (frame.args._reuse, "%s*,%s*");
-- make a table from its contents
        end
```

```
        if frame.args['_alias-map'] and ('' ~= frame.args['_alias-map']) then
-- if there is |_alias-map= and it's not empty
                alias_map = alias_map_get (frame.args['_alias-map']);
-- make a table from its contents
        end

        template = frame_args_get (frame.args, args, list);
-- get parameters provided in the {{#invoke:template wrapper|...|...}}
        if nil == template or '' == template then
-- this is the one parameter that is required by this module
                return nil;
-- not present, tell calling function to emit an error message
        end
        _include_positional = 'yes' == frame.args['_include-positional'];
-- when true pass all positional parameters along with non-excluded named
parameters to ...
-- ... the working template; positional parameters are not excludable
        local _pframe_args = frame:getParent().args;
-- here we get the wrapper template's 'live' parameters from pframe.args
        local pframe_args = {};
-- a local table that we can modify

        for k, v in pairs (_pframe_args) do
-- make a copy that we can modify
                pframe_args[k] = v;
        end
-- here we look for pframe parameters that are aliases of canonical parameter
names; when found
-- we replace the alias with the canonical.  We do this here because the
reuse_list works on
-- canonical parameter names so first we convert alias parameter names to
canonical names and then
-- we remove those canonical names from the pframe table that are reused
(provided to the working
-- template through the frame args table)

        for k, v in pairs (alias_map) do
-- k is alias name, v is canonical name
                if pframe_args[k] then
-- if pframe_args has parameter with alias name
                        pframe_args[v] = _pframe_args[k];
-- create new canonical name with alias' value
                        pframe_args[k] = nil;
-- unset the alias
                end
        end

        for k, v in pairs (pframe_args) do
-- do enumerated parameter alias -> canonical translation
                if 'string' == type (k) then
-- only named parameters can be enumerated
```

```
                        if alias_map[k..'#'] then
-- non-enumerated alias matches enumerated parameter pattern? enumerator at
end only
                                pframe_args[alias_map[k..'#']:gsub('#', '')]
= v;                                -- remove '#' and copy parameter to
pframe_args table
                                pframe_args[k] = nil;
-- unset the alias
                        elseif k:match ('%d+') then
-- if this parameter name contains digits
                                local temp = k:gsub ('%d+', '#');
-- make a copy; digits replaced with single '#'
                                local enum = k:match ('%d+');
-- get the enumerator
                                if alias_map[temp] then
-- if this parameter is a recognized enumerated alias
                                        pframe_args[alias_map[temp]:gsub('#',
enum)] = v;                          -- use canonical name and replace '#' with
enumerator and add to pframe_args
                                        pframe_args[k] = nil;
-- unset the alias
                                end
                        end
                end
        end

-- pframe parameters that are _reused are 'reused' have the form something
like this:
--      |chapter=[[wikisource:{{{chapter}}}|{{{chapter}}}]]
-- where a parameter in the wrapping template is modified and then passed to
the working template
-- using the same parameter name (in this example |chapter=)

-- remove parameters that will be reused
        for k, v in ipairs (reuse_list) do
-- k is numerical index, v is canonical parameter name to ignore
                if pframe_args[v] then
-- if pframe_args has parameter that should be ignored
                        pframe_args[v] = nil;
-- unset the ignored parameter
                end
        end

        pframe_args_get (pframe_args, args, exclude, _include_positional,
list);        -- add parameters and values to args that are not listed in the
exclude table

        return template;
-- args now has all default and live parameters, return working template name
end
```

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

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

]]

local function wrap (frame)
        local args = {};
-- table of default and live parameters and their values to be passed to the
wrapped template
        local template;
-- the name of the working template

        template = _main (frame, args, false);
-- get default and live parameters and the name of the working template
        if not template then
-- template name is required
                return error_msg;
-- emit error message and abandon if template name not present
        end

        return frame:expandTemplate {title=template, args=args};
-- render the working template
end


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

Template entry point.  Call this function to 'display' the source for the
working template.  This function added
as a result of a TfD here:
Wikipedia:Templates_for_discussion/Log/2018_April_28#Module:PassArguments

This function replaces a similarly named function which was used in {{cite
compare}} and {{cite compare2}}

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

]]


local function list (frame)
        local args = {};
-- table of default and live parameters and their values to be passed to the
listed template
        local template;
-- the name of the listed template
```

```
        template = _main (frame, args, true);
-- get default and live parameters and the name of the listed template
        if not template then
-- template name is required
                return error_msg;
-- emit error message and abandon if template name not present
        end

        return frame:preprocess (table.concat ({'<code style="color:inherit;
background:inherit; border:none;"><nowiki>{{', template, ' |', table.concat(
args, ' |' ), '}}</nowiki></code>'}));        -- render the template
end


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

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

Retrieved from "https://www.bluegoldwiki.com/index.php?title=Module:Template_wrapper&oldid=3534"

Namespaces

- Module
- Discussion

Variants

This page was last edited on 23 August 2020, at 06:04.

Blue Gold Program Wiki

The wiki version of the Lessons Learnt Report of the Blue Gold program, documents the experiences of a technical assistance (TA) team working in a development project implemented by the Bangladesh Water Development Board (BWDB) and the Department of Agricultural Extension (DAE) over an eight+ year period from March 2013 to December 2021. The wiki lessons learnt report (LLR) is intended to complement the BWDB and DAE project completion reports (PCRs), with the aim of recording lessons learnt for use in the design and implementation of future interventions in the coastal zone.

Developed and maintained by Big Blue Communications for Blue Gold Program

- Powered by MediaWiki

Blue Gold Program Wiki