Module:Arguments

Wechseln zu:Navigation, Suche

WikipediaEN:Module:Arguments

```lua
-- This module provides easy processing of arguments passed to Scribunto
from
-- #invoke. It is intended for use by other Lua modules, and should not
be
-- called from #invoke directly.

local libraryUtil = require('libraryUtil')
local checkType = libraryUtil.checkType

local arguments = {}

-- Generate four different tidyVal functions, so that we don't have to
check the
-- options every time we call it.

local function tidyValDefault(key, val)
	if type(val) == 'string' then
		val = val:match('^%s*(.-)%s*$')
		if val == '' then
			return nil
		else
			return val
		end
	else
		return val
	end
end

local function tidyValTrimOnly(key, val)
	if type(val) == 'string' then
		return val:match('^%s*(.-)%s*$')
	else
		return val
	end
end

local function tidyValRemoveBlanksOnly(key, val)
	if type(val) == 'string' then
		if val:find('%S') then
			return val
		else
			return nil
		end
	else
		return val
	end
end

local function tidyValNoChange(key, val)
```

```
        return val
end

local function matchesTitle(given, title)
        local tp = type( given )
        return (tp == 'string' or tp == 'number') and mw.title.new( given
).prefixedText == title
end

local translate_mt = { __index = function(t, k) return k end }

function arguments.getArgs(frame, options)
        checkType('getArgs', 1, frame, 'table', true)
        checkType('getArgs', 2, options, 'table', true)
        frame = frame or {}
        options = options or {}

        --[[
       -- Set up argument translation.
       --]]
        options.translate = options.translate or {}
        if getmetatable(options.translate) == nil then
                setmetatable(options.translate, translate_mt)
        end
        if options.backtranslate == nil then
                options.backtranslate = {}
                for k,v in pairs(options.translate) do
                        options.backtranslate[v] = k
                end
        end
        if options.backtranslate and getmetatable(options.backtranslate)
== nil then
                setmetatable(options.backtranslate, {
                        __index = function(t, k)
                                if options.translate[k] ~= k then
                                        return nil
                                else
                                        return k
                                end
                        end
                })
        end

        --[[
       -- Get the argument tables. If we were passed a valid frame
object, get the
       -- frame arguments (fargs) and the parent frame arguments (pargs),
depending
       -- on the options set and on the parent frame's availability. If
we weren't
```

```
	-- passed a valid frame object, we are being called from another
Lua module
	-- or from the debug console, so assume that we were passed a
table of args
	-- directly, and assign it to a new variable (luaArgs).
	--]]
	 local fargs, pargs, luaArgs
	 if type(frame.args) == 'table' and type(frame.getParent) ==
'function' then
		if options.wrappers then
			 --[[
			-- The wrappers option makes Module:Arguments look
up arguments in
			-- either the frame argument table or the parent
argument table, but
			-- not both. This means that users can use either
the #invoke syntax
			-- or a wrapper template without the loss of
performance associated
			-- with looking arguments up in both the frame and
the parent frame.
			-- Module:Arguments will look up arguments in the
parent frame
			-- if it finds the parent frame's title in
options.wrapper;
			-- otherwise it will look up arguments in the
frame object passed
			-- to getArgs.
			--]]
			 local parent = frame:getParent()
			 if not parent then
				fargs = frame.args
			 else
				local title =
parent:getTitle():gsub('/sandbox$', '')
				local found = false
				if matchesTitle(options.wrappers, title)
then
					found = true
				elseif type(options.wrappers) == 'table'
then
					for _,v in
pairs(options.wrappers) do
						if matchesTitle(v, title)
then
							found = true
							break
						end
					end
				end
```

```
                                -- We test for false specifically here so
that nil (the default) acts like true.
                                if found or options.frameOnly == false
then
                                        pargs = parent.args
                                end
                                if not found or options.parentOnly ==
false then
                                        fargs = frame.args
                                end
                        end
                else
                        -- options.wrapper isn't set, so check the other
options.
                        if not options.parentOnly then
                                fargs = frame.args
                        end
                        if not options.frameOnly then
                                local parent = frame:getParent()
                                pargs = parent and parent.args or nil
                        end
                end
                if options.parentFirst then
                        fargs, pargs = pargs, fargs
                end
        else
                luaArgs = frame
        end

        -- Set the order of precedence of the argument tables. If the
variables are
        -- nil, nothing will be added to the table, which is how we avoid
clashes
        -- between the frame/parent args and the Lua args.
        local argTables = {fargs}
        argTables[#argTables + 1] = pargs
        argTables[#argTables + 1] = luaArgs

        --[[
       -- Generate the tidyVal function. If it has been specified by the
user, we
       -- use that; if not, we choose one of four functions depending on
the
       -- options chosen. This is so that we don't have to call the
options table
       -- every time the function is called.
       --]]
        local tidyVal = options.valueFunc
        if tidyVal then
                if type(tidyVal) ~= 'function' then
```

```
                        error(
                                "bad value assigned to option
'valueFunc'"
                                        .. '(function expected, got '
                                        .. type(tidyVal)
                                        .. ')',
                                2
                        )
                end
        elseif options.trim ~= false then
                if options.removeBlanks ~= false then
                        tidyVal = tidyValDefault
                else
                        tidyVal = tidyValTrimOnly
                end
        else
                if options.removeBlanks ~= false then
                        tidyVal = tidyValRemoveBlanksOnly
                else
                        tidyVal = tidyValNoChange
                end
        end

        --[[
       -- Set up the args, metaArgs and nilArgs tables. args will be the
one
       -- accessed from functions, and metaArgs will hold the actual
arguments. Nil
       -- arguments are memoized in nilArgs, and the metatable connects
all of them
       -- together.
       --]]
        local args, metaArgs, nilArgs, metatable = {}, {}, {}, {}
        setmetatable(args, metatable)

        local function mergeArgs(tables)
                --[[
               -- Accepts multiple tables as input and merges their keys
and values
               -- into one table. If a value is already present it is not
overwritten;
               -- tables listed earlier have precedence. We are also
memoizing nil
               -- values, which can be overwritten if they are 's'
(soft).
               --]]
                for _, t in ipairs(tables) do
                        for key, val in pairs(t) do
                                if metaArgs[key] == nil and nilArgs[key]
~= 'h' then
```

```
                                        local tidiedVal = tidyVal(key,
val)
                                        if tidiedVal == nil then
                                                nilArgs[key] = 's'
                                        else
                                                metaArgs[key] = tidiedVal
                                        end
                                end
                        end
                end
        end

        --[[
        -- Define metatable behaviour. Arguments are memoized in the
metaArgs table,
        -- and are only fetched from the argument tables once. Fetching
arguments
        -- from the argument tables is the most resource-intensive step in
this
        -- module, so we try and avoid it where possible. For this reason,
nil
        -- arguments are also memoized, in the nilArgs table. Also, we
keep a record
        -- in the metatable of when pairs and ipairs have been called, so
we do not
        -- run pairs and ipairs on the argument tables more than once. We
also do
        -- not run ipairs on fargs and pargs if pairs has already been
run, as all
        -- the arguments will already have been copied over.
        --]]

        metatable.__index = function (t, key)
                --[[
                -- Fetches an argument when the args table is indexed.
First we check
                -- to see if the value is memoized, and if not we try and
fetch it from
                -- the argument tables. When we check memoization, we need
to check
                -- metaArgs before nilArgs, as both can be non-nil at the
same time.
                -- If the argument is not present in metaArgs, we also
check whether
                -- pairs has been run yet. If pairs has already been run,
we return nil.
                -- This is because all the arguments will have already
been copied into
                -- metaArgs by the mergeArgs function, meaning that any
other arguments
```

```
                -- must be nil.
                --]]
                if type(key) == 'string' then
                        key = options.translate[key]
                end
                local val = metaArgs[key]
                if val ~= nil then
                        return val
                elseif metatable.donePairs or nilArgs[key] then
                        return nil
                end
                for _, argTable in ipairs(argTables) do
                        local argTableVal = tidyVal(key, argTable[key])
                        if argTableVal ~= nil then
                                metaArgs[key] = argTableVal
                                return argTableVal
                        end
                end
                nilArgs[key] = 'h'
                return nil
        end

        metatable.__newindex = function (t, key, val)
                -- This function is called when a module tries to add a
new value to the
                -- args table, or tries to change an existing value.
                if type(key) == 'string' then
                        key = options.translate[key]
                end
                if options.readOnly then
                        error(
                                'could not write to argument table key "'
                                        .. tostring(key)
                                        .. '"; the table is read-only',
                                2
                        )
                elseif options.noOverwrite and args[key] ~= nil then
                        error(
                                'could not write to argument table key "'
                                        .. tostring(key)
                                        .. '"; overwriting existing
arguments is not permitted',
                                2
                        )
                elseif val == nil then
                        --[[
                        -- If the argument is to be overwritten with nil,
we need to erase
                        -- the value in metaArgs, so that __index, __pairs
and __ipairs do
```

```
                        -- not use a previous existing value, if present;
and we also need
                        -- to memoize the nil in nilArgs, so that the
value isn't looked
                        -- up in the argument tables if it is accessed
again.
                        --]]
                        metaArgs[key] = nil
                        nilArgs[key] = 'h'
                else
                        metaArgs[key] = val
                end
        end

        local function translatenext(invariant)
                local k, v = next(invariant.t, invariant.k)
                invariant.k = k
                if k == nil then
                        return nil
                elseif type(k) ~= 'string' or not options.backtranslate
then
                        return k, v
                else
                        local backtranslate = options.backtranslate[k]
                        if backtranslate == nil then
                                -- Skip this one. This is a tail call, so
this won't cause stack overflow
                                return translatenext(invariant)
                        else
                                return backtranslate, v
                        end
                end
        end

        metatable.__pairs = function ()
                -- Called when pairs is run on the args table.
                if not metatable.donePairs then
                        mergeArgs(argTables)
                        metatable.donePairs = true
                end
                return translatenext, { t = metaArgs }
        end

        local function inext(t, i)
                -- This uses our __index metamethod
                local v = t[i + 1]
                if v ~= nil then
                        return i + 1, v
                end
        end
```

```
        metatable.__ipairs = function (t)
                -- Called when ipairs is run on the args table.
                return inext, t, 0
        end

        return args
end

return arguments
```