Navigation

Tools

Personal tools

personal-extra

Random page

Views

Actions

Module:Infobox

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:Infobox/doc

```
--
-- This module implements {{Infobox}}
--

local p = {}

local navbar = require('Module:Navbar')._navbar

local args = {}
local origArgs = {}
local root

local function notempty( s ) return s and s:match( '%S' ) end

local function fixChildBoxes(sval, tt)
        if notempty(sval) then
                local marker = '<span class=special_infobox_marker>'
                local s = sval
                s = mw.ustring.gsub(s, '(<%s*[Tt][Rr])', marker .. '%1')
                s = mw.ustring.gsub(s, '(</[Tt][Rr]%s*>)', '%1' .. marker)
                if s:match(marker) then
                        s = mw.ustring.gsub(s, marker .. '%s*' .. marker, '')
                        s = mw.ustring.gsub(s, '([\r\n]|-[^\r\n]*[\r\n])%s*'
.. marker, '%1')
                        s = mw.ustring.gsub(s, marker .. '%s*([\r\n]|-)',
'%1')
                        s = mw.ustring.gsub(s,
'(</[Cc][Aa][Pp][Tt][Ii][Oo][Nn]%s*>%s*)' .. marker, '%1')
                        s = mw.ustring.gsub(s,
'(<%s*[Tt][Aa][Bb][Ll][Ee][^<>]*>%s*)' .. marker, '%1')
                        s = mw.ustring.gsub(s, '^(%{|[^\r\n]*[\r\n]%s*)' ..
marker, '%1')
                        s = mw.ustring.gsub(s, '([\r\n]%{|[^\r\n]*[\r\n]%s*)'
.. marker, '%1')
                        s = mw.ustring.gsub(s, marker ..
'(%s*</[Tt][Aa][Bb][Ll][Ee]%s*>)', '%1')
                        s = mw.ustring.gsub(s, marker .. '(%s*\n|%})', '%1')
                end
                if s:match(marker) then
                        local subcells = mw.text.split(s, marker)
                        s = ''
                        for k = 1, #subcells do
                                if k == 1 then
                                        s = s .. subcells[k] .. '</' .. tt ..
'></tr>'
                                elseif k == #subcells then
                                        local rowstyle = '
style="display:none"'
                                        if notempty(subcells[k]) then
rowstyle = ''        end
                                        s = s .. '<tr' .. rowstyle ..'><' ..
```

```
tt .. ' colspan=2>\n' .. subcells[k]
                                elseif notempty(subcells[k]) then
                                        if (k % 2) == 0 then
                                                s = s .. subcells[k]
                                        else
                                                s = s .. '<tr><' .. tt .. '
colspan=2>\n' .. subcells[k] .. '</' .. tt .. '></tr>'
                                        end
                                end
                        end
                end
                -- the next two lines add a newline at the end of lists for
the PHP parser
                --
https://en.wikipedia.org/w/index.php?title=Template_talk:Infobox_musical_arti
st&oldid=849054481
                -- remove when [[:phab:T191516]] is fixed or OBE
                s = mw.ustring.gsub(s, '([\r\n][%*#;:][^\r\n]*)$', '%1\n')
                s = mw.ustring.gsub(s, '^([%*#;:][^\r\n]*)$', '%1\n')
                s = mw.ustring.gsub(s, '^([%*#;:])', '\n%1')
                s = mw.ustring.gsub(s, '^(%{%|)', '\n%1')
                return s
        else
                return sval
        end
end

local function union(t1, t2)
        -- Returns the union of the values of two tables, as a sequence.
        local vals = {}
        for k, v in pairs(t1) do
                vals[v] = true
        end
        for k, v in pairs(t2) do
                vals[v] = true
        end
        local ret = {}
        for k, v in pairs(vals) do
                table.insert(ret, k)
        end
        return ret
end

local function getArgNums(prefix)
        -- Returns a table containing the numbers of the arguments that exist
        -- for the specified prefix. For example, if the prefix was 'data',
and
        -- 'data1', 'data2', and 'data5' exist, it would return {1, 2, 5}.
        local nums = {}
        for k, v in pairs(args) do
                local num = tostring(k):match('^' .. prefix .. '([1-9]%d*)$')
```

```
                if num then table.insert(nums, tonumber(num)) end
        end
        table.sort(nums)
        return nums
end

local function addRow(rowArgs)
        -- Adds a row to the infobox, with either a header cell
        -- or a label/data cell combination.
        if rowArgs.header and rowArgs.header ~= '_BLANK_' then
                root
                        :tag('tr')
                                :addClass(rowArgs.rowclass)
                                :cssText(rowArgs.rowstyle)
                                :attr('id', rowArgs.rowid)
                                :tag('th')
                                        :attr('colspan', 2)
                                        :attr('id', rowArgs.headerid)
                                        :addClass(rowArgs.class)
                                        :addClass(args.headerclass)
                                        :css('text-align', 'center')
                                        :cssText(args.headerstyle)
                                        :cssText(rowArgs.rowcellstyle)
:wikitext(fixChildBoxes(rowArgs.header, 'th'))
                if rowArgs.data then
                        root:wikitext('[[Category:Pages which use infobox
templates with ignored data cells]]')
                end
        elseif rowArgs.data then
                if not
rowArgs.data:gsub('%[%[%s*[Cc][Aa][Tt][Ee][Gg][Oo][Rr][Yy]%s*:[^]]*]]',
''):match('^%S') then
                        rowArgs.rowstyle = 'display:none'
                end
                local row = root:tag('tr')
                row:addClass(rowArgs.rowclass)
                row:cssText(rowArgs.rowstyle)
                row:attr('id', rowArgs.rowid)
                if rowArgs.label then
                        row
                                :tag('th')
                                        :attr('scope', 'row')
                                        :attr('id', rowArgs.labelid)
                                        :cssText(args.labelstyle)
                                        :cssText(rowArgs.rowcellstyle)
                                        :wikitext(rowArgs.label)
                                        :done()
                end

                local dataCell = row:tag('td')
                if not rowArgs.label then
```

```
                        dataCell
                                :attr('colspan', 2)
                                :css('text-align', 'center')
                end
                dataCell
                        :attr('id', rowArgs.dataid)
                        :addClass(rowArgs.class)
                        :cssText(rowArgs.datastyle)
                        :cssText(rowArgs.rowcellstyle)
                        :wikitext(fixChildBoxes(rowArgs.data, 'td'))
        end
end

local function renderTitle()
        if not args.title then return end

        root
                :tag('caption')
                        :addClass(args.titleclass)
                        :cssText(args.titlestyle)
                        :wikitext(args.title)
end

local function renderAboveRow()
        if not args.above then return end

        root
                :tag('tr')
                        :tag('th')
                                :attr('colspan', 2)
                                :addClass(args.aboveclass)
                                :css('text-align', 'center')
                                :css('font-size', '125%')
                                :css('font-weight', 'bold')
                                :cssText(args.abovestyle)
                                :wikitext(fixChildBoxes(args.above,'th'))
end

local function renderBelowRow()
        if not args.below then return end

        root
                :tag('tr')
                        :tag('td')
                                :attr('colspan', '2')
                                :addClass(args.belowclass)
                                :css('text-align', 'center')
                                :cssText(args.belowstyle)
                                :wikitext(fixChildBoxes(args.below,'td'))
end
```

```
local function renderSubheaders()
	if args.subheader then
		args.subheader1 = args.subheader
	end
	if args.subheaderrowclass then
		args.subheaderrowclass1 = args.subheaderrowclass
	end
	local subheadernums = getArgNums('subheader')
	for k, num in ipairs(subheadernums) do
		addRow({
			data = args['subheader' .. tostring(num)],
			datastyle = args.subheaderstyle,
			rowcellstyle = args['subheaderstyle' ..
tostring(num)],
			class = args.subheaderclass,
			rowclass = args['subheaderrowclass' .. tostring(num)]
		})
	end
end

local function renderImages()
	if args.image then
		args.image1 = args.image
	end
	if args.caption then
		args.caption1 = args.caption
	end
	local imagenums = getArgNums('image')
	for k, num in ipairs(imagenums) do
		local caption = args['caption' .. tostring(num)]
		local data = mw.html.create():wikitext(args['image' ..
tostring(num)])
		if caption then
			data
				:tag('div')
					:cssText(args.captionstyle)
					:wikitext(caption)
		end
		addRow({
			data = tostring(data),
			datastyle = args.imagestyle,
			class = args.imageclass,
			rowclass = args['imagerowclass' .. tostring(num)]
		})
	end
end

local function preprocessRows()
	-- Gets the union of the header and data argument numbers,
	-- and renders them all in order using addRow.
	local rownums = union(getArgNums('header'), getArgNums('data'))
```

```
        table.sort(rownums)
        local lastheader
        for k, num in ipairs(rownums) do
                if args['header' .. tostring(num)] then
                        if lastheader then
                                args['header' .. tostring(lastheader)] = nil
                        end
                        lastheader = num
                elseif args['data' .. tostring(num)] and args['data' ..
tostring(num)]:gsub('%[%[%s*[Cc][Aa][Tt][Ee][Gg][Oo][Rr][Yy]%s*:[^]]*]]',
''):match('^%S') then
                        local data = args['data' .. tostring(num)]
                        if
data:gsub('%[%[%s*[Cc][Aa][Tt][Ee][Gg][Oo][Rr][Yy]%s*:[^]]*]]',
''):match('%S') then
                                lastheader = nil
                        end
                end
        end
        if lastheader then
                args['header' .. tostring(lastheader)] = nil
        end
end

local function renderRows()
        -- Gets the union of the header and data argument numbers,
        -- and renders them all in order using addRow.
        local rownums = union(getArgNums('header'), getArgNums('data'))
        table.sort(rownums)
        for k, num in ipairs(rownums) do
                addRow({
                        header = args['header' .. tostring(num)],
                        label = args['label' .. tostring(num)],
                        data = args['data' .. tostring(num)],
                        datastyle = args.datastyle,
                        class = args['class' .. tostring(num)],
                        rowclass = args['rowclass' .. tostring(num)],
                        rowstyle = args['rowstyle' .. tostring(num)],
                        rowcellstyle = args['rowcellstyle' .. tostring(num)],
                        dataid = args['dataid' .. tostring(num)],
                        labelid = args['labelid' .. tostring(num)],
                        headerid = args['headerid' .. tostring(num)],
                        rowid = args['rowid' .. tostring(num)]
                })
        end
end

local function renderNavBar()
        if not args.name then return end

        root
```

```
                :tag('tr')
                        :tag('td')
                                :attr('colspan', '2')
                                :css('text-align', 'right')
                                :wikitext(navbar{
                                        args.name,
                                        mini = 1,
                                })
end

local function renderItalicTitle()
        local italicTitle = args['italic title'] and
mw.ustring.lower(args['italic title'])
        if italicTitle == '' or italicTitle == 'force' or italicTitle ==
'yes' then
                root:wikitext(mw.getCurrentFrame():expandTemplate({title =
'italic title'}))
        end
end

local function renderTrackingCategories()
        if args.decat ~= 'yes' then
                if args.child == 'yes' then
                        if args.title then
                                root:wikitext('[[Category:Pages which use
embedded infobox templates with the title parameter]]')
                        end
                elseif #(getArgNums('data')) == 0 and
mw.title.getCurrentTitle().namespace == 0 then
                        root:wikitext('[[Category:Articles which use infobox
templates with no data rows]]')
                end
        end
end

local function _infobox()
        -- Specify the overall layout of the infobox, with special settings
        -- if the infobox is used as a 'child' inside another infobox.
        if args.child ~= 'yes' then
                root = mw.html.create('table')

                root
                        :addClass((args.subbox ~= 'yes') and 'infobox' or
nil)
                        :addClass(args.bodyclass)

                        if args.subbox == 'yes' then
                                root
                                        :css('padding', '0')
                                        :css('border', 'none')
                                        :css('margin', '-3px')
```

```
                                        :css('width', 'auto')
                                        :css('min-width', '100%')
                                        :css('font-size', '100%')
                                        :css('clear', 'none')
                                        :css('float', 'none')
                                        :css('background-color',
'transparent')
                        else
                                root
                                        :css('width', '22em')
                        end
                root
                        :cssText(args.bodystyle)

                renderTitle()
                renderAboveRow()
        else
                root = mw.html.create()

                root
                        :wikitext(args.title)
        end

        renderSubheaders()
        renderImages()
        if args.autoheaders then
                preprocessRows()
        end
        renderRows()
        renderBelowRow()
        renderNavBar()
        renderItalicTitle()
        renderTrackingCategories()

        return tostring(root)
end

local function preprocessSingleArg(argName)
        -- If the argument exists and isn't blank, add it to the argument
table.
        -- Blank arguments are treated as nil to match the behaviour of
ParserFunctions.
        if origArgs[argName] and origArgs[argName] ~= '' then
                args[argName] = origArgs[argName]
        end
end

local function preprocessArgs(prefixTable, step)
        -- Assign the parameters with the given prefixes to the args table,
in order, in batches
        -- of the step size specified. This is to prevent references etc.
```

```
from appearing in the
        -- wrong order. The prefixTable should be an array containing tables,
each of which has
        -- two possible fields, a "prefix" string and a "depend" table. The
function always parses
        -- parameters containing the "prefix" string, but only parses
parameters in the "depend"
        -- table if the prefix parameter is present and non-blank.
        if type(prefixTable) ~= 'table' then
                error("Non-table value detected for the prefix table", 2)
        end
        if type(step) ~= 'number' then
                error("Invalid step value detected", 2)
        end

        -- Get arguments without a number suffix, and check for bad input.
        for i,v in ipairs(prefixTable) do
                if type(v) ~= 'table' or type(v.prefix) ~= "string" or
(v.depend and type(v.depend) ~= 'table') then
                        error('Invalid input detected to preprocessArgs
prefix table', 2)
                end
                preprocessSingleArg(v.prefix)
                -- Only parse the depend parameter if the prefix parameter is
present and not blank.
                if args[v.prefix] and v.depend then
                        for j, dependValue in ipairs(v.depend) do
                                if type(dependValue) ~= 'string' then
                                        error('Invalid "depend" parameter
value detected in preprocessArgs')
                                end
                                preprocessSingleArg(dependValue)
                        end
                end
        end

        -- Get arguments with number suffixes.
        local a = 1 -- Counter variable.
        local moreArgumentsExist = true
        while moreArgumentsExist == true do
                moreArgumentsExist = false
                for i = a, a + step - 1 do
                        for j,v in ipairs(prefixTable) do
                                local prefixArgName = v.prefix .. tostring(i)
                                if origArgs[prefixArgName] then
                                        moreArgumentsExist = true -- Do
another loop if any arguments are found, even blank ones.
                                        preprocessSingleArg(prefixArgName)
                                end
                                -- Process the depend table if the prefix
argument is present and not blank, or
```

```
                                -- we are processing "prefix1" and "prefix"
is present and not blank, and
                                -- if the depend table is present.
                                if v.depend and (args[prefixArgName] or (i ==
1 and args[v.prefix])) then
                                        for j,dependValue in ipairs(v.depend)
do
                                                local dependArgName =
dependValue .. tostring(i)
preprocessSingleArg(dependArgName)
                                        end
                                end
                        end
                end
                a = a + step
        end
end

local function parseDataParameters()
        -- Parse the data parameters in the same order that the old
{{infobox}} did, so that
        -- references etc. will display in the expected places. Parameters
that depend on
        -- another parameter are only processed if that parameter is present,
to avoid
        -- phantom references appearing in article reference lists.
        preprocessSingleArg('autoheaders')
        preprocessSingleArg('child')
        preprocessSingleArg('bodyclass')
        preprocessSingleArg('subbox')
        preprocessSingleArg('bodystyle')
        preprocessSingleArg('title')
        preprocessSingleArg('titleclass')
        preprocessSingleArg('titlestyle')
        preprocessSingleArg('above')
        preprocessSingleArg('aboveclass')
        preprocessSingleArg('abovestyle')
        preprocessArgs({
                {prefix = 'subheader', depend = {'subheaderstyle',
'subheaderrowclass'}}
        }, 10)
        preprocessSingleArg('subheaderstyle')
        preprocessSingleArg('subheaderclass')
        preprocessArgs({
                {prefix = 'image', depend = {'caption', 'imagerowclass'}}
        }, 10)
        preprocessSingleArg('captionstyle')
        preprocessSingleArg('imagestyle')
        preprocessSingleArg('imageclass')
        preprocessArgs({
                {prefix = 'header'},
```

```
                {prefix = 'data', depend = {'label'}},
                {prefix = 'rowclass'},
                {prefix = 'rowstyle'},
                {prefix = 'rowcellstyle'},
                {prefix = 'class'},
                {prefix = 'dataid'},
                {prefix = 'labelid'},
                {prefix = 'headerid'},
                {prefix = 'rowid'}
        }, 50)
        preprocessSingleArg('headerclass')
        preprocessSingleArg('headerstyle')
        preprocessSingleArg('labelstyle')
        preprocessSingleArg('datastyle')
        preprocessSingleArg('below')
        preprocessSingleArg('belowclass')
        preprocessSingleArg('belowstyle')
        preprocessSingleArg('name')
        args['italic title'] = origArgs['italic title'] -- different
behaviour if blank or absent
        preprocessSingleArg('decat')
end

function p.infobox(frame)
        -- If called via #invoke, use the args passed into the invoking
template.
        -- Otherwise, for testing purposes, assume args are being passed
directly in.
        if frame == mw.getCurrentFrame() then
                origArgs = frame:getParent().args
        else
                origArgs = frame
        end
        parseDataParameters()
        return _infobox()
end

function p.infoboxTemplate(frame)
        -- For calling via #invoke within a template
        origArgs = {}
        for k,v in pairs(frame.args) do origArgs[k] = mw.text.trim(v) end
        parseDataParameters()
        return _infobox()
end
return p
```


Namespaces

- Module
- Discussion

Variants

This page was last edited on 15 November 2020, at 11:23.

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.