Modul:Val/sandbox

Die Dokumentation für dieses Modul kann unter Modul:Val/sandbox/Doku erstellt werden

```
-- For Template:Val, output a number and optional unit.
-- Format options include scientific and uncertainty notations.

local data_module = 'Module:Val/units'
local convert_module = 'Module:Convert'

local function valerror(msg, nocat, iswarning)
        -- Return formatted message text for an error or warning.
        -- Can append "#FormattingError" to URL of a page with a problem to find
        local anchor = '<span id="FormattingError"></span>'
        local body, category
        if nocat or mw.title.getCurrentTitle():inNamespaces(1, 2, 3, 5) then
                -- No category in Talk, User, User_talk, or Wikipedia_talk.
                category = ''
        else
                category = '[[Category:Pages with incorrect formatting templates
        end
        iswarning = false  -- problems are infrequent so try showing large error
        if iswarning then
                body = '<sup class="noprint Inline-Template" style="white-space:r
                        '[[Template:Val|<span title="' ..
                        msg:gsub('"', '&quot;') ..
                        '">warning</span>]]</sup>'
        else
                body = '<strong class="error">' ..
                        'Error in &#123;&#123;[[Template:val|val]]&#125;&#125;: '
                        msg ..
                        '</strong>'
        end
        return anchor .. body .. category
end

local range_types = {
        -- No need for ' ' because nowrap applies to all output.
        [","]   = ", ",
        ["by"]  = " by ",
        ["-"]   = "–",
        ["–"]   = "–",
        ["and"] = " and ",
        ["or"]  = " or " ,
        ["to"]  = " to " ,
        ["x"]   = " × ",
        ["×"]   = " × ",
        ["/"]   = "/",
}
local range_repeat_unit = {
        -- WP:UNIT wants unit repeated when a "multiply" range is used.
        ["x"]   = true,
        ["×"]   = true,
}

local function extract_item(index, numbers, arg)
        -- Extract an item from arg and store the result in numbers[index].
        -- If no argument or if argument is valid, return nil (no error);
        -- otherwise, return an error message.
        -- The stored result is:
```

```
	-- * a table for a number (empty if there was no specified number); or
	-- * a string for range text
	-- Input like 1e3 is regarded as invalid for all except argument 1
	-- which accepts e notation as an alternative to the 'e' argument.
	-- Input commas are removed so 1,234 is the same as 1234.
	local which = index
	local function fail(msg)
		local description
		if which == 'e' then
			description = 'exponent parameter (<b>e</b>)'
		else
			description = 'parameter ' .. which
		end
		return description .. ' ' .. (msg or 'is not a valid number') ..
	end
	local result = {}
	local range = range_types[arg]
	if range then
		if type(index) == 'number' and (index % 2 == 0) then
			if index == 2 then
				if numbers[1] and numbers[1].exp then
					return fail('cannot use a range if the fi
				end
				numbers.has_ranges = true
			else
				if not numbers.has_ranges then
					return fail('needs a range in parameter 2
				end
			end
			numbers[index] = range
			if range_repeat_unit[arg] then
				-- Any "repeat" range forces unit (if any) to be
				numbers.isrepeat = true
			end
			return nil
		end
		return fail('does not accept a range')
	end
	if numbers.has_ranges and type(index) == 'number' and (index % 2 == 0) th
		return fail('should be a range')
	end
	if index == 'e' then
		local e = numbers[1] and numbers[1].exp
		if e then
			if arg then
				return fail('cannot be used if the first paramete
			end
			arg = e
			which = 1
		end
	end
	if arg and arg ~= '' then
		arg = arg:gsub(',', '')
		if arg:sub(1, 1) == '(' and arg:sub(-1) == ')' then
			result.parens = true
			arg = arg:sub(2, -2)
		end
		local a, b = arg:match('^(.+)[Ee](.+)$')
		if a then
			if index == 1 then
				arg = a
				result.exp = b
			else
				return fail('cannot use e notation')
```

```
                        end
                end
                local isnegative, propersign, prefix
                local minus = '−'
                prefix, arg = arg:match('^(.-)([%d.]+)$')
                local value = tonumber(arg)
                if not value then
                        return fail()
                end
                if arg:sub(1, 1) == '.' then
                        arg = '0' .. arg
                end
                if prefix == '' then
                        -- Ignore.
                elseif prefix == '±' then
                        -- Display for first number, ignore for others.
                        if index == 1 then
                                propersign = '±'
                        end
                elseif prefix == '+' then
                        propersign = '+'
                elseif prefix == '-' or prefix == minus then
                        propersign = minus
                        isnegative = true
                else
                        return fail()
                end
                result.clean = arg
                result.sign = propersign or ''
                result.value = isnegative and -value or value
        end
        numbers[index] = result
        return nil  -- no error
end

local function get_args(numbers, args)
        -- Extract arguments and store the results in numbers.
        -- Return nothing (no error) if ok; otherwise, return an error message.
        for index = 1, 99 do
                local which = index
                local arg = args[which]  -- has been trimmed
                if not arg then
                        which = 'e'
                        arg = args[which]
                end
                local msg = extract_item(which, numbers, arg)
                if msg then
                        return msg
                end
                if which == 'e' then
                        break
                end
                if index > 19 then
                        return 'too many parameters'
                end
        end
        if numbers.has_ranges and (#numbers % 2 == 0) then
                return 'need a number after the last parameter because it is a ra
        end
end

local function get_scale(text, ucode)
        -- Return the value of text as a number, or throw an error.
        -- This supports extremely basic expressions of the form:
```

```
	--   a / b
	--   a ^ b
	-- where a and b are numbers or 'pi'.
	local n = tonumber(text)
	if n then
		return n
	end
	n = text:gsub('pi', math.pi)
	for _, op in ipairs({ '/', '^' }) do
		local a, b = n:match('^(.-)' .. op .. '(.*)$')
		if a then
			a = tonumber(a)
			b = tonumber(b)
			if a and b then
				if op == '/' then
					return a / b
				elseif op == '^' then
					return a ^ b
				end
			end
			break
		end
	end
	error('Unit "' .. ucode .. '" has invalid scale "' .. text .. '"')
end

local function get_builtin_unit(ucode, definitions)
	-- Return table of information for the specified built-in unit, or nil if
	-- Each defined unit code must be followed by two spaces (not tab charact
	local _, pos = definitions:find('\n' .. ucode .. '  ', 1, true)
	if pos then
		local endline = definitions:find('%s*\n', pos)
		if endline then
			local result = {}
			local n = 0
			local text = definitions:sub(pos + 1, endline - 1):gsub('
			for item in (text .. '\t'):gmatch('(%S.-)\t') do
				if item == 'ALIAS' then
					result.alias = true
				elseif item == 'ANGLE' then
					result.isangle = true
					result.nospace = true
				elseif item == 'NOSPACE' then
					result.nospace = true
				elseif item == 'SI' then
					result.si = true
				else
					n = n + 1
					if n == 1 then
						local link, symbol = item:match('
						if link then
							result.symbol = symbol
							result.link = link
							n = 2
						else
							result.symbol = item
						end
					elseif n == 2 then
						result.link = item
					elseif n == 3 then
						result.scale_text = item
						result.scale = get_scale(item, uc
					else
						result.more_ignored = item
```

```
                                                break
                                        end
                                end
                        end
                        if result.si then
                                local s = result.symbol
                                if ucode == 'mc' .. s or ucode == 'mu' .. s then
                                        result.ucode = 'µ' .. s  -- unit code for
                                end
                        end
                        if n >= 2 or (n >= 1 and result.alias) then
                                return result
                        end
                        -- Ignore invalid definition, treating it as a comment.
                end
        end
end

local function convert_lookup(ucode, value, scaled_top, want_link, si, options)
        local lookup = require(convert_module)._unit
        return lookup(ucode, {
                        value = value,
                        scaled_top = scaled_top,
                        link = want_link,
                        si = si,
                        sort = options.sortable,
                })
end

local function get_unit(ucode, value, scaled_top, options)
        local want_link = options.want_link
        if scaled_top then
                want_link = options.want_per_link
        end
        local data = mw.loadData(data_module)
        local result = options.want_longscale and
                get_builtin_unit(ucode, data.builtin_units_long_scale) or
                get_builtin_unit(ucode, data.builtin_units)
        local si, use_convert
        if result then
                if result.alias then
                        ucode = result.symbol
                        use_convert = true
                end
                if result.scale then
                        -- Setting si means convert will use the unit as given, a
                        -- will be calculated from the value without any extra sc
                        -- occur if convert found the unit code. For example, if
                        -- unit 'year' with a scale and if si were not set, conve
                        -- its own scale because convert knows that a year is 31
                        si = { result.symbol, result.link }
                        value = value * result.scale
                end
                if result.si then
                        ucode = result.ucode or ucode
                        si = { result.symbol, result.link }
                        use_convert = true
                end
        else
                result = {}
                use_convert = true
        end
        local convert_unit = convert_lookup(ucode, value, scaled_top, want_link,
        result.sortkey = convert_unit.sortspan
```

```
        if use_convert then
                result.text = convert_unit.text
                result.scaled_top = convert_unit.scaled_value
        else
                if want_link then
                        result.text = '[[' .. result.link .. '|' .. result.symbol
                else
                        result.text = result.symbol
                end
                result.scaled_top = value
        end
        return result
end

local function makeunit(value, options)
        -- Return table of information for the requested unit and options, or
        -- return nil if no unit.
        options = options or {}
        local unit
        local ucode = options.u
        local percode = options.per
        if ucode then
                unit = get_unit(ucode, value, nil, options)
        elseif percode then
                unit = { nospace = true, scaled_top = value }
        else
                return nil
        end
        local text = unit.text or ''
        local sortkey = unit.sortkey
        if percode then
                local function bracketed(code, text)
                        return code:find('[*./]') and '(' .. text .. ')' or text
                end
                local perunit = get_unit(percode, 1, unit.scaled_top, options)
                text = (ucode and bracketed(ucode, text) or '') ..
                                '/' .. bracketed(percode, perunit.text)
                sortkey = perunit.sortkey
        end
        if not (unit.nospace or options.nospace) then
                text = ' ' .. text
        end
        return { text = text, isangle = unit.isangle, sortkey = sortkey }
end

local function list_units(mode)
        -- Return wikitext to list the built-in units.
        -- A unit code should not contain wikimarkup so don't bother escaping.
        local data = mw.loadData(data_module)
        local definitions = data.builtin_units .. data.builtin_units_long_scale
        local last_was_blank = true
        local n = 0
        local result = {}
        local function add(line)
                if line == '' then
                        last_was_blank = true
                else
                        if last_was_blank and n > 0 then
                                n = n + 1
                                result[n] = ''
                        end
                        last_was_blank = false
                        n = n + 1
                        result[n] = line
```

```
		end
	end
	local si_prefixes = {
		-- These are the prefixes recognized by convert; u is accepted fo
		y = 'y',
		z = 'z',
		a = 'a',
		f = 'f',
		p = 'p',
		n = 'n',
		u = 'µ',
		['µ'] = 'µ',
		m = 'm',
		c = 'c',
		d = 'd',
		da = 'da',
		h = 'h',
		k = 'k',
		M = 'M',
		G = 'G',
		T = 'T',
		P = 'P',
		E = 'E',
		Z = 'Z',
		Y = 'Y',
	}
	local function is_valid(ucode, unit)
		if unit and not unit.more_ignored then
			assert(type(unit.symbol) == 'string' and unit.symbol ~= 
			if unit.alias then
				if unit.link or unit.scale_text or unit.si then
					return false
				end
			end
			if unit.si then
				if unit.scale_text then
					return false
				end
				ucode = unit.ucode or ucode
				local base = unit.symbol
				if ucode == base then
					unit.display = base
					return true
				end
				local plen = #ucode - #base
				if plen > 0 then
					local prefix = si_prefixes[ucode:sub(1, p
					if prefix and ucode:sub(plen + 1) == base
						unit.display = prefix .. base
						return true
					end
				end
			else
				unit.display = unit.symbol
				return true
			end
		end
		return false
	end
	local lookup = require(convert_module)._unit
	local function show_convert(ucode, unit)
		-- If a built-in unit defines a scale or sets the SI flag, any ur
		-- convert is not used (the scale or SI prefix's scale is used fo
		-- If there is no scale or SI flag, and the unit is not defined i
```

```
                -- the sort key may not be correct; this allows such units to be
                if not (unit.si or unit.scale_text) then
                        if mode == 'convert' then
                                unit.show = not lookup(unit.alias and unit.symbol
                                unit.show_text = 'CONVERT'
                        elseif mode == 'unknown' then
                                unit.show = lookup(unit.alias and unit.symbol or
                                unit.show_text = 'UNKNOWN'
                        elseif not unit.alias then
                                -- Show convert's scale in square brackets ('[1]'
                                -- Don't show scale for an alias because it's mis
                                -- and an alias is probably not useful for anythi
                                local scale = lookup(ucode, {value=1, sort='on'})
                                if type(scale) == 'number' then
                                        scale = string.format('%.5g', scale):gsub
                                else
                                        scale = '?'
                                end
                                unit.show = true
                                unit.show_text = '[' .. scale .. ']'
                        end
                end
        end
        for line in definitions:gmatch('([^\n]*)\n') do
                local pos, _ = line:find('  ', 1, true)
                if pos then
                        local ucode = line:sub(1, pos - 1)
                        local unit = get_builtin_unit(ucode, '\n' .. line .. '\n'
                        if is_valid(ucode, unit) then
                                show_convert(ucode, unit)
                                local flags, text
                                if unit.alias then
                                        text = unit.symbol
                                else
                                        text = '[[' .. unit.link .. '|' .. unit.c
                                end
                                if unit.isangle then
                                        unit.nospace = nil  -- don't show redunda
                                end
                                for _, f in ipairs({
                                                { 'alias', 'ALIAS' },
                                                { 'isangle', 'ANGLE' },
                                                { 'nospace', 'NOSPACE' },
                                                { 'si', 'SI' },
                                                { 'scale_text', unit.scale_text }
                                                { 'show', unit.show_text },
                                        }) do
                                        if unit[f[1]] then
                                                local t = f[2]
                                                if t:match('^%u+$') then
                                                        t = '<small>' .. t .. '</
                                                end
                                                if flags then
                                                        flags = flags .. ' ' .. t
                                                else
                                                        flags = t
                                                end
                                        end
                                end
                                if flags then
                                        text = text .. ' • ' .. flags
                                end
                                add(ucode .. ' = ' .. text .. '<br />')
                        else
```

```
                                add(line .. ' ◆ <b>invalid definition</b><br />')
                        end
                else
                        add(line)
                end
        end
        return table.concat(result, '\n')
end

local delimit_groups = require('Module:Gapnum').groups
local function delimit(sign, numstr, fmt)
        -- Return sign and numstr (unsigned digits or '.' only) after formatting
        -- Four-digit integers are not formatted with gaps.
        fmt = (fmt or ''):lower()
        if fmt == 'none' or (fmt == '' and #numstr == 4 and numstr:match('^%d+$')
                return sign .. numstr
        end
        -- Group number by integer and decimal parts.
        -- If there is no decimal part, delimit_groups returns only one table.
        local ipart, dpart = delimit_groups(numstr)
        local result
        if fmt == 'commas' then
                result = sign .. table.concat(ipart, ',')
                if dpart then
                        result = result .. '.' .. table.concat(dpart)
                end
        else
                -- Delimit with a small gap by default.
                local groups = {}
                groups[1] = table.remove(ipart, 1)
                for _, v in ipairs(ipart) do
                        table.insert(groups, '<span style="margin-left:.25em;">'
                end
                if dpart then
                        table.insert(groups, '.' .. (table.remove(dpart, 1) or '
                        for _, v in ipairs(dpart) do
                                table.insert(groups, '<span style="margin-left:.2
                        end
                end
                result = sign .. table.concat(groups)
        end
        return result
end

local function sup_sub(sup, sub, align)
        -- Return the same result as Module:Su except val defaults to align=right
        if align == 'l' or align == 'left' then
                align = 'left'
        elseif align == 'c' or align == 'center' then
                align = 'center'
        else
                align = 'right'
        end
        return '<span style="display:inline-block;margin-bottom:-0.3em;vertical-a
                align .. ';">' .. sup .. '<br />' .. sub .. '</span>'
end

local function range_text(items, unit_table, options)
        local fmt = options.fmt
        local nend = items.nend or ''
        if items.isrepeat or unit_table.isangle then
                nend = nend .. unit_table.text
        end
        local text = ''
```

```
        for i = 1, #items do
                if i % 2 == 0 then
                        text = text .. items[i]
                else
                        text = text .. delimit(items[i].sign, items[i].clean, fmt
                end
        end
        return text
end

local function uncertainty_text(uncertainty, unit_table, options)
        local angle, text, need_parens
        if unit_table.isangle then
                angle = unit_table.text
        end
        local upper = uncertainty.upper or {}
        local lower = uncertainty.lower or {}
        local uncU = upper.clean
        if uncU then
                local fmt = options.fmt
                local uncL = lower.clean
                if uncL then
                        uncU = delimit('+', uncU, fmt) .. (upper.errend or '')
                        uncL = delimit('−', uncL, fmt) .. (lower.errend or '')
                        if angle then
                                uncU = uncU .. angle
                                uncL = uncL .. angle
                        end
                        text = (angle or '') ..
                                '<span style="margin-left:0.3em;">' ..
                                sup_sub(uncU, uncL, options.align) ..
                                '</span>'
                else
                        if upper.parens then
                                text = '(' .. uncU .. ')'  -- old template did nc
                        else
                                text = (angle or '') ..
                                        '<span style="margin-left:0.3em;margin-ri
                                        delimit('', uncU, fmt)
                                need_parens = true
                        end
                        if uncertainty.errend then
                                text = text .. uncertainty.errend
                        end
                        if angle then
                                text = text .. angle
                        end
                end
        else
                if angle then
                        text = angle
                end
        end
        return text, need_parens
end

local function _main(values, unit_spec, options)
        if options.sandbox then
                data_module = data_module .. '/sandbox'
                convert_module = convert_module .. '/sandbox'
        end
        local action = options.action
        if action then
                if action == 'list' then
```

```
                        -- Kludge: am using the align parameter (a=xxx) for type
                        return list_units(options.align)
                end
                return valerror('invalid action "' .. action .. '".', options.noc
        end
        local number = values.number or (values.numbers and values.numbers[1]) or
        local e_10 = options.e or {}
        local novalue = (number.value == nil and e_10.clean == nil)
        local fmt = options.fmt
        local want_sort = true
        local sortable = options.sortable
        if sortable == 'off' or (sortable == nil and novalue) then
                want_sort = false
        elseif sortable == 'debug' then
                -- Same as sortable = 'on' but the sort key is displayed.
        else
                sortable = 'on'
        end
        local sort_value = 1
        if want_sort then
                sort_value = number.value or 1
                if e_10.value and sort_value ~= 0 then
                        -- The 'if' avoids {{val|0|e=1234}} giving an invalid sor
                        sort_value = sort_value * 10^e_10.value
                end
        end
        local unit_table = makeunit(sort_value, {
                                                u = unit_spec.u,
                                                want_link = unit_spec.want_link,
                                                per = unit_spec.per,
                                                want_per_link = unit_spec.want_pe
                                                nospace = novalue,
                                                want_longscale = unit_spec.want_l
                                                sortable = sortable,
                                        })
        local sortkey
        if unit_table then
                if want_sort then
                        sortkey = unit_table.sortkey
                end
        else
                unit_table = { text = '' }
                if want_sort then
                        sortkey = convert_lookup('dummy', sort_value, nil, nil, n
                end
        end
        local final_unit = unit_table.isangle and '' or unit_table.text
        local e_text, n_text, need_parens
        local uncertainty = values.uncertainty
        if uncertainty then
                if number.clean then
                        n_text = delimit(number.sign, number.clean, fmt) .. (numb
                        local text
                        text, need_parens = uncertainty_text(uncertainty, unit_ta
                        if text then
                                n_text = n_text .. text
                        end
                else
                        n_text = ''
                end
        else
                if values.numbers.isrepeat then
                        final_unit = ''
                end
```

```
                n_text = range_text(values.numbers, unit_table, options)
                need_parens = true
        end
        if e_10.clean then
                if need_parens then
                        n_text = '(' .. n_text .. ')'
                end
                e_text = '10<sup>' .. delimit(e_10.sign, e_10.clean, fmt) .. '</s
                if number.clean then
                        e_text = '<span style="margin-left:0.25em;margin-right:0
                end
        else
                e_text = ''
        end
        local result =
                (sortkey or '') ..
                (options.prefix or '') ..
                n_text ..
                e_text ..
                final_unit ..
                (options.suffix or '')
        if result ~= '' then
                result = '<span class="nowrap">' .. result .. '</span>'
        end
        return result .. (options.warning or '')
end

local function check_parameters(args, has_ranges, nocat)
        -- Return warning text for the first problem parameter found, or nothing
        local whitelist = {
                a = true,
                action = true,
                debug = true,
                e = true,
                ['end'] = true,
                errend = true,
                ['+errend'] = true,
                ['-errend'] = true,
                fmt = true,
                ['long scale'] = true,
                long_scale = true,
                longscale = true,
                nocategory = true,
                p = true,
                s = true,
                sortable = true,
                u = true,
                ul = true,
                up = true,
                upl = true,
        }
        for k, v in pairs(args) do
                if type(k) == 'string' and not whitelist[k] then
                        local warning = 'Val parameter "' .. k .. '=' .. v .. '"
                        return valerror(warning, nocat, true)
                end
        end
        if not has_ranges and args[4] then
                return valerror('Val parameter 4 ignored', nocat, true)
        end
end

local function main(frame)
        local getArgs = require('Module:Arguments').getArgs
```

```
	local args = getArgs(frame, {wrappers = { 'Template:Val' }})
	local nocat = args.nocategory
	local numbers = {}  -- table of number tables, perhaps with range text
	local msg = get_args(numbers, args)
	if msg then
		return valerror(msg, nocat)
	end
	if args.u and args.ul then
		return valerror('unit (<b>u</b>) and unit with link (<b>ul</b>) a
	end
	if args.up and args.upl then
		return valerror('unit per (<b>up</b>) and unit per with link (<b>
	end
	local values
	if numbers.has_ranges then
		-- Multiple values with range separators but no uncertainty.
		numbers.nend = args['end']
		values = {
			numbers = numbers,
		}
	else
		-- A single value with optional uncertainty.
		local function setfield(i, dst, src)
			local v = args[src]
			if v then
				if numbers[i] then
					numbers[i][dst] = v
				else
					numbers[i] = { [dst] = v }
				end
			end
		end
		setfield(1, 'nend', 'end')
		setfield(2, 'errend', '+errend')
		setfield(3, 'errend', '-errend')
		values = {
			number = numbers[1],
			uncertainty = {
				upper = numbers[2],
				lower = numbers[3],
				errend = args.errend,
			}
		}
	end
	local unit_spec = {
			u = args.ul or args.u,
			want_link = args.ul ~= nil,
			per = args.upl or args.up,
			want_per_link = args.upl ~= nil,
			want_longscale = (args.longscale or args.long_scale or a
		}
	local options = {
			action = args.action,
			align = args.a,
			e = numbers.e,
			fmt = args.fmt,
			nocat = nocat,
			prefix = args.p,
			sandbox = string.find(frame:getTitle(), 'sandbox', 1, tru
			sortable = args.sortable or (args.debug == 'yes' and 'deb
			suffix = args.s,
```

```
                        warning = check_parameters(args, numbers.has_ranges, noca
                    }
            return _main(values, unit_spec, options)
end

return { main = main, _main = _main }
```