Modul:Convert/wikidata/sandbox

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

```
-- Functions to access Wikidata for Module:Convert.

local Collection = {}
Collection.__index = Collection
do
	function Collection:add(item)
		if item ~= nil then
			self.n = self.n + 1
			self[self.n] = item
		end
	end
	function Collection:join(sep)
		return table.concat(self, sep)
	end
	function Collection:remove(pos)
		if self.n > 0 and (pos == nil or (0 < pos and pos <= self.n)) the
			self.n = self.n - 1
			return table.remove(self, pos)
		end
	end
	function Collection:sort(comp)
		table.sort(self, comp)
	end
	function Collection.new()
		return setmetatable({n = 0}, Collection)
	end
end

local function strip_to_nil(text)
	-- If text is a non-empty string, return its trimmed content,
	-- otherwise return nothing (empty string or not a string).
	if type(text) == 'string' then
		return text:match('(%S.-)%s*$')
	end
end

local function frequency_unit(value, unit_table)
	-- For use when converting m to Hz.
	-- Return true, s where s = name of unit's default output unit,
	-- or return false, t where t is an error message table.
	-- However, for simplicity a valid result is always returned.
	local unit
	if unit_table._symbol == 'm' then
		-- c = speed of light in a vacuum = 299792458 m/s
		-- frequency = c / wavelength
		local w = value * (unit_table.scale or 1)
		local f = 299792458 / w  -- if w == 0, f = math.huge which works
		if f >= 1e12 then
			unit = 'THz'
		elseif f >= 1e9 then
			unit = 'GHz'
		elseif f >= 1e6 then
			unit = 'MHz'
		elseif f >= 1e3 then
```

```
                        unit = 'kHz'
                else
                        unit = 'Hz'
                end
        end
        return true, unit or 'Hz'
end

local function wavelength_unit(value, unit_table)
        -- Like frequency_unit but for use when converting Hz to m.
        local unit
        if unit_table._symbol == 'Hz' then
                -- Using 0.9993 rather than 1 avoids rounding which would give re
                -- like converting 300 MHz to 100 cm instead of 1 m.
                local w = 1 / (value * (unit_table.scale or 1))  -- Hz scale is i
                if w >= 0.9993e6 then
                        unit = 'Mm'
                elseif w >= 0.9993e3 then
                        unit = 'km'
                elseif w >= 0.9993 then
                        unit = 'm'
                elseif w >= 0.9993e-2 then
                        unit = 'cm'
                elseif w >= 0.9993e-3 then
                        unit = 'mm'
                else
                        unit = 'um'
                end
        end
        return true, unit or 'm'
end

local specials = {
        frequency = { frequency_unit },
        wavelength = { wavelength_unit },
        ------------------------------------------------------------------------
        -- Following is a removed experiment to show two values as a range
        -- using '-' as the separator.
        -- frequencyrange = { frequency_unit, '-' },
        -- wavelengthrange = { wavelength_unit, '-' },
}

local function make_unit(units, parms, uid)
        -- Return a unit code for convert or nil if unit unknown.
        -- If necessary, add a dummy unit to parms so convert will use it
        -- for the input without attempting a conversion since nothing
        -- useful is available (for example, with unit volt).
        local unit = units[uid]
        if type(unit) ~= 'table' then
                return nil
        end
        local ucode = unit.ucode
        if ucode and not unit.si then
                return ucode                -- a unit known to convert
        end
        parms.opt_ignore_error = true
        ucode = ucode or unit._ucode    -- must be a non-empty string
        local ukey, utable
        if unit.si then
                local base = units[unit.si]
                ukey = base.symbol          -- must be a non-empty string
                local n1 = base.name1
                local n2 = base.name2
                if not n1 then
```

```
                        n1 = ukey
                        n2 = n2 or n1           -- do not append 's'
                end
                utable = {
                        _symbol = ukey,
                        _name1 = n1,
                        _name2 = n2,
                        link = unit.link or base.link,
                        utype = n1,
                        prefixes = 1,
                }
        else
                ukey = ucode
                utable = {
                        symbol = ucode,         -- must be a non-empty string
                        name1 = unit.name1,     -- if nil, uses symbol
                        name2 = unit.name2,     -- if nil, uses name1..'s'
                        link = unit.link,       -- if nil, uses name1
                        utype = unit.name1 or ucode,
                }
        end
        utable.scale = 1
        utable.default = ''
        utable.defkey = ''
        utable.linkey = ''
        utable.bad_mcode = ''
        parms.unittable = { [ukey] = utable }
        return ucode
end

local function matches_qualifier(statement, qual)
        -- Return:
        --   false, nil : if statement does not match specification
        --   true, nil  : if matches, and statement has no qualifier
        --   true, sq   : if matches, where sq is the statement's qualifier
        -- A match means that no qualifier was specified (qual == nil), or that
        -- the statement has a qualifier matching the specification.
        -- If a match occurs, the caller needs the statement's qualifier (if any)
        -- so statements that duplicate the qualifier are not used, after the fi
        -- Then, if convert is showing all values for a property such as the diam
        -- of a telescope's mirror (diameters of primary and secondary mirrors),
        -- will not show alternative values that could in principle be present fo
        -- same item (telescope) and property (diameter) and qualifier (primary/s
        local target = (statement.qualifiers or {}).P518  -- P518 is "applies to
        if type(target) == 'table' then
                for _, q in ipairs(target) do
                        if type(q) == 'table' then
                                local value = (q.datavalue or {}).value
                                if value then
                                        if qual == nil or qual == value.id then
                                                return true, value.id
                                        end
                                end
                        end
                end
        end
        if qual == nil then
                return true, nil  -- only occurs if statement has no qualifier
        end
        return false, nil  -- statement's qualifier is not relevant because state
end

local function get_statements(parms, pid)
        -- Get specified item and return a list of tables with each statement for
```

```
        -- Each table is of form {statqual=sq, stmt=statement} where sq = stateme
        -- Statements are in Wikidata's order except that those with preferred ra
        -- are first, then normal rank. Any other rank is ignored.
        local stored = {}  -- qualifiers of statements that are first for the qua
        local qid = strip_to_nil(parms.qid)  -- nil for current page's item, or a
        local qual = strip_to_nil(parms.qual)  -- nil or id of wanted P518 (appli
        local result = Collection.new()
        local entity = mw.wikibase.getEntity(qid)
        if type(entity) == 'table' then
                local statements = (entity.claims or {})[pid]
                if type(statements) == 'table' then
                        for _, rank in ipairs({ 'preferred', 'normal' }) do
                                for _, statement in ipairs(statements) do
                                        if type(statement) == 'table' and rank ==
                                                local is_match, statqual = matche
                                                if is_match then
                                                        result:add({ statqual = s
                                                end
                                        end
                                end
                        end
                end
        end
        return result
end

local function input_from_property(tdata, parms, pid)
        -- Given that pid is a Wikidata property identifier like 'P123',
        -- return a collection of {amount, ucode} pairs (two strings)
        -- for each matching item/property, or return nothing.
        ------------------------------------------------------------------------
        -- There appear to be few restrictions on how Wikidata is organized so it
        -- very likely that any decision a module makes about how to handle data
        -- will be wrong for some cases at some time. This meets current requirem
        -- For each qualifier (or if no qualifier), if there are any preferred
        -- statements, use them and ignore any normal statements.
        -- For each qualifier, for the preferred statements if any, or for
        -- the normal statements (but not both):
        -- * Accept each statement if it has no qualifier (this will not occur
        --   if qual=x is specified because other code already ensures that in th
        --   case, only statements with a qualifier matching x are considered).
        -- * Ignore any statements after the first if it has a qualifier.
        -- The rationale is that for the diameter at [[South Pole Telescope]], wa
        -- convert to show the diameters for both the primary and secondary mirro
        -- if the convert does not specify which diameter is wanted.
        -- However, if convert is given the wanted qualifier, only one value
        -- (_the_ diameter) is wanted. For simplicity/consistency, that is also d
        -- even if no qual=x is specified. Unclear what should happen.
        -- For the wavelength at [[Nançay Radio Telescope]], want to show all thr
        -- values, and the values have no qualifiers.
        ------------------------------------------------------------------------
        local result = Collection.new()
        local done = {}
        local skip_normal
        for _, t in ipairs(get_statements(parms, pid)) do
                local statement = t.stmt
                if statement.mainsnak and statement.mainsnak.datatype == 'quantit
                        local value = (statement.mainsnak.datavalue or {}).value
                        if value then
                                local amount = value.amount
                                if amount then
                                        amount = tostring(amount)  -- in case amo
                                        if amount:sub(1, 1) == '+' then
                                                amount = amount:sub(2)
```

```
                                        end
                                        local unit = value.unit
                                        if type(unit) == 'string' then
                                                unit = unit:match('Q%d+$')  -- ur
                                                local ucode = make_unit(tdata.wik
                                                if ucode then
                                                        local skip
                                                        if t.statqual then
                                                                if done[t.statqua
                                                                        skip = tr
                                                                else
                                                                        done[t.st
                                                                end
                                                        else
                                                                if statement.rank
                                                                        skip_norm
                                                                elseif skip_norma
                                                                        skip = tr
                                                                end
                                                        end
                                                        if not skip then
                                                                result:add({ amou
                                                        end
                                                end
                                        end
                                end
                        end
                end
        end
        return result
end

local function input_from_text(tdata, parms, text, insert2)
        -- Given string should be of form "<value><space><unit>" or
        -- "<value1><space>ft<space><value2><space>in" for a special case (feet a
        -- Return true if values/units were extracted and inserted, or return not
        text = text:gsub(' ', ' '):gsub('%s+', ' ')
        local pos = text:find(' ', 1, true)
        if pos then
                -- Leave checking of value to convert which can handle fractions
                local value = text:sub(1, pos - 1)
                local uid = text:sub(pos + 1)
                if uid:sub(1, 3) == 'ft ' and uid:sub(-3) == ' in' then
                        -- Special case for enwiki to allow {{convert|input=5 ft
                        insert2(uid:sub(4, -4), 'in')
                        insert2(value, 'ft')
                else
                        insert2(value, make_unit(tdata.wikidata_units, parms, uid
                end
                return true
        end
end

local function adjustparameters(tdata, parms, index)
        -- For Module:Convert, adjust parms (a table of {{convert}} parameters).
        -- Return true if successful or return false, t where t is an error messa
        -- This is intended mainly for use in infoboxes where the input might be
        --    <value><space><unit>    or
        --    <wikidata-property-id>
        -- If successful, insert values and units in parms, before given index.
        local text = parms.input  -- should be a trimmed, non-empty string
        local pid = text:match('^P%d+$')
        local sep = ','
        local special = specials[parms[index]]
```

```
        if special then
                parms.out_unit = special[1]
                sep = special[2] or sep
                table.remove(parms, index)
        end
        local function quit()
                return false, pid and { 'cvt_no_output' } or { 'cvt_bad_input', t
        end
        local function insert2(first, second)
                table.insert(parms, index, second)
                table.insert(parms, index, first)
        end
        if pid then
                parms.input_text = ''  -- output an empty string if an error occu
                local result = input_from_property(tdata, parms, pid)
                if result.n == 0 then
                        return quit()
                end
                local ucode
                for i, t in ipairs(result) do
                        -- Convert requires each input unit to be identical.
                        if i == 1 then
                                ucode = t[2]
                        elseif ucode ~= t[2] then
                                return quit()
                        end
                end
                local item = ucode
                if item == parms[index] then
                        -- Remove specified output unit if it is the same as the
                        -- For example, {{convert|input=P2044|km}} with property
                        table.remove(parms, index)
                end
                for i = result.n, 1, -1 do
                        insert2(result[i][1], item)
                        item = sep
                end
                return true
        else
                if input_from_text(tdata, parms, text, insert2) then
                        return true
                end
        end
        return quit()
end

-------------------------------------------------------------------------------
--- List units and check syntax of definitions --------------------------------
-------------------------------------------------------------------------------
local specifications = {
        -- seq = sequence in which fields are displayed
        base = {
                title = 'SI base units',
                fields = {
                        symbol = { seq = 2, mandatory = true },
                        name1  = { seq = 3, mandatory = true },
                        name2  = { seq = 4 },
                        link   = { seq = 5 },
                },
                noteseq = 6,
                header = '{| class="wikitable"\n!si !!symbol !!name1 !!name2 !!li
                item = '|-\n|%s ||%s ||%s ||%s ||%s ||%s',
                footer = '|}',
        },
```

```
	alias = {
		title = 'Aliases for convert',
		fields = {
			ucode  = { seq = 2, mandatory = true },
			si     = { seq = 3 },
		},
		noteseq = 4,
		header = '{| class="wikitable"\n!alias !!ucode !!base !!note',
		item = '|-\n|%s ||%s ||%s ||%s',
		footer = '|}',
	},
	known = {
		title = 'Units known to convert',
		fields = {
			ucode  = { seq = 2, mandatory = true },
			label  = { seq = 3, mandatory = true },
		},
		noteseq = 4,
		header = '{| class="wikitable"\n!qid !!ucode !!label !!note',
		item = '|-\n|%s ||%s ||%s ||%s',
		footer = '|}',
	},
	unknown = {
		title = 'Units not known to convert',
		fields = {
			_ucode = { seq = 2, mandatory = true },
			si     = { seq = 3 },
			name1  = { seq = 4 },
			name2  = { seq = 5 },
			link   = { seq = 6 },
			label  = { seq = 7, mandatory = true },
		},
		noteseq = 8,
		header = '{| class="wikitable"\n!qid !!_ucode !!base !!name1 !!na
		item = '|-\n|%s ||%s ||%s ||%s ||%s ||%s ||%s',
		footer = '|}',
	},
}

local function listunits(tdata, ulookup)
	-- For Module:Convert, make wikitext to list the built-in Wikidata units
	-- Return true, wikitext if successful or return false, t where t is an
	-- error message table. Currently, an error return never occurs.
	-- The syntax of each unit definition is checked and a note is added if
	-- a problem is detected.
	local function safe_cells(t)
		-- This is not currently needed, but in case definitions ever use
		-- like '[[kilogram|kg]]', escape the text so it works in a table
		local result = {}
		for i, v in ipairs(t) do
			if v:find('|', 1, true) then
				v = v:gsub('(%[%[[^%[%]]-)|(.-%]%])', '%1\0%2')
				v = v:gsub('|', '&#124;')
				v = v:gsub('%z', '|')
			end
			result[i] = v:gsub('{', '&#123;')                    -- e
		end
		return unpack(result)
	end
	local wdunits = tdata.wikidata_units
	local speckeys = { 'base', 'alias', 'unknown', 'known' }
	for _, sid in ipairs(speckeys) do
		specifications[sid].units = Collection.new()
	end
```

```
	local keys = Collection.new()
	for k, v in pairs(wdunits) do
		keys:add(k)
	end
	table.sort(keys)
	local note_count = 0
	for _, key in ipairs(keys) do
		local unit = wdunits[key]
		local ktext, sid
		if key:match('^Q%d+$') then
			ktext = '[[d:' .. key .. '|' .. key .. ']]'
			if unit.ucode then
				sid = 'known'
			else
				sid = 'unknown'
			end
		elseif unit.ucode then
			ktext = key
			sid = 'alias'
		else
			ktext = key
			sid = 'base'
		end
		local result = { ktext }
		local spec = specifications[sid]
		local fields = spec.fields
		local note = Collection.new()
		for k, v in pairs(unit) do
			if fields[k] then
				local seq = fields[k].seq
				if result[seq] then
					note:add('duplicate ' .. k)  -- cannot ha
				else
					result[seq] = v
				end
			else
				note:add('invalid ' .. k)
			end
		end
		for k, v in pairs(fields) do
			local value = result[v.seq]
			if value then
				if k == 'si' and not wdunits[value] then
					note:add('need si ' .. value)
				end
				if k == 'label' then
					local wdl = mw.wikibase.getLabel(key)
					if wdl ~= value then
						note:add('label changed to ' .. t
					end
				end
			else
				result[v.seq] = ''
				if v.mandatory then
					note:add('missing ' .. k)
				end
			end
		end
		local text
		if note.n > 0 then
			note_count = note_count + 1
			text = '*' .. note:join('<br />')
		end
		result[spec.noteseq] = text or ''
```

```
                spec.units:add(result)
        end
        local results = Collection.new()
        if note_count > 0 then
                local text = note_count .. (note_count == 1 and ' note' or ' note
                results:add("'''Search for * to see " .. text .. "'''\n")
        end
        for _, sid in ipairs(speckeys) do
                local spec = specifications[sid]
                results:add("''''" .. spec.title .. "''''")
                results:add(spec.header)
                local fmt = spec.item
                for _, unit in ipairs(spec.units) do
                        results:add(string.format(fmt, safe_cells(unit)))
                end
                results:add(spec.footer)
        end
        return true, results:join('\n')
end

return { _adjustparameters = adjustparameters, _listunits = listunits }
```