



Inhaltsverzeichnis

1. Modul:Convert/wikidata	2
2. Modul:Convert/wikidata/data	11
3. Modul:Convert/wikidata/data/sandbox	26
4. Modul:Convert/wikidata/sandbox	41

Modul:Convert/wikidata

Die Dokumentation für dieses Modul kann unter [Modul:Convert/wikidata/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)) then
            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
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
    end
end
```

```
        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 first
    -- Then, if convert is showing all values for a property such as the diameter
    -- of a telescope's mirror (diameters of primary and secondary mirrors),
    -- will not show alternative values that could in principle be present for
    -- same item (telescope) and property (diameter) and qualifier (primary/secondary)
    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 statement
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 = statement's
    -- Statements are in Wikidata's order except that those with preferred rank
```

```
-- are first, then normal rank. Any other rank is ignored.
local stored = {} -- qualifiers of statements that are first for the qual
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 (appl
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
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 require
-- 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]], we
-- 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 c
-- even if no qual=x is specified. Unclear what should happen.
-- For the wavelength at [[Nançay Radio Telescope]], want to show all th
-- 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 amc
                if amount:sub(1, 1) == '+' then
                    amount = amount:sub(2)
                end
                local unit = value.unit
            end
        end
    end
end
end
```

```

        if type(unit) == 'string' then
            unit = unit:match('0%d+$') -- un
            local ucode = make_unit(tdata.wiki
            if ucode then
                local skip
                if t.statqual then
                    if done[t.statqua
                        skip = t
                    else
                        done[t.st
                    end
                else
                    if statement.rank
                        skip_norm
                    elseif skip_norma
                        skip = t
                    end
                end
                if not skip then
                    result:add({ amo
                end
            end
        end
    end
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('&nbsp;', ' '):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
    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]
    end
end
```

```
        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 occur
        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 !!!l
        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 !!name2 !!link !!label',
        item = '|-\n|%s ||%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;')
end
return unpack(result)
end
local wdunits = tdata.wikidata_units
local speckey = { 'base', 'alias', 'unknown', 'known' }
for _, sid in ipairs(speckey) 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.unicode then
                sid = 'known'
            else
                sid = 'unknown'
            end
        elseif unit.unicode 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 handle
                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 ' .. value)
                    end
                end
            else
                result[v.seq] = ''
                if v.mandatory then
                    note:add('missing ' .. k)
                end
            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
end
```

```
local results = Collection.new()
if note_count > 0 then
    local text = note_count .. (note_count == 1 and ' note' or ' notes')
    results:add("''Search for * to see " .. text .. ""'\n")
end
for _, sid in ipairs(speckey) 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 }
```



Modul:Convert/wikidata/data

Die Dokumentation für dieses Modul kann unter [Modul:Convert/wikidata/data/Doku](#) erstellt werden

```
--[[ Cache of Wikidata information with units for Module:Convert.
The codes should rarely change, and using a cache means that changing a
unit at Wikidata will not cause lots of converts in articles to break.

For a unit known to convert, the unit here must have:
    label = Wikidata label for unit (used only when listing units)
    ucode = unit code for input to convert
    (there are no optional fields because convert handles everything)

For a unit not known to convert, the unit here must have:
    label = Wikidata label for unit (used only when listing units)
    (no ucode field)
    _ucode = unit code for input to convert, and the
             symbol used to display the unit when abbr=on
    (convert will use the specified fields to display the unit,
    and will not attempt to do a conversion)

For a unit not known to convert, the unit here may have:
    name1 = singular name used to display the unit when abbr=off
    name2 = plural name used to display the unit when abbr=off
    link = name of article that unit will be linked to when lk=on
    si = key for the SI base unit, if any

The base unit for each SI unit here must have:
    symbol = symbol used to display the base unit when abbr=on
    name1 = singular name of base unit used to display the unit when abbr=off
    (if name1 is not given, symbol will be used, but an SI unit should have a
    name1)

The base unit for each SI unit here may have:
    name2 = plural name of base unit used to display the unit when abbr=off
    link = name of article that unit will be linked to when lk=on
    (applies for all SI units using this base, where the
    SI unit does not define its own link field)

If name1 is not specified, the symbol is used for the name.
If name2 is not specified, a plural name is formed by appending 's' to name1.
If link is not specified, name1 is used for the link.

SI units are assumed to be simple items like V (volt) where 'mV' would
cause convert to insert:
    'm' before the base symbol 'V' to make 'mV', if abbr=on
    'milli' before the base name 'volt' to make 'millivolt', if abbr=off
A unit like "square meter" would not work because it needs an SI prefix
inserted before "meter" rather than at the beginning of the name.

Items that should not be used with convert as no precise unit is implied:
Q11247037  ton          generic (cannot use)
Q178413   gallon       generic
Q130964   calorie      dubious (ambiguous, should not use)
Q216658   bushel       dubious
Q420266   fluid ounce  dubious
]]

local wikidata_units = {
```

```
-- Following are SI base units.
A = {
    symbol = 'A',
    name1 = 'ampere',
},
F = {
    symbol = 'F',
    name1 = 'faraday',
},
H = {
    symbol = 'H',
    name1 = 'henry',
},
V = {
    symbol = 'V',
    name1 = 'volt',
},
-- Following are aliases to convert unit codes, used with "input=<value>
kilograms = {
    ucode = 'kg',
},
-- Following are SI units not known to convert, used with "input=<value>
kV = {
    ucode = 'kV',
    si = 'V',
},
mV = {
    ucode = 'mV',
    si = 'V',
},
-- Following are Wikidata units.
Q131255 = {
    label = 'farad',
    _ucode = 'F',
    _si = 'F',
},
Q163354 = {
    label = 'henry',
    _ucode = 'H',
    _si = 'H',
},
Q1916026 = {
    label = 'microvolt',
    _ucode = 'uV',
    _si = 'V',
},
Q193933 = {
    label = 'dioptre',
    name1 = 'dioptre',
    _ucode = 'dpt',
},
Q212120 = {
    label = 'ampere hour',
    name1 = 'ampere hour',
    _ucode = 'A·h',
},
Q2448803 = {
    label = 'millivolt',
    _ucode = 'mV',
    _si = 'V',
},
Q2451296 = {
    label = 'microfarad',
    _ucode = 'uF',
```

```
        si = 'F',
    },
    Q2490574 = {
        label = 'milliampere',
        _ucode = 'mA',
        si = 'A',
    },
    Q25250 = {
        label = 'volt',
        _ucode = 'V',
        si = 'V',
    },
    Q25272 = {
        label = 'ampere',
        _ucode = 'A',
        si = 'A',
    },
    Q2553708 = {
        label = 'megavolt',
        _ucode = 'MV',
        si = 'V',
    },
    Q2554092 = {
        label = 'kilovolt',
        _ucode = 'kV',
        si = 'V',
    },
    Q2636421 = {
        label = 'nanohenry',
        _ucode = 'nH',
        si = 'H',
    },
    Q2679083 = {
        label = 'microhenry',
        _ucode = 'uH',
        si = 'H',
    },
    Q2682463 = {
        label = 'nanofarad',
        _ucode = 'nF',
        si = 'F',
    },
    Q2756030 = {
        label = 'picofarad',
        _ucode = 'pF',
        si = 'F',
    },
    Q2793566 = {
        label = 'gigavolt',
        _ucode = 'GV',
        si = 'V',
    },
    Q2924137 = {
        label = 'millihenry',
        _ucode = 'mH',
        si = 'H',
    },
    Q3117809 = {
        label = 'microampere',
        _ucode = 'uA',
        si = 'A',
    },
    Q33680 = {
        label = 'radian',
```

```
        name1 = 'radian',
        _ucode = 'rad',
    },
    Q4456994 = {
        label = 'millifarad',
        _ucode = 'mF',
        _si = 'F',
    },
    Q47083 = {
        label = 'ohm',
        name1 = 'ohm',
        _ucode = 'Ω',
    },
    Q483261 = {
        label = 'dalton',
        name1 = 'dalton',
        _ucode = 'u',
    },
    Q550341 = {
        label = 'volt-ampere',
        name1 = 'volt-ampere',
        _ucode = 'VA',
    },
    Q100995 = {
        label = 'pound',
        ucode = 'lb',
    },
    Q1022113 = {
        label = 'cubic centimetre',
        ucode = 'cc',
    },
    Q102573 = {
        label = 'becquerel',
        ucode = 'Bq',
    },
    Q103246 = {
        label = 'sievert',
        ucode = 'Sv',
    },
    Q1050958 = {
        label = 'inch of mercury',
        ucode = 'inHg',
    },
    Q1051665 = {
        label = 'metre per second squared',
        ucode = 'm/s2',
    },
    Q1052397 = {
        label = 'rad',
        ucode = 'rad',
    },
    Q1054140 = {
        label = 'megametre',
        ucode = 'Mm',
    },
    Q1057069 = {
        label = 'hectogram',
        ucode = 'hg',
    },
    Q1063786 = {
        label = 'square inch',
        ucode = 'sqin',
    },
    Q1092296 = {
```

```
        label = 'annum',
        ucode = 'year',
    },
    Q11570 = {
        label = 'kilogram',
        ucode = 'kg',
    },
    Q11573 = {
        label = 'metre',
        ucode = 'm',
    },
    Q11574 = {
        label = 'second',
        ucode = 's',
    },
    Q11579 = {
        label = 'kelvin',
        ucode = 'K',
    },
    Q11582 = {
        label = 'litre',
        ucode = 'litre',
    },
    Q1165588 = {
        label = 'rod',
        ucode = 'rod',
    },
    Q1165799 = {
        label = 'thou',
        ucode = 'thou',
    },
    Q11776930 = {
        label = 'megagram',
        ucode = 'Mg',
    },
    Q11929860 = {
        label = 'kiloparsec',
        ucode = 'kpc',
    },
    Q1194225 = {
        label = 'pound-force',
        ucode = 'lbf',
    },
    Q12129 = {
        label = 'parsec',
        ucode = 'pc',
    },
    Q12438 = {
        label = 'newton',
        ucode = 'N',
    },
    Q1255620 = {
        label = 'dram',
        ucode = 'drachm',
    },
    Q12874593 = {
        label = 'watt-hour',
        ucode = 'W.h',
    },
    Q128822 = {
        label = 'knot',
        ucode = 'kn',
    },
    Q1374438 = {
```

```
        label = 'kilosecond',
        ucode = 'ks',
    },
    Q1377051 = {
        label = 'gigasecond',
        ucode = 'Gs',
    },
    Q14754979 = {
        label = 'zettagram',
        ucode = 'Zg',
    },
    Q14786969 = {
        label = 'megajoule',
        ucode = 'MJ',
    },
    Q14787261 = {
        label = 'megawatt hour',
        ucode = 'MW.h',
    },
    Q1550511 = {
        label = 'square yard',
        ucode = 'sqyd',
    },
    Q160857 = {
        label = 'metric horsepower',
        ucode = 'hp',
    },
    Q1628990 = {
        label = 'horsepower-hour',
        ucode = 'hph',
    },
    Q163343 = {
        label = 'tesla',
        ucode = 'T',
    },
    Q1645498 = {
        label = 'microgram',
        ucode = 'ug',
    },
    Q17087835 = {
        label = 'cuerda',
        ucode = 'cda',
    },
    Q174728 = {
        label = 'centimetre',
        ucode = 'cm',
    },
    Q174789 = {
        label = 'millimetre',
        ucode = 'mm',
    },
    Q175821 = {
        label = 'micrometre',
        ucode = 'um',
    },
    Q1770733 = {
        label = 'teragram',
        ucode = 'Tg',
    },
    Q1772386 = {
        label = 'decigram',
        ucode = 'dg',
    },
    Q177493 = {
```

```
        label = 'gauss',
        ucode = 'G',
    },
    Q1777507 = {
        label = 'femtosecond',
        ucode = 'fs',
    },
    Q177974 = {
        label = 'standard atmosphere',
        ucode = 'atm',
    },
    Q178674 = {
        label = 'nanometre',
        ucode = 'nm',
    },
    Q180154 = {
        label = 'kilometre per hour',
        ucode = 'km/h',
    },
    Q180892 = {
        label = 'solar mass',
        ucode = 'solar mass',
    },
    Q1811 = {
        label = 'astronomical unit',
        ucode = 'au',
    },
    Q1815100 = {
        label = 'centilitre',
        ucode = 'cl',
    },
    Q182098 = {
        label = 'kilowatt hour',
        ucode = 'kW.h',
    },
    Q1823150 = {
        label = 'microwatt',
        ucode = 'uW',
    },
    Q182429 = {
        label = 'metre per second',
        ucode = 'm/s',
    },
    Q1826195 = {
        label = 'decilitre',
        ucode = 'dl',
    },
    Q185078 = {
        label = 'are',
        ucode = 'a',
    },
    Q185153 = {
        label = 'erg',
        ucode = 'erg',
    },
    Q185648 = {
        label = 'torr',
        ucode = 'Torr',
    },
    Q190095 = {
        label = 'gray',
        ucode = 'Gy',
    },
    Q191118 = {
```

```
        label = 'tonne',
        ucode = 'tonne',
    },
    Q1913097 = {
        label = 'femtogram',
        ucode = 'fg',
    },
    Q192274 = {
        label = 'picometre',
        ucode = 'pm',
    },
    Q1972579 = {
        label = 'poundal',
        ucode = 'pdl',
    },
    Q200323 = {
        label = 'decimetre',
        ucode = 'dm',
    },
    Q201933 = {
        label = 'dyne',
        ucode = 'dyn',
    },
    Q2029519 = {
        label = 'hectolitre',
        ucode = 'hl',
    },
    Q2051195 = {
        label = 'gigawatt hour',
        ucode = 'GW.h',
    },
    Q207488 = {
        label = 'Rankine scale',
        ucode = 'R',
    },
    Q208788 = {
        label = 'femtometre',
        ucode = 'fm',
    },
    Q2101 = {
        label = 'elementary charge',
        ucode = 'e',
    },
    Q21014455 = {
        label = 'metre per minute',
        ucode = 'm/min',
    },
    Q21062777 = {
        label = 'megapascal',
        ucode = 'MPa',
    },
    Q21064807 = {
        label = 'kilopascal',
        ucode = 'kPa',
    },
    Q211256 = {
        label = 'mile per hour',
        ucode = 'mph',
    },
    Q21178489 = {
        label = 'barrels per day',
        ucode = 'oilbbl/d',
    },
    Q2143992 = {
```

```
        label = 'kilohertz',
        ucode = 'kHz',
    },
    Q21467992 = {
        label = 'cubic foot per second',
        ucode = 'cuft/s',
    },
    Q215571 = {
        label = 'newton metre',
        ucode = 'Nm',
    },
    Q216795 = {
        label = 'dunam',
        ucode = 'dunam',
    },
    Q216880 = {
        label = 'kilogram-force',
        ucode = 'kgf',
    },
    Q18413919 = {
        label = 'centimetre per second',
        ucode = 'cm/s',
    },
    Q218593 = {
        label = 'inch',
        ucode = 'in',
    },
    Q2282891 = {
        label = 'microlitre',
        ucode = 'ul',
    },
    Q2282906 = {
        label = 'nanogram',
        ucode = 'ng',
    },
    Q229354 = {
        label = 'curie',
        ucode = 'Ci',
    },
    Q232291 = {
        label = 'square mile',
        ucode = 'sqmi',
    },
    Q2332346 = {
        label = 'millilitre',
        ucode = 'ml',
    },
    Q23387 = {
        label = 'week',
        ucode = 'week',
    },
    Q23823681 = {
        label = 'terawatt',
        ucode = 'TW',
    },
    Q23925410 = {
        label = 'gallon (UK)',
        ucode = 'impgal',
    },
    Q23925413 = {
        label = 'gallon (US)',
        ucode = 'USgal',
    },
    Q2438073 = {
```

```
        label = 'attogram',
        ucode = 'ag',
    },
    Q2474258 = {
        label = 'millisievert',
        ucode = 'mSv',
    },
    Q2483628 = {
        label = 'attosecond',
        ucode = 'as',
    },
    Q2489298 = {
        label = 'square centimetre',
        ucode = 'cm2',
    },
    Q2518569 = {
        label = 'nanosievert',
        ucode = 'nSv',
    },
    Q25235 = {
        label = 'hour',
        ucode = 'h',
    },
    Q25236 = {
        label = 'watt',
        ucode = 'W',
    },
    Q25267 = {
        label = 'degree Celsius',
        ucode = 'C',
    },
    Q25269 = {
        label = 'joule',
        ucode = 'J',
    },
    Q253276 = {
        label = 'mile',
        ucode = 'mi',
    },
    Q25343 = {
        label = 'square metre',
        ucode = 'm2',
    },
    Q25406 = {
        label = 'coulomb',
        ucode = 'coulomb',
    },
    Q25517 = {
        label = 'cubic metre',
        ucode = 'm3',
    },
    Q260126 = {
        label = 'Roentgen equivalent man',
        ucode = 'rem',
    },
    Q2612219 = {
        label = 'petagram',
        ucode = 'Pg',
    },
    Q2619500 = {
        label = 'foe',
        ucode = 'foe',
    },
    Q2637946 = {
```

```
        label = 'decalitre',
        ucode = 'dal',
    },
    Q2655272 = {
        label = 'exagram',
        ucode = 'Eg',
    },
    Q2691798 = {
        label = 'centigram',
        ucode = 'cg',
    },
    Q2739114 = {
        label = 'microsievert',
        ucode = 'uSv',
    },
    Q2799294 = {
        label = 'gigagram',
        ucode = 'Gg',
    },
    Q3013059 = {
        label = 'kiloannum',
        ucode = 'millennium',
    },
    Q305896 = {
        label = 'dots per inch',
        ucode = 'dpi',
    },
    Q3207456 = {
        label = 'milliwatt',
        ucode = 'mW',
    },
    Q3221356 = {
        label = 'yoctometre',
        ucode = 'ym',
    },
    Q3239557 = {
        label = 'picogram',
        ucode = 'pg',
    },
    Q3241121 = {
        label = 'milligram',
        ucode = 'mg',
    },
    Q3267417 = {
        label = 'terametre',
        ucode = 'Tm',
    },
    Q3270676 = {
        label = 'zeptometre',
        ucode = 'zm',
    },
    Q3276763 = {
        label = 'gigahertz',
        ucode = 'GHz',
    },
    Q3277907 = {
        label = 'exametre',
        ucode = 'Em',
    },
    Q3277915 = {
        label = 'zettametre',
        ucode = 'Zm',
    },
    Q3277919 = {
```

```
        label = 'petametre',
        ucode = 'Pm',
    },
    Q3312063 = {
        label = 'femtolitre',
        ucode = 'fl',
    },
    Q3320608 = {
        label = 'kilowatt',
        ucode = 'kW',
    },
    Q3332822 = {
        label = 'megaton of TNT',
        ucode = 'Mt(TNT)',
    },
    Q35852 = {
        label = 'hectare',
        ucode = 'ha',
    },
    Q3675550 = {
        label = 'cubic millimetre',
        ucode = 'mm3',
    },
    Q3710 = {
        label = 'foot',
        ucode = 'ft',
    },
    Q3773454 = {
        label = 'megaparsec',
        ucode = 'Mpc',
    },
    Q3902688 = {
        label = 'picolitre',
        ucode = 'pl',
    },
    Q3902709 = {
        label = 'picosecond',
        ucode = 'ps',
    },
    Q39369 = {
        label = 'hertz',
        ucode = 'Hz',
    },
    Q3972226 = {
        label = 'kilolitre',
        ucode = 'kl',
    },
    Q4068266 = {
        label = "apothecaries' drachm",
        ucode = 'drachm',
    },
    Q41803 = {
        label = 'gram',
        ucode = 'g',
    },
    Q4220561 = {
        label = 'kilometre per second',
        ucode = 'km/s',
    },
    Q42289 = {
        label = 'degree Fahrenheit',
        ucode = 'F',
    },
    Q4243638 = {
```

```
        label = 'cubic kilometre',
        ucode = 'km3',
    },
    Q44395 = {
        label = 'pascal',
        ucode = 'Pa',
    },
    Q48013 = {
        label = 'ounce',
        ucode = 'oz',
    },
    Q482798 = {
        label = 'yard',
        ucode = 'yd',
    },
    Q4989854 = {
        label = 'kilojoule',
        ucode = 'kJ',
    },
    Q4992853 = {
        label = 'kiloton of TNT',
        ucode = 'kt(TNT)',
    },
    Q5139563 = {
        label = 'hectopascal',
        ucode = 'hPa',
    },
    Q5151 = {
        label = 'month',
        ucode = 'month',
    },
    Q531 = {
        label = 'light-year',
        ucode = 'ly',
    },
    Q5465723 = {
        label = 'foot-poundal',
        ucode = 'ftpdL',
    },
    Q573 = {
        label = 'day',
        ucode = 'd',
    },
    Q577 = {
        label = 'year',
        ucode = 'year',
    },
    Q5879479 = {
        label = 'gigawatt',
        ucode = 'GW',
    },
    Q6003257 = {
        label = 'attometre',
        ucode = 'am',
    },
    Q613726 = {
        label = 'yottagram',
        ucode = 'Yg',
    },
    Q6170164 = {
        label = 'yoctogram',
        ucode = 'yg',
    },
    Q667419 = {
```

```
        label = 'long ton',
        ucode = 'LT',
    },
    Q673166 = {
        label = 'gravity of Earth',
        ucode = 'g0',
    },
    Q693944 = {
        label = 'grain',
        ucode = 'gr',
    },
    Q6982035 = {
        label = 'megawatt',
        ucode = 'MW',
    },
    Q712226 = {
        label = 'square kilometre',
        ucode = 'km2',
    },
    Q723733 = {
        label = 'millisecond',
        ucode = 'ms',
    },
    Q732454 = {
        label = 'megaannum',
        ucode = 'Myr',
    },
    Q732707 = {
        label = 'megahertz',
        ucode = 'MHz',
    },
    Q752079 = {
        label = 'gross register ton',
        ucode = 'grt',
    },
    Q752197 = {
        label = 'kilojoule per mole',
        ucode = 'kJ/mol',
    },
    Q7727 = {
        label = 'minute',
        ucode = 'min',
    },
    Q794261 = {
        label = 'cubic metre per second',
        ucode = 'm3/s',
    },
    Q809678 = {
        label = 'barye',
        ucode = 'Ba',
    },
    Q81292 = {
        label = 'acre',
        ucode = 'acre',
    },
    Q81454 = {
        label = 'ångström',
        ucode = 'angstrom',
    },
    Q828224 = {
        label = 'kilometre',
        ucode = 'km',
    },
    Q83327 = {
```

```
        label = 'electronvolt',
        ucode = 'eV',
    },
    Q838801 = {
        label = 'nanosecond',
        ucode = 'ns',
    },
    Q842015 = {
        label = 'microsecond',
        ucode = 'us',
    },
    Q844211 = {
        label = 'kilogram per cubic metre',
        ucode = 'kg/m3',
    },
    Q844338 = {
        label = 'hectometre',
        ucode = 'hm',
    },
    Q844976 = {
        label = 'oersted',
        ucode = 'Oe',
    },
    Q848856 = {
        label = 'decametre',
        ucode = 'dam',
    },
    Q854546 = {
        label = 'gigametre',
        ucode = 'Gm',
    },
    Q857027 = {
        label = 'square foot',
        ucode = 'sqft',
    },
    Q9048643 = {
        label = 'nanolitre',
        ucode = 'nl',
    },
    Q93318 = {
        label = 'nautical mile',
        ucode = 'nmi',
    },
},
}

return { wikidata_units = wikidata_units }
```



Modul:Convert/wikidata/data/sandbox

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

```
--[[ Cache of Wikidata information with units for Module:Convert.
The codes should rarely change, and using a cache means that changing a
unit at Wikidata will not cause lots of converts in articles to break.

For a unit known to convert, the unit here must have:
    label = Wikidata label for unit (used only when listing units)
    ucode = unit code for input to convert
    (there are no optional fields because convert handles everything)

For a unit not known to convert, the unit here must have:
    label = Wikidata label for unit (used only when listing units)
    (no ucode field)
    _ucode = unit code for input to convert, and the
             symbol used to display the unit when abbr=on
    (convert will use the specified fields to display the unit,
    and will not attempt to do a conversion)

For a unit not known to convert, the unit here may have:
    name1 = singular name used to display the unit when abbr=off
    name2 = plural name used to display the unit when abbr=off
    link = name of article that unit will be linked to when lk=on
    si = key for the SI base unit, if any

The base unit for each SI unit here must have:
    symbol = symbol used to display the base unit when abbr=on
    name1 = singular name of base unit used to display the unit when abbr=off
    (if name1 is not given, symbol will be used, but an SI unit should have a

The base unit for each SI unit here may have:
    name2 = plural name of base unit used to display the unit when abbr=off
    link = name of article that unit will be linked to when lk=on
           (applies for all SI units using this base, where the
           SI unit does not define its own link field)

If name1 is not specified, the symbol is used for the name.
If name2 is not specified, a plural name is formed by appending 's' to name1.
If link is not specified, name1 is used for the link.

SI units are assumed to be simple items like V (volt) where 'mV' would
cause convert to insert:
    'm' before the base symbol 'V' to make 'mV', if abbr=on
    'milli' before the base name 'volt' to make 'millivolt', if abbr=off
A unit like "square meter" would not work because it needs an SI prefix
inserted before "meter" rather than at the beginning of the name.

Items that should not be used with convert as no precise unit is implied:
Q11247037  ton          generic (cannot use)
Q178413   gallon       generic
Q130964   calorie      dubious (ambiguous, should not use)
Q216658   bushel       dubious
Q420266   fluid ounce  dubious
]]

local wikidata_units = {
```

```
-- Following are SI base units.
A = {
    symbol = 'A',
    name1 = 'ampere',
},
F = {
    symbol = 'F',
    name1 = 'faraday',
},
H = {
    symbol = 'H',
    name1 = 'henry',
},
V = {
    symbol = 'V',
    name1 = 'volt',
},
-- Following are aliases to convert unit codes, used with "input=<value>
kilograms = {
    ucode = 'kg',
},
-- Following are SI units not known to convert, used with "input=<value>
kV = {
    ucode = 'kV',
    si = 'V',
},
mV = {
    ucode = 'mV',
    si = 'V',
},
-- Following are Wikidata units.
Q131255 = {
    label = 'farad',
    _ucode = 'F',
    _si = 'F',
},
Q163354 = {
    label = 'henry',
    _ucode = 'H',
    _si = 'H',
},
Q1916026 = {
    label = 'microvolt',
    _ucode = 'uV',
    _si = 'V',
},
Q193933 = {
    label = 'dioptre',
    name1 = 'dioptre',
    _ucode = 'dpt',
},
Q212120 = {
    label = 'ampere hour',
    name1 = 'ampere hour',
    _ucode = 'A·h',
},
Q2448803 = {
    label = 'millivolt',
    _ucode = 'mV',
    _si = 'V',
},
Q2451296 = {
    label = 'microfarad',
    _ucode = 'uF',
```

```
        si = 'F',
    },
    Q2490574 = {
        label = 'milliampere',
        _ucode = 'mA',
        si = 'A',
    },
    Q25250 = {
        label = 'volt',
        _ucode = 'V',
        si = 'V',
    },
    Q25272 = {
        label = 'ampere',
        _ucode = 'A',
        si = 'A',
    },
    Q2553708 = {
        label = 'megavolt',
        _ucode = 'MV',
        si = 'V',
    },
    Q2554092 = {
        label = 'kilovolt',
        _ucode = 'kV',
        si = 'V',
    },
    Q2636421 = {
        label = 'nanohenry',
        _ucode = 'nH',
        si = 'H',
    },
    Q2679083 = {
        label = 'microhenry',
        _ucode = 'uH',
        si = 'H',
    },
    Q2682463 = {
        label = 'nanofarad',
        _ucode = 'nF',
        si = 'F',
    },
    Q2756030 = {
        label = 'picofarad',
        _ucode = 'pF',
        si = 'F',
    },
    Q2793566 = {
        label = 'gigavolt',
        _ucode = 'GV',
        si = 'V',
    },
    Q2924137 = {
        label = 'millihenry',
        _ucode = 'mH',
        si = 'H',
    },
    Q3117809 = {
        label = 'microampere',
        _ucode = 'uA',
        si = 'A',
    },
    Q33680 = {
        label = 'radian',
```

```
        name1 = 'radian',
        _ucode = 'rad',
    },
    Q4456994 = {
        label = 'millifarad',
        _ucode = 'mF',
        _si = 'F',
    },
    Q47083 = {
        label = 'ohm',
        name1 = 'ohm',
        _ucode = 'Ω',
    },
    Q483261 = {
        label = 'dalton',
        name1 = 'dalton',
        _ucode = 'u',
    },
    Q550341 = {
        label = 'volt-ampere',
        name1 = 'volt-ampere',
        _ucode = 'VA',
    },
    Q100995 = {
        label = 'pound',
        ucode = 'lb',
    },
    Q1022113 = {
        label = 'cubic centimetre',
        ucode = 'cc',
    },
    Q102573 = {
        label = 'becquerel',
        ucode = 'Bq',
    },
    Q103246 = {
        label = 'sievert',
        ucode = 'Sv',
    },
    Q1050958 = {
        label = 'inch of mercury',
        ucode = 'inHg',
    },
    Q1051665 = {
        label = 'metre per second squared',
        ucode = 'm/s2',
    },
    Q1052397 = {
        label = 'rad',
        ucode = 'rad',
    },
    Q1054140 = {
        label = 'megametre',
        ucode = 'Mm',
    },
    Q1057069 = {
        label = 'hectogram',
        ucode = 'hg',
    },
    Q1063786 = {
        label = 'square inch',
        ucode = 'sqin',
    },
    Q1092296 = {
```

```
        label = 'annum',
        ucode = 'year',
    },
    Q11570 = {
        label = 'kilogram',
        ucode = 'kg',
    },
    Q11573 = {
        label = 'metre',
        ucode = 'm',
    },
    Q11574 = {
        label = 'second',
        ucode = 's',
    },
    Q11579 = {
        label = 'kelvin',
        ucode = 'K',
    },
    Q11582 = {
        label = 'litre',
        ucode = 'litre',
    },
    Q1165588 = {
        label = 'rod',
        ucode = 'rod',
    },
    Q1165799 = {
        label = 'thou',
        ucode = 'thou',
    },
    Q11776930 = {
        label = 'megagram',
        ucode = 'Mg',
    },
    Q11929860 = {
        label = 'kiloparsec',
        ucode = 'kpc',
    },
    Q1194225 = {
        label = 'pound-force',
        ucode = 'lbf',
    },
    Q12129 = {
        label = 'parsec',
        ucode = 'pc',
    },
    Q12438 = {
        label = 'newton',
        ucode = 'N',
    },
    Q1255620 = {
        label = 'dram',
        ucode = 'drachm',
    },
    Q12874593 = {
        label = 'watt-hour',
        ucode = 'W.h',
    },
    Q128822 = {
        label = 'knot',
        ucode = 'kn',
    },
    Q1374438 = {
```

```
        label = 'kilosecond',
        ucode = 'ks',
    },
    Q1377051 = {
        label = 'gigasecond',
        ucode = 'Gs',
    },
    Q14754979 = {
        label = 'zettagram',
        ucode = 'Zg',
    },
    Q14786969 = {
        label = 'megajoule',
        ucode = 'MJ',
    },
    Q14787261 = {
        label = 'megawatt hour',
        ucode = 'MW.h',
    },
    Q1550511 = {
        label = 'square yard',
        ucode = 'sqyd',
    },
    Q160857 = {
        label = 'metric horsepower',
        ucode = 'hp',
    },
    Q1628990 = {
        label = 'horsepower-hour',
        ucode = 'hph',
    },
    Q163343 = {
        label = 'tesla',
        ucode = 'T',
    },
    Q1645498 = {
        label = 'microgram',
        ucode = 'ug',
    },
    Q17087835 = {
        label = 'cuerda',
        ucode = 'cda',
    },
    Q174728 = {
        label = 'centimetre',
        ucode = 'cm',
    },
    Q174789 = {
        label = 'millimetre',
        ucode = 'mm',
    },
    Q175821 = {
        label = 'micrometre',
        ucode = 'um',
    },
    Q1770733 = {
        label = 'teragram',
        ucode = 'Tg',
    },
    Q1772386 = {
        label = 'decigram',
        ucode = 'dg',
    },
    Q177493 = {
```

```
        label = 'gauss',
        ucode = 'G',
    },
    Q1777507 = {
        label = 'femtosecond',
        ucode = 'fs',
    },
    Q177974 = {
        label = 'standard atmosphere',
        ucode = 'atm',
    },
    Q178674 = {
        label = 'nanometre',
        ucode = 'nm',
    },
    Q180154 = {
        label = 'kilometre per hour',
        ucode = 'km/h',
    },
    Q180892 = {
        label = 'solar mass',
        ucode = 'solar mass',
    },
    Q1811 = {
        label = 'astronomical unit',
        ucode = 'au',
    },
    Q1815100 = {
        label = 'centilitre',
        ucode = 'cl',
    },
    Q182098 = {
        label = 'kilowatt hour',
        ucode = 'kW.h',
    },
    Q1823150 = {
        label = 'microwatt',
        ucode = 'uW',
    },
    Q182429 = {
        label = 'metre per second',
        ucode = 'm/s',
    },
    Q1826195 = {
        label = 'decilitre',
        ucode = 'dl',
    },
    Q185078 = {
        label = 'are',
        ucode = 'a',
    },
    Q185153 = {
        label = 'erg',
        ucode = 'erg',
    },
    Q185648 = {
        label = 'torr',
        ucode = 'Torr',
    },
    Q190095 = {
        label = 'gray',
        ucode = 'Gy',
    },
    Q191118 = {
```

```
        label = 'tonne',
        ucode = 'tonne',
    },
    Q1913097 = {
        label = 'femtogram',
        ucode = 'fg',
    },
    Q192274 = {
        label = 'picometre',
        ucode = 'pm',
    },
    Q1972579 = {
        label = 'poundal',
        ucode = 'pdl',
    },
    Q200323 = {
        label = 'decimetre',
        ucode = 'dm',
    },
    Q201933 = {
        label = 'dyne',
        ucode = 'dyn',
    },
    Q2029519 = {
        label = 'hectolitre',
        ucode = 'hl',
    },
    Q2051195 = {
        label = 'gigawatt hour',
        ucode = 'GW.h',
    },
    Q207488 = {
        label = 'Rankine scale',
        ucode = 'R',
    },
    Q208788 = {
        label = 'femtometre',
        ucode = 'fm',
    },
    Q2101 = {
        label = 'elementary charge',
        ucode = 'e',
    },
    Q21014455 = {
        label = 'metre per minute',
        ucode = 'm/min',
    },
    Q21062777 = {
        label = 'megapascal',
        ucode = 'MPa',
    },
    Q21064807 = {
        label = 'kilopascal',
        ucode = 'kPa',
    },
    Q211256 = {
        label = 'mile per hour',
        ucode = 'mph',
    },
    Q21178489 = {
        label = 'barrels per day',
        ucode = 'oilbbl/d',
    },
    Q2143992 = {
```

```
        label = 'kilohertz',
        ucode = 'kHz',
    },
    Q21467992 = {
        label = 'cubic foot per second',
        ucode = 'cuft/s',
    },
    Q215571 = {
        label = 'newton metre',
        ucode = 'Nm',
    },
    Q216795 = {
        label = 'dunam',
        ucode = 'dunam',
    },
    Q216880 = {
        label = 'kilogram-force',
        ucode = 'kgf',
    },
    Q18413919 = {
        label = 'centimetre per second',
        ucode = 'cm/s',
    },
    Q218593 = {
        label = 'inch',
        ucode = 'in',
    },
    Q2282891 = {
        label = 'microlitre',
        ucode = 'ul',
    },
    Q2282906 = {
        label = 'nanogram',
        ucode = 'ng',
    },
    Q229354 = {
        label = 'curie',
        ucode = 'Ci',
    },
    Q232291 = {
        label = 'square mile',
        ucode = 'sqmi',
    },
    Q2332346 = {
        label = 'millilitre',
        ucode = 'ml',
    },
    Q23387 = {
        label = 'week',
        ucode = 'week',
    },
    Q23823681 = {
        label = 'terawatt',
        ucode = 'TW',
    },
    Q23925410 = {
        label = 'gallon (UK)',
        ucode = 'impgal',
    },
    Q23925413 = {
        label = 'gallon (US)',
        ucode = 'USgal',
    },
    Q2438073 = {
```

```
        label = 'attogram',
        ucode = 'ag',
    },
    Q2474258 = {
        label = 'millisievert',
        ucode = 'mSv',
    },
    Q2483628 = {
        label = 'attosecond',
        ucode = 'as',
    },
    Q2489298 = {
        label = 'square centimetre',
        ucode = 'cm2',
    },
    Q2518569 = {
        label = 'nanosievert',
        ucode = 'nSv',
    },
    Q25235 = {
        label = 'hour',
        ucode = 'h',
    },
    Q25236 = {
        label = 'watt',
        ucode = 'W',
    },
    Q25267 = {
        label = 'degree Celsius',
        ucode = 'C',
    },
    Q25269 = {
        label = 'joule',
        ucode = 'J',
    },
    Q253276 = {
        label = 'mile',
        ucode = 'mi',
    },
    Q25343 = {
        label = 'square metre',
        ucode = 'm2',
    },
    Q25406 = {
        label = 'coulomb',
        ucode = 'coulomb',
    },
    Q25517 = {
        label = 'cubic metre',
        ucode = 'm3',
    },
    Q260126 = {
        label = 'Roentgen equivalent man',
        ucode = 'rem',
    },
    Q2612219 = {
        label = 'petagram',
        ucode = 'Pg',
    },
    Q2619500 = {
        label = 'foe',
        ucode = 'foe',
    },
    Q2637946 = {
```

```
        label = 'decalitre',
        ucode = 'dal',
    },
    Q2655272 = {
        label = 'exagram',
        ucode = 'Eg',
    },
    Q2691798 = {
        label = 'centigram',
        ucode = 'cg',
    },
    Q2739114 = {
        label = 'microsievert',
        ucode = 'uSv',
    },
    Q2799294 = {
        label = 'gigagram',
        ucode = 'Gg',
    },
    Q3013059 = {
        label = 'kiloannum',
        ucode = 'millennium',
    },
    Q305896 = {
        label = 'dots per inch',
        ucode = 'dpi',
    },
    Q3207456 = {
        label = 'milliwatt',
        ucode = 'mW',
    },
    Q3221356 = {
        label = 'yoctometre',
        ucode = 'ym',
    },
    Q3239557 = {
        label = 'picogram',
        ucode = 'pg',
    },
    Q3241121 = {
        label = 'milligram',
        ucode = 'mg',
    },
    Q3267417 = {
        label = 'terametre',
        ucode = 'Tm',
    },
    Q3270676 = {
        label = 'zeptometre',
        ucode = 'zm',
    },
    Q3276763 = {
        label = 'gigahertz',
        ucode = 'GHz',
    },
    Q3277907 = {
        label = 'exametre',
        ucode = 'Em',
    },
    Q3277915 = {
        label = 'zettametre',
        ucode = 'Zm',
    },
    Q3277919 = {
```

```
        label = 'petametre',
        ucode = 'Pm',
    },
    Q3312063 = {
        label = 'femtolitre',
        ucode = 'fl',
    },
    Q3320608 = {
        label = 'kilowatt',
        ucode = 'kW',
    },
    Q3332822 = {
        label = 'megaton of TNT',
        ucode = 'Mt(TNT)',
    },
    Q35852 = {
        label = 'hectare',
        ucode = 'ha',
    },
    Q3675550 = {
        label = 'cubic millimetre',
        ucode = 'mm3',
    },
    Q3710 = {
        label = 'foot',
        ucode = 'ft',
    },
    Q3773454 = {
        label = 'megaparsec',
        ucode = 'Mpc',
    },
    Q3902688 = {
        label = 'picolitre',
        ucode = 'pl',
    },
    Q3902709 = {
        label = 'picosecond',
        ucode = 'ps',
    },
    Q39369 = {
        label = 'hertz',
        ucode = 'Hz',
    },
    Q3972226 = {
        label = 'kilolitre',
        ucode = 'kl',
    },
    Q4068266 = {
        label = "apothecaries' drachm",
        ucode = 'drachm',
    },
    Q41803 = {
        label = 'gram',
        ucode = 'g',
    },
    Q4220561 = {
        label = 'kilometre per second',
        ucode = 'km/s',
    },
    Q42289 = {
        label = 'degree Fahrenheit',
        ucode = 'F',
    },
    Q4243638 = {
```

```
        label = 'cubic kilometre',
        ucode = 'km3',
    },
    Q44395 = {
        label = 'pascal',
        ucode = 'Pa',
    },
    Q48013 = {
        label = 'ounce',
        ucode = 'oz',
    },
    Q482798 = {
        label = 'yard',
        ucode = 'yd',
    },
    Q4989854 = {
        label = 'kilojoule',
        ucode = 'kJ',
    },
    Q4992853 = {
        label = 'kiloton of TNT',
        ucode = 'kt(TNT)',
    },
    Q5139563 = {
        label = 'hectopascal',
        ucode = 'hPa',
    },
    Q5151 = {
        label = 'month',
        ucode = 'month',
    },
    Q531 = {
        label = 'light-year',
        ucode = 'ly',
    },
    Q5465723 = {
        label = 'foot-poundal',
        ucode = 'ftpdℓ',
    },
    Q573 = {
        label = 'day',
        ucode = 'd',
    },
    Q577 = {
        label = 'year',
        ucode = 'year',
    },
    Q5879479 = {
        label = 'gigawatt',
        ucode = 'GW',
    },
    Q6003257 = {
        label = 'attometre',
        ucode = 'am',
    },
    Q613726 = {
        label = 'yottagram',
        ucode = 'Yg',
    },
    Q6170164 = {
        label = 'yoctogram',
        ucode = 'yg',
    },
    Q667419 = {
```

```
        label = 'long ton',
        ucode = 'LT',
    },
    Q673166 = {
        label = 'gravity of Earth',
        ucode = 'g0',
    },
    Q693944 = {
        label = 'grain',
        ucode = 'gr',
    },
    Q6982035 = {
        label = 'megawatt',
        ucode = 'MW',
    },
    Q712226 = {
        label = 'square kilometre',
        ucode = 'km2',
    },
    Q723733 = {
        label = 'millisecond',
        ucode = 'ms',
    },
    Q732454 = {
        label = 'megaannum',
        ucode = 'Myr',
    },
    Q732707 = {
        label = 'megahertz',
        ucode = 'MHz',
    },
    Q752079 = {
        label = 'gross register ton',
        ucode = 'grt',
    },
    Q752197 = {
        label = 'kilojoule per mole',
        ucode = 'kJ/mol',
    },
    Q7727 = {
        label = 'minute',
        ucode = 'min',
    },
    Q794261 = {
        label = 'cubic metre per second',
        ucode = 'm3/s',
    },
    Q809678 = {
        label = 'barye',
        ucode = 'Ba',
    },
    Q81292 = {
        label = 'acre',
        ucode = 'acre',
    },
    Q81454 = {
        label = 'ångström',
        ucode = 'angstrom',
    },
    Q828224 = {
        label = 'kilometre',
        ucode = 'km',
    },
    Q83327 = {
```

```
        label = 'electronvolt',
        ucode = 'eV',
    },
    Q838801 = {
        label = 'nanosecond',
        ucode = 'ns',
    },
    Q842015 = {
        label = 'microsecond',
        ucode = 'us',
    },
    Q844211 = {
        label = 'kilogram per cubic metre',
        ucode = 'kg/m3',
    },
    Q844338 = {
        label = 'hectometre',
        ucode = 'hm',
    },
    Q844976 = {
        label = 'oersted',
        ucode = 'Oe',
    },
    Q848856 = {
        label = 'decametre',
        ucode = 'dam',
    },
    Q854546 = {
        label = 'gigametre',
        ucode = 'Gm',
    },
    Q857027 = {
        label = 'square foot',
        ucode = 'sqft',
    },
    Q9048643 = {
        label = 'nanolitre',
        ucode = 'nl',
    },
    Q93318 = {
        label = 'nautical mile',
        ucode = 'nmi',
    },
},
}

return { wikidata_units = wikidata_units }
```

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)) then
            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'
        end
    end
end
```

```
                unit = 'kHz'
            else
                unit = 'Hz'
            end
        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 1
        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 first
    -- Then, if convert is showing all values for a property such as the diameter
    -- of a telescope's mirror (diameters of primary and secondary mirrors),
    -- will not show alternative values that could in principle be present for
    -- same item (telescope) and property (diameter) and qualifier (primary/secondary)
    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 statement
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 = statement
-- Statements are in Wikidata's order except that those with preferred rank
-- are first, then normal rank. Any other rank is ignored.
local stored = {} -- qualifiers of statements that are first for the quality
local qid = strip_to_nil(params.qid) -- nil for current page's item, or a
local qual = strip_to_nil(params.qual) -- nil or id of wanted P518 (applies)
local result = Collection.new()
local entity = mw.wikibase.getEntity(qid)
if type(entity) == 'table' then
    local statements = (entity.claims or {})[qid]
    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 = match
                    if is_match then
                        result:add({ statqual = s
                    end
                end
            end
        end
    end
end
return result
end

local function input_from_property(tdata, params, 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 c
-- even if no qual=x is specified. Unclear what should happen.
-- For the wavelength at [[Nançay Radio Telescope]], want to show all th
-- values, and the values have no qualifiers.
-----
local result = Collection.new()
local done = {}
local skip_normal
for _, t in ipairs(get_statements(params, 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 am
                if amount:sub(1, 1) == '+' then
                    amount = amount:sub(2)
                end
            end
        end
    end
end
return result
end
```



```
    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 occur
        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 !!l
        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 !!name2 !!link !!label',
  item   = '|-\n|%s ||%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;') --
end
return unpack(result)
end
local wdunits = tdata.wikidata_units
local speckey = { 'base', 'alias', 'unknown', 'known' }
for _, sid in ipairs(speckey) do
  specifications[sid].units = Collection.new()
end
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.unicode then
            sid = 'known'
        else
            sid = 'unknown'
        end
    elseif unit.unicode 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 ''
end
```

```
        spec.units:add(result)
    end
    local results = Collection.new()
    if note_count > 0 then
        local text = note_count .. (note_count == 1 and ' note' or ' notes')
        results:add("''Search for * to see " .. text .. ""'\n")
    end
    for _, sid in ipairs(speckey) 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 }
```