

# Inhaltsverzeichnis

- [1. Modul:Convert/text/sandbox/Doku](#)
- [2. Modul:Convert](#)
- [3. Modul:Convert/text/sandbox](#)

## Modul:Convert/text/sandbox/Doku

Dies ist die Dokumentationsseite für [Modul:Convert/text/sandbox](#)

This page defines text used by [Module:Convert](#). All documentation (from [Module:Convert/doc](#)) is at that module. The text includes messages and categories output by the module, and parameters used as input.

This is a separate module to simplify translation for use on another wiki. For example, see **translation\_table** and the other tables in [bn:Module:Convert/text](#). Documentation is at [Template:Convert/Transwiki guide](#).

Any changes should first be tested at [Module:Convert/text/sandbox](#)—see [Template:Convert/testcases#Sandbox testcases](#).

## Modul:Convert

[Vorlage:Transwiki guide](#) [Vorlage:Lua](#) [Vorlage:Uses TemplateStyles](#) This module converts a value from one unit of measurement to another. For example:

- `{{convert|123|lb|kg}}` → 123 pounds (56 kg)

The module is called using a template—parameters passed to the template are used by this module to control how a conversion is performed. For example, units can be abbreviated (like **kg**), or displayed as names (like **kilogram**), and the output value can be rounded to a specified precision. For usage information, see [Help:Convert](#).

## Inhaltsverzeichnis

- [1 Templates and modules](#)
- [2 Sandbox](#)
- [3 Configuration](#)
- [4 To do](#)
- [5 Module version history](#)

## Templates and modules

Templates that invoke this module are:

- [Template:Convert](#)
- [Template:Cvt](#) [Vorlage:Green](#)

The following modules are required:

- [Module:Convert](#) - (*this module*) code to convert units
- [Module:Convert/data](#) - unit definitions
- [Module:Convert/text](#) - text messages, and parameter names and values

The following modules are optional and are used only if required and if the module exists:

- [Module:Convert/extra](#) - extra (temporary) unit definitions; used if a unit is not found in [Module:Convert/data](#)
- [Module:ConvertNumeric](#) - code to spell an input value in words (only English is supported; however, see [vi:Module:ConvertNumeric](#))

For Wikidata support the following modules are required:

- [Module:Convert/wikidata](#)
- [Module:Convert/wikidata/data](#)

The following help pages are available:

- [Help:Convert](#) - overview
- [Help:Convert messages](#) - describes error and warning messages; messages link to this page so it is required when the module is copied to another wiki
- [Help:Convert units](#) - overview of units

A page containing a convert error is added to the following hidden category, providing the page is in a specified [namespace](#) (articles, by default):

- [Vorlage:Clc](#)

Units are defined in the wikitext of the master list of units.

- [Module:Convert/documentation/conversion data](#) - master list of unit definitions
- [Module:Convert/makeunits](#) - translates wikitext from the master list to Lua
- [Module talk:Convert/makeunits](#) - makeunits results; copy the text to [Module:Convert/data](#)

[Module:Convert/data](#) is transcluded into every page using the convert module, so experimenting with a new unit in that module would involve a significant overhead. The [Module:Convert/extra](#) module is an alternative which is only transcluded on pages with a unit that is not defined in the main data module.

[Module talk:Convert/show](#) lists all unit links so they can be checked.

## Sandbox

When making a change, copy the current modules to the sandbox pages, then edit the sandbox copies:

- [Module:Convert](#) • [Module:Convert/sandbox](#) • same content
- [Module:Convert/data](#) • [Module:Convert/data/sandbox](#) • same content

- [Module:Convert/text](#) • [Module:Convert/text/sandbox](#) • same content
- [Module:Convert/extra](#) • [Module:Convert/extra/sandbox](#) • same content
- [Module:Convert/wikidata](#) • [Module:Convert/wikidata/sandbox](#) • same content
- [Module:Convert/wikidata/data](#) • [Module:Convert/wikidata/data/sandbox](#) • same content

Use the following template to test the results (example `{{convert/sandbox|123|lb|kg}}`):

- [Template:Convert/sandbox](#)

[Template:Convert/sandbox](#) invokes [Module:Convert/sandbox](#) with parameter [Vorlage:Para](#) which causes convert to use the sandbox modules rather than the normal modules.

The following should be used to test the results of editing the convert modules.

- [Template:Convert/testcases#Sandbox testcases](#) - links to testcases
- [Module:Convert/tester](#) - module to run tests by comparing template output with fixed text

It is not necessary to save a testcases page before viewing test results. For example, [Template:Convert/testcases/sandbox4](#) could be edited to change the tests. While still editing that page, paste [Vorlage:Nowrap](#) (without quotes) into the page title box under "Preview page with this template", then click "Show preview".

## Configuration

The template that invokes this module can define options to configure the module. For example:

- `{{#invoke:convert|convert|numdot=,|numsep=.}}`

Sets the [decimal mark](#) to be a comma, and the thousands separator to be a dot.

Other options, with default values, are:

- `|maxsigfig=14` - maximum number of significant figures
- `|nscat=0` - [namespaces](#) (comma separated) in which an error or warning adds a category to the page
- `|warnings=0` - 0 (zero) disables warnings; 1 shows important warnings; 2 shows all warnings

An option in the template can specify that the sandbox versions of the modules be used. If specified, the text on the right-hand side of the equals sign must be the name of the subpage for each sandbox module.

- `|sandbox=sandbox` - omit for normal operation

All text used for input parameters and for output messages and categories can be customized. For example, at enwiki the option `|lk=on` can be used to link each displayed unit to its article. The "lk" and "on" can be replaced with any desired text. In addition, input and output numbers can be formatted and can use digits in the local language. See the [translation guide](#) for more information.

## To do

Document the modules to access Wikidata!

## Module version history

- [Version 1](#) December 2013
- [Version 2](#) January 2014
- [Version 3](#) April 2014
- [Version 4](#) July 2014
- [Version 5](#) September 2014
- [Version 6](#) November 2014
- [Version 7](#) December 2014
- [Version 8](#) February 2015
- [Version 9](#) February 2015
- [Version 10](#) May 2015
- [Version 11](#) June 2015
- [Version 12](#) August 2015
- [Version 13](#) March 2016
- [Version 14](#) June 2016 [Vorlage:Green](#)
- [Version 15](#) September 2016
- [Version 16](#) January 2017
- [Version 17](#) May 2017
- [Version 18](#) July 2017
- [Version 19](#) August 2017
- [Version 20](#) December 2017 [Vorlage:Green](#)
- [Version 21](#) January 2018 [Vorlage:Green](#)
- [Version 22](#) February 2018 [Vorlage:Green](#)
- [Version 23](#) June 2018 [Vorlage:Green](#)
- [Version 24](#) May 2019 [Vorlage:Green](#)
- [Version 25](#) May 2021 [Vorlage:Green](#)
- [Version 26](#) June 2021 [Vorlage:Green](#)
- [Version 27](#) February 2022 [Vorlage:Green](#)

```
-- Convert a value from one unit of measurement to another.
-- Example: {{convert|123|lb|kg}} --> 123 pounds (56 kg)
-- See [[:en:Template:Convert/Transwiki guide]] if copying to another wiki.
```

```
local MINUS = '-' -- Unicode U+2212 MINUS SIGN (UTF-8: e2 88 92)
local abs = math.abs
local floor = math.floor
local format = string.format
local log10 = math.log10
local ustring = mw.ustring
local ulen = ustring.len
local usub = ustring.sub
```

```
-- Configuration options to keep magic values in one location.
-- Conversion data and message text are defined in separate modules.
local config, maxsigfig
local numdot -- must be '.' or ',' or a character which works in a regex
local numsep, numsep_remove, numsep_remove2
local data_code, all_units
```

```

local text_code
local varname      -- can be a code to use variable names that depend on value
local from_en_table -- to translate an output string of en digits to local language
local to_en_table  -- to translate an input string of digits in local language to en
-- Use translation_table in convert/text to change the following.
local en_default   -- true uses lang=en unless convert has lang=local or local digits
local group_method = 3 -- code for how many digits are in a group
local per_word = 'per' -- for units like "liters per kilometer"
local plural_suffix = 's' -- only other useful value is probably '' to disable plural unit names
local omitsep      -- true to omit separator before local symbol/name

-- All units should be defined in the data module. However, to cater for quick changes
-- and experiments, any unknown unit is looked up in an extra data module, if it exists.
-- That module would be transcluded in only a small number of pages, so there should be
-- little server overhead from making changes, and changes should propagate quickly.
local extra_module -- name of module with extra units
local extra_units  -- nil or table of extra units from extra_module

-- Some options in the invoking template can set variables used later in the module.
local currency_text -- for a user-defined currency symbol: {{convert|12|$/ha| $\$=\text{€}$ }} (euro replaces

local function from_en(text)
  -- Input is a string representing a number in en digits with '.' decimal mark,
  -- without digit grouping (which is done just after calling this).
  -- Return the translation of the string with numdot and digits in local language.
  if numdot ~= '.' then
    text = text:gsub('%.', numdot)
  end
  if from_en_table then
    text = text:gsub('%d', from_en_table)
  end
  return text
end

local function to_en(text)
  -- Input is a string representing a number in the local language with
  -- an optional numdot decimal mark and numsep digit grouping.
  -- Return the translation of the string with '.' mark and en digits,
  -- and no separators (they have to be removed here to handle cases like
  -- numsep = '.' and numdot = ',' with input "1.234.567,8").
  if to_en_table then
    text = ustring.gsub(text, '%d', to_en_table)
  end
  if numsep_remove then
    text = text:gsub(numsep_remove, '')
  end
  if numsep_remove2 then
    text = text:gsub(numsep_remove2, '')
  end
  end
  if numdot ~= '.' then
    text = text:gsub(numdot, '.')
  end

```



```

-- The following kludge should be a sufficient test.
if omitsep then
    if id:sub(1, 2) == '-{' then -- for "-{...}-" content language variant
        return true
    end
    if id:byte() > 127 then
        local first = usub(id, 1, 1)
        if first ~= 'À' and first ~= '°' and first ~= 'µ' then
            return true
        end
    end
end
return id:sub(1, 1) == '/' -- no separator before units like "/ha"
end

local spell_module -- name of module that can spell numbers
local speller      -- function from that module to handle spelling (set if needed)
local wikidata_module, wikidata_data_module -- names of Wikidata modules
local wikidata_code, wikidata_data -- exported tables from those modules (set if needed)

local function set_config(args)
    -- Set configuration options from template #invoke or defaults.
    config = args
    maxsigfig = config.maxsigfig or 14 -- maximum number of significant figures
    local data_module, text_module
    local sandbox = config.sandbox and ('/' .. config.sandbox) or ''
    data_module = "Module:Convert/data" .. sandbox
    text_module = "Module:Convert/text" .. sandbox
    extra_module = "Module:Convert/extra" .. sandbox
    wikidata_module = "Module:Convert/wikidata" .. sandbox
    wikidata_data_module = "Module:Convert/wikidata/data" .. sandbox
    spell_module = "Module:ConvertNumeric"
    data_code = mw.loadData(data_module)
    text_code = mw.loadData(text_module)
    all_units = data_code.all_units
    local translation = text_code.translation_table
    if translation then
        numdot = translation.numdot
        numsep = translation.numsep
        if numdot == ',' and numsep == '.' then
            if text_code.all_messages.cvt_enwiki_num then
                to_en = to_en_with_check
            end
        end
    end
    if translation.group then
        group_method = translation.group
    end
    if translation.per_word then
        per_word = translation.per_word
    end
    if translation.plural_suffix then

```

```

        plural_suffix = translation.plural_suffix
    end
    varname = translation.varname
    from_en_table = translation.from_en
    local use_workaround = true
    if use_workaround then
        -- 2013-07-05 workaround bug by making a copy of the required table.
        -- mw.uststring.gsub fails with a table (to_en_table) as the replacement,
        -- if the table is accessed via mw.loadData.
        local source = translation.to_en
        if source then
            to_en_table = {}
            for k, v in pairs(source) do
                to_en_table[k] = v
            end
        end
    else
        to_en_table = translation.to_en
    end
    if translation.lang == 'en default' then
        en_default = true -- for hiwiki
    end
    omitsep = translation.omitsep -- for zhwiki
end
numdot = config.numdot or numdot or '.' -- decimal mark before fractional digits
numsep = config.numsep or numsep or ',' -- group separator for numbers
-- numsep should be ',' or '.' or '' or '&nbsp;' or a Unicode character.
-- numsep_remove must work in a regex to identify separators to be removed.
if numsep ~= '' then
    numsep_remove = (numsep == '.') and '%.' or numsep
end
if numsep ~= ',' and numdot ~= '.' then
    numsep_remove2 = ',' -- so numbers copied from enwiki will work
end
end

local function collection()
    -- Return a table to hold items.
    return {
        n = 0,
        add = function (self, item)
            self.n = self.n + 1
            self[self.n] = item
        end,
    }
end

local function divide(numerator, denominator)
    -- Return integers quotient, remainder resulting from dividing the two
    -- given numbers, which should be unsigned integers.
    local quotient, remainder = floor(numerator / denominator), numerator % denominator

```

```

    if not (0 <= remainder and remainder < denominator) then
        -- Floating point limits may need this, as in {{convert|160.02|Ym|ydftin}}.
        remainder = 0
    end
    return quotient, remainder
end

local function split(text, delimiter)
    -- Return a numbered table with fields from splitting text.
    -- The delimiter is used in a regex without escaping (for example, '.' would fail).
    -- Each field has any leading/trailing whitespace removed.
    local t = {}
    text = text .. delimiter -- to get last item
    for item in text:gmatch('%s*(.)%s*' .. delimiter) do
        table.insert(t, item)
    end
    return t
end

local function strip(text)
    -- If text is a string, return its content with no leading/trailing
    -- whitespace. Otherwise return nil (a nil argument gives a nil result).
    if type(text) == 'string' then
        return text:match("^%s*(.)%s*$")
    end
end

local function table_len(t)
    -- Return length (<100) of a numbered table to replace #t which is
    -- documented to not work if t is accessed via mw.loadData().
    for i = 1, 100 do
        if t[i] == nil then
            return i - 1
        end
    end
end

local function wanted_category(catkey, catsort, want_warning)
    -- Return message category if it is wanted in current namespace,
    -- otherwise return ''.
    local cat
    local title = mw.title.getCurrentTitle()
    if title then
        local nsdefault = '0' -- default namespace: '0' = article; '0,10' = article and 1
        local namespace = title.namespace
        for _, v in ipairs(split(config.nscat or nsdefault, ',')) do
            if namespace == tonumber(v) then
                cat = text_code.all_categories[want_warning and 'warning' or catk
                if catsort and catsort ~= '' and cat:sub(-2) == ']]' then
                    cat = cat:sub(1, -3) .. '|' .. mw.text.nowiki(usub(catsort
            end
        end
    end
end

```

```

                break
            end
        end
    end
end
return cat or ''
end

local function message(parms, mcode, is_warning)
    -- Return wikitext for an error message, including category if specified
    -- for the message type.
    -- mcode = numbered table specifying the message:
    --   mcode[1] = 'cvt_xxx' (string used as a key to get message info)
    --   mcode[2] = 'parm1' (string to replace '$1' if any in message)
    --   mcode[3] = 'parm2' (string to replace '$2' if any in message)
    --   mcode[4] = 'parm3' (string to replace '$3' if any in message)
    local msg
    if type(mcode) == 'table' then
        if mcode[1] == 'cvt_no_output' then
            -- Some errors should cause convert to output an empty string,
            -- for example, for an optional field in an infobox.
            return ''
        end
        msg = text_code.all_messages[mcode[1]]
    end
    parms.have_problem = true
    local function subparm(fmt, ...)
        local rep = {}
        for i, v in ipairs({...}) do
            rep['$' .. i] = v
        end
        return (fmt:gsub('%d+', rep))
    end
    if msg then
        local parts = {}
        local regex, replace = msg.regex, msg.replace
        for i = 1, 3 do
            local limit = 40
            local s = mcode[i + 1]
            if s then
                if regex and replace then
                    s = s:gsub(regex, replace)
                    limit = nil -- allow long "should be" messages
                end
                -- Escape user input so it does not break the message.
                -- To avoid tags (like {{convert|1<math>23</math>|m}}) breaking
                -- the mouseover title, any strip marker starting with char(127) is
                -- replaced with '...' (text not needing i18n).
                local append
                local pos = s:find(string.char(127), 1, true)
                if pos then
                    append = '...'
                end
            end
        end
    end
end

```

```

        s = s:sub(1, pos - 1)
    end
    if limit and ulen(s) > limit then
        s = usub(s, 1, limit)
        append = '...'
    end
    s = mw.text.nowiki(s) .. (append or '')
else
    s = '?'
end
parts['$' .. i] = s
end
local function ispreview()
    -- Return true if a prominent message should be shown.
    if parms.test == 'preview' or parms.test == 'nopreview' then
        -- For testing, can preview a real message or simulate a preview
        -- when running automated tests.
        return parms.test == 'preview'
    end
    local success, revid = pcall(function ()
        return (parms.frame):preprocess('{{REVISIONID}}') end)
    return success and (revid == '')
end
local want_warning = is_warning and
    not config.warnings and -- show unobtrusive warnings if config.warnings r
    not msg.nowarn         -- but use msg settings, not standard warning, if
local title = string.gsub(msg[1] or 'Missing message', '%$d+', parts)
local text = want_warning and '*' or msg[2] or 'Missing message'
local cat = wanted_category(msg[3], mcode[2], want_warning)
local anchor = msg[4] or ''
local fmtkey = ispreview() and 'cvt_format_preview' or
    (want_warning and 'cvt_format2' or msg.format or 'cvt_format')
local fmt = text_code.all_messages[fmtkey] or 'convert: bug'
return subparm(fmt, title:gsub('\"', '&quot;'), text, cat, anchor)
end
return 'Convert internal error: unknown message'
end

function add_warning(parms, level, key, text1, text2) -- for forward declaration above
    -- If enabled, add a warning that will be displayed after the convert result.
    -- A higher level is more verbose: more kinds of warnings are displayed.
    -- To reduce output noise, only the first warning is displayed.
    if level <= (tonumber(config.warnings) or 1) then
        if parms.warnings == nil then
            parms.warnings = message(parms, { key, text1, text2 }, true)
        end
    end
end

local function spell_number(parms, inout, number, numerator, denominator)
    -- Return result of spelling (number, numerator, denominator), or

```

```

-- return nil if spelling is not available or not supported for given text.
-- Examples (each value must be a string or nil):
--   number numerator denominator output
--   -----
--   "1.23"   nil          nil          one point two three
--   "1"      "2"         "3"         one and two thirds
--   nil      "2"         "3"         two thirds
if not speller then
    local function get_speller(module)
        return require(module).spell_number
    end
    local success
    success, speller = pcall(get_speller, spell_module)
    if not success or type(speller) ~= 'function' then
        add_warning(parms, 1, 'cvt_no_spell', 'spell')
    end
    return nil
end
end
local case
if parms.spell_upper == inout then
    case = true
    parms.spell_upper = nil -- only uppercase first word in a multiple unit
end
local sp = not parms.opt_sp_us
local adj = parms.opt_adjectival
return speller(number, numerator, denominator, case, sp, adj)
end

-----
-- BEGIN: Code required only for built-in units.
-- LATER: If need much more code, move to another module to simplify this module.
local function speed_of_sound(altitude)
    -- This is for the Mach built-in unit of speed.
    -- Return speed of sound in metres per second at given altitude in feet.
    -- If no altitude given, use default (zero altitude = sea level).
    -- Table gives speed of sound in miles per hour at various altitudes:
    --   altitude = -17,499 to 402,499 feet
    --   mach_table[a + 4] = s where
    --   a = (altitude / 5000) rounded to nearest integer (-3 to 80)
    --   s = speed of sound (mph) at that altitude
    -- LATER: Should calculate result from an interpolation between the next
    -- lower and higher altitudes in table, rather than rounding to nearest.
    -- From: http://www.aerospaceweb.org/question/atmosphere/q0112.shtml
    local mach_table = {
        799.5, 787.0, 774.2, 761.207051, -- a =
        748.0, 734.6, 721.0, 707.0, 692.8, 678.3, 663.5, 660.1, 660.1, 660.1, -- -3 to 10
        660.1, 660.1, 660.1, 662.0, 664.3, 666.5, 668.9, 671.1, 673.4, 675.6, -- 11 to 20
        677.9, 683.7, 689.9, 696.0, 702.1, 708.1, 714.0, 719.9, 725.8, 731.6, -- 21 to 30
        737.3, 737.7, 737.7, 736.2, 730.5, 724.6, 718.8, 712.9, 707.0, 701.0, -- 31 to 40
        695.0, 688.9, 682.8, 676.6, 670.4, 664.1, 657.8, 652.9, 648.3, 643.7, -- 41 to 50
        639.1, 634.4, 629.6, 624.8, 620.0, 615.2, 613.2, 613.2, 613.2, 613.5, -- 51 to 60
    }

```

```

        614.4, 615.3, 616.7, 619.8, 623.4, 629.7, 635.0, 641.1, 650.6, 660.0, -- 61 to 70
        672.5, 674.3, 676.1, 677.9, 679.7, 681.5, 683.3, 685.1, 686.8, 688.6, -- 71 to 80
    }
    altitude = altitude or 0
    local a = (altitude < 0) and -altitude or altitude
    a = floor(a / 5000 + 0.5)
    if altitude < 0 then
        a = -a
    end
    if a < -3 then
        a = -3
    elseif a > 80 then
        a = 80
    end
    return mach_table[a + 4] * 0.44704 -- mph converted to m/s
end
-- END: Code required only for built-in units.
-----

local function add_style(parms, class)
    -- Add selected template style to parms if not already present.
    parms.templatestyles = parms.templatestyles or {}
    if not parms.templatestyles[class] then
        parms.templatestyles[class] = parms.frame:extensionTag({
            name = 'templatestyles', args = { src = text_code.titles[class] }
        })
    end
end

local function get_styles(parms)
    -- Return string of required template styles, empty if none.
    if parms.templatestyles then
        local t = {}
        for _, v in pairs(parms.templatestyles) do
            table.insert(t, v)
        end
        return table.concat(t)
    end
    return ''
end

local function get_range(word)
    -- Return a range (string or table) corresponding to word (like "to"),
    -- or return nil if not a range word.
    local ranges = text_code.ranges
    return ranges.types[word] or ranges.types[ranges.aliases[word]]
end

local function check_mismatch(unit1, unit2)
    -- If unit1 cannot be converted to unit2, return an error message table.
    -- This allows conversion between units of the same type, and between

```

```

-- Nm (normally torque) and ftlb (energy), as in gun-related articles.
-- This works because Nm is the base unit (scale = 1) for both the
-- primary type (torque), and the alternate type (energy, where Nm = J).
-- A match occurs if the primary types are the same, or if unit1 matches
-- the alternate type of unit2, and vice versa. That provides a whitelist
-- of which conversions are permitted between normally incompatible types.
if unit1.utype == unit2.utype or
    (unit1.utype == unit2.alttype and unit1.alttype == unit2.utype) then
    return nil
end
return { 'cvt_mismatch', unit1.utype, unit2.utype }
end

local function override_from(out_table, in_table, fields)
-- Copy the specified fields from in_table to out_table, but do not
-- copy nil fields (keep any corresponding field in out_table).
for _, field in ipairs(fields) do
    if in_table[field] then
        out_table[field] = in_table[field]
    end
end
end

local function shallow_copy(t)
-- Return a shallow copy of table t.
-- Do not need the features and overhead of the Scribunto mw.clone().
local result = {}
for k, v in pairs(t) do
    result[k] = v
end
return result
end

local unit_mt = {
-- Metatable to get missing values for a unit that does not accept SI prefixes.
-- Warning: The boolean value 'false' is returned for any missing field
-- so __index is not called twice for the same field in a given unit.
__index = function (self, key)
    local value
    if key == 'name1' or key == 'sym_us' then
        value = self.symbol
    elseif key == 'name2' then
        value = self.name1 .. plural_suffix
    elseif key == 'name1_us' then
        value = self.name1
        if not rawget(self, 'name2_us') then
            -- If name1_us is 'foot', do not make name2_us by appending plural
            self.name2_us = self.name2
        end
    elseif key == 'name2_us' then
        local raw1_us = rawget(self, 'name1_us')

```

```

        if raw1_us then
            value = raw1_us .. plural_suffix
        else
            value = self.name2
        end
    elseif key == 'link' then
        value = self.name1
    else
        value = false
    end
    rawset(self, key, value)
    return value
end
}

```

```

local function prefixed_name(unit, name, index)
    -- Return unit name with SI prefix inserted at correct position.
    -- index = 1 (name1), 2 (name2), 3 (name1_us), 4 (name2_us).
    -- The position is a byte (not character) index, so use Lua's sub().
    local pos = rawget(unit, 'prefix_position')
    if type(pos) == 'string' then
        pos = tonumber(split(pos, ',')[index])
    end
    if pos then
        return name:sub(1, pos - 1) .. unit.si_name .. name:sub(pos)
    end
    return unit.si_name .. name
end
end

```

```

local unit_prefixed_mt = {
    -- Metatable to get missing values for a unit that accepts SI prefixes.
    -- Before use, fields si_name, si_prefix must be defined.
    -- The unit must define _symbol, _name1 and
    -- may define _sym_us, _name1_us, _name2_us
    -- (_sym_us, _name2_us may be defined for a language using sp=us
    -- to refer to a variant unrelated to U.S. units).
    __index = function (self, key)
        local value
        if key == 'symbol' then
            value = self.si_prefix .. self._symbol
        elseif key == 'sym_us' then
            value = rawget(self, '_sym_us')
            if value then
                value = self.si_prefix .. value
            else
                value = self.symbol
            end
        elseif key == 'name1' then
            value = prefixed_name(self, self._name1, 1)
        elseif key == 'name2' then
            value = rawget(self, '_name2')
        end
    end
}

```

```

        if value then
            value = prefixed_name(self, value, 2)
        else
            value = self.name1 .. plural_suffix
        end
elseif key == 'name1_us' then
    value = rawget(self, '_name1_us')
    if value then
        value = prefixed_name(self, value, 3)
    else
        value = self.name1
    end
elseif key == 'name2_us' then
    value = rawget(self, '_name2_us')
    if value then
        value = prefixed_name(self, value, 4)
    elseif rawget(self, '_name1_us') then
        value = self.name1_us .. plural_suffix
    else
        value = self.name2
    end
elseif key == 'link' then
    value = self.name1
else
    value = false
end
rawset(self, key, value)
return value
end
}

local unit_per_mt = {
    -- Metatable to get values for a per unit of form "x/y".
    -- This is never called to determine a unit name or link because per units
    -- are handled as a special case.
    -- Similarly, the default output is handled elsewhere, and for a symbol
    -- this is only called from get_default() for default_exceptions.
    __index = function (self, key)
        local value
        if key == 'symbol' then
            local per = self.per
            local unit1, unit2 = per[1], per[2]
            if unit1 then
                value = unit1[key] .. '/' .. unit2[key]
            else
                value = '/' .. unit2[key]
            end
        elseif key == 'sym_us' then
            value = self.symbol
        elseif key == 'scale' then
            local per = self.per

```

```

        local unit1, unit2 = per[1], per[2]
        value = (unit1 and unit1.scale or 1) * self.scalemultiplier / unit2.scale
    else
        value = false
    end
    rawset(self, key, value)
    return value
end
}

```

```

local function make_per(unitcode, unit_table, ulookup)
    -- Return true, t where t is a per unit with unit codes expanded to unit tables,
    -- or return false, t where t is an error message table.
    local result = {
        unitcode = unitcode,
        utype = unit_table.utype,
        per = {}
    }
    override_from(result, unit_table, { 'invert', 'iscomplex', 'default', 'link', 'symbol', 'symbol_raw' })
    result.symbol_raw = (result.symbol or false) -- to distinguish between a defined exceptic
    local prefix
    for i, v in ipairs(unit_table.per) do
        if i == 1 and v == '' then
            -- First unit symbol can be empty; that gives a nil first unit table.
        elseif i == 1 and text_code.currency[v] then
            prefix = currency_text or v
        else
            local success, t = ulookup(v)
            if not success then return false, t end
            result.per[i] = t
        end
    end
    end
    local multiplier = unit_table.multiplier
    if not result.utype then
        -- Creating an automatic per unit.
        local unit1 = result.per[1]
        local utype = (unit1 and unit1.utype or prefix or '') .. '/' .. result.per[2].utype
        local t = data_code.per_unit_fixups[utype]
        if t then
            if type(t) == 'table' then
                utype = t.utype or utype
                result.link = result.link or t.link
                multiplier = multiplier or t.multiplier
            else
                utype = t
            end
        end
        end
        result.utype = utype
    end
    result.scalemultiplier = multiplier or 1
    result.vprefix = prefix or false -- set to non-nil to avoid calling __index

```

```

    return true, setmetatable(result, unit_per_mt)
end

local function lookup(parms, unitcode, what, utable, fails, depth)
    -- Return true, t where t is a copy of the unit's converter table,
    -- or return false, t where t is an error message table.
    -- Parameter 'what' determines whether combination units are accepted:
    --   'no_combination' : single unit only
    --   'any_combination' : single unit or combination or output multiple
    --   'only_multiple'  : single unit or output multiple only
    -- Parameter unitcode is a symbol (like 'g'), with an optional SI prefix (like 'kg').
    -- If, for example, 'kg' is in this table, that entry is used;
    -- otherwise the prefix ('k') is applied to the base unit ('g').
    -- If unitcode is a known combination code (and if allowed by what),
    -- a table of output multiple unit tables is included in the result.
    -- For compatibility with the old template, an underscore in a unitcode is
    -- replaced with a space so usage like {{convert|350|board_feet}} works.
    -- Wikignomes may also put two spaces or "&nbsp;" in combinations, so
    -- replace underscore, "&nbsp;", and multiple spaces with a single space.
    utable = utable or parms.unittable or all_units
    fails = fails or {}
    depth = depth and depth + 1 or 1
    if depth > 9 then
        -- There are ways to mistakenly define units which result in infinite
        -- recursion when lookup() is called. That gives a long delay and very
        -- confusing error messages, so the depth parameter is used as a guard.
        return false, { 'cvt_lookup', unitcode }
    end
    if unitcode == nil or unitcode == '' then
        return false, { 'cvt_no_unit' }
    end
    unitcode = unitcode:gsub('_', ' '):gsub('&nbsp;', ' '):gsub(' +', ' ')
    local function call_make_per(t)
        return make_per(unitcode, t,
            function (ucode) return lookup(parms, ucode, 'no_combination', utable, fai
            )
        end
    end
    local t = utable[unitcode]
    if t then
        if t.shouldbe then
            return false, { 'cvt_should_be', t.shouldbe }
        end
        if t.sp_us then
            parms.opt_sp_us = true
        end
        local target = t.target -- nil, or unitcode is an alias for this target
        if target then
            local success, result = lookup(parms, target, what, utable, fails, depth)
            if not success then return false, result end
            override_from(result, t, { 'customary', 'default', 'link', 'symbol', 'sym
            local multiplier = t.multiplier

```

```

        if multiplier then
            result.multiplier = tostring(multiplier)
            result.scale = result.scale * multiplier
        end
        return true, result
    end
    if t.per then
        return call_make_per(t)
    end
    local combo = t.combination -- nil or a table of unitcodes
    if combo then
        local multiple = t.multiple
        if what == 'no_combination' or (what == 'only_multiple' and not multiple)
            return false, { 'cvt_bad_unit', unitcode }
        end
        -- Recursively create a combination table containing the
        -- converter table of each unitcode.
        local result = { utype = t.utype, multiple = multiple, combination = {} }
        local cvt = result.combination
        for i, v in ipairs(combo) do
            local success, t = lookup(parms, v, multiple and 'no_combination'
            if not success then return false, t end
            cvt[i] = t
        end
        return true, result
    end
    local result = shallow_copy(t)
    result.unitcode = unitcode
    if result.prefixes then
        result.si_name = ''
        result.si_prefix = ''
        return true, setmetatable(result, unit_prefixed_mt)
    end
    return true, setmetatable(result, unit_mt)
end
local SIprefixes = text_code.SIprefixes
for plen = SIprefixes[1] or 2, 1, -1 do
    -- Look for an SI prefix; should never occur with an alias.
    -- Check for longer prefix first ('dam' is decametre).
    -- SIprefixes[1] = prefix maximum #characters (as seen by mw.ustr.string.sub).
    local prefix = usub(unitcode, 1, plen)
    local si = SIprefixes[prefix]
    if si then
        local t = utable[usub(unitcode, plen+1)]
        if t and t.prefixes then
            local result = shallow_copy(t)
            result.unitcode = unitcode
            result.si_name = parms.opt_sp_us and si.name_us or si.name
            result.si_prefix = si.prefix or prefix
            result.scale = t.scale * 10 ^ (si.exponent * t.prefixes)
            return true, setmetatable(result, unit_prefixed_mt)
        end
    end
end

```

```

        end
    end
end
-- Accept user-defined combinations like "acre+m2+ha" or "acre m2 ha" for output.
-- If '+' is used, each unit code can include a space, and any error is fatal.
-- If ' ' is used and if each space-separated word is a unit code, it is a combo,
-- but errors are not fatal so the unit code can be looked up as an extra unit.
local err_is_fatal
local combo = collection()
if unitcode:find('+', 1, true) then
    err_is_fatal = true
    for item in (unitcode .. '+'):gmatch('%s*(.-%s*%+') do
        if item ~= '' then
            combo:add(item)
        end
    end
end
elseif unitcode:find('%s') then
    for item in unitcode:gmatch('%S+') do
        combo:add(item)
    end
end
end
if combo.n > 1 then
    local function lookup_combo()
        if what == 'no_combination' or what == 'only_multiple' then
            return false, { 'cvt_bad_unit', unitcode }
        end
        local result = { combination = {} }
        local cvt = result.combination
        for i, v in ipairs(combo) do
            local success, t = lookup(parms, v, 'only_multiple', utable, fail)
            if not success then return false, t end
            if i == 1 then
                result.utype = t.utype
            else
                local mismatch = check_mismatch(result, t)
                if mismatch then
                    return false, mismatch
                end
            end
            cvt[i] = t
        end
        return true, result
    end
    local success, result = lookup_combo()
    if success or err_is_fatal then
        return success, result
    end
end
-- Accept any unit with an engineering notation prefix like "e6cuft"
-- (million cubic feet), but not chained prefixes like "e3e6cuft",
-- and not if the unit is a combination or multiple,

```

```

-- and not if the unit has an offset or is a built-in.
-- Only en digits are accepted.
local exponent, baseunit = unitcode:match('^e(%d+)(.*)')
if exponent then
    local engscale = text_code.eng_scales[exponent]
    if engscale then
        local success, result = lookup(parms, baseunit, 'no_combination', utable,
            if success and not (result.offset or result.builtin or result.engscale) then
                result.unitcode = unitcode -- 'e6cuft' not 'cuft'
                result.defkey = unitcode -- key to lookup default exception
                result.engscale = engscale
                result.scale = result.scale * 10 ^ tonumber(exponent)
                return true, result
            end
        end
    end
end
-- Look for x/y; split on right-most slash to get scale correct (x/y/z is x/y per z).
local top, bottom = unitcode:match('^(-)/([^/]+)$')
if top and not unitcode:find('e%d') then
    -- If valid, create an automatic per unit for an "x/y" unit code.
    -- The unitcode must not include extraneous spaces.
    -- Engineering notation (apart from at start and which has been stripped before here
    -- is not supported so do not make a per unit if find text like 'e3' in unitcode.
    local success, result = call_make_per({ per = {top, bottom} })
    if success then
        return true, result
    end
end
if not parms.opt_ignore_error and not get_range(unitcode) then
    -- Want the "what links here" list for the extra_module to show only cases
    -- where an extra unit is used, so do not require it if invoked from {{val}}
    -- or if looking up a range word which cannot be a unit.
    if not extra_units then
        local success, extra = pcall(function () return require(extra_module).extra_units
            if success and type(extra) == 'table' then
                extra_units = extra
            end
        end
    end
    if extra_units then
        -- A unit in one data table might refer to a unit in the other table, so
        -- switch between them, relying on fails or depth to terminate loops.
        if not fails[unitcode] then
            fails[unitcode] = true
            local other = (utable == all_units) and extra_units or all_units
            local success, result = lookup(parms, unitcode, what, other, fails)
            if success then
                return true, result
            end
        end
    end
end
end
end

```

```

if to_en_table then
    -- At fawiki it is common to translate all digits so a unit like "km2" becomes "km
    local en_code = ustring.gsub(unitcode, '%d', to_en_table)
    if en_code ~= unitcode then
        return lookup(parms, en_code, what, utable, fails, depth)
    end
end
return false, { 'cvt_unknown', unitcode }
end

local function valid_number(num)
    -- Return true if num is a valid number.
    -- In Scribunto (different from some standard Lua), when expressed as a string,
    -- overflow or other problems are indicated with text like "inf" or "nan"
    -- which are regarded as invalid here (each contains "n").
    if type(num) == 'number' and tostring(num):find('n', 1, true) == nil then
        return true
    end
end

local function hyphenated(name, parts)
    -- Return a hyphenated form of given name (for adjectival usage).
    -- The name may be linked and the target of the link must not be changed.
    -- Hypothetical examples:
    -- [[long ton|ton]]          → [[long ton|ton]]          (no change)
    -- [[tonne|long ton]]       → [[tonne|long-ton]]
    -- [[metric ton|long ton]] → [[metric ton|long-ton]]
    -- [[long ton]]             → [[long ton|long-ton]]
    -- Input can also have multiple links in a single name like:
    -- [[United States customary units|U.S.]] [[US gallon|gallon]]
    -- [[mile]]s per [[United States customary units|U.S.]] [[quart]]
    -- [[long ton]]s per [[short ton]]
    -- Assume that links cannot be nested (never like "[[abc[[def]]ghi]]").
    -- This uses a simple and efficient procedure that works for most cases.
    -- Some units (if used) would require more, and can later think about
    -- adding a method to handle exceptions.
    -- The procedure is to replace each space with a hyphen, but
    -- not a space after ')' [for "(pre-1954&nbsp;US) nautical mile"], and
    -- not spaces immediately before '(' or in '(...)' [for cases like
    -- "British thermal unit (ISO)" and "Calorie (International Steam Table)"].
    if name:find(' ', 1, true) then
        if parts then
            local pos
            if name:sub(1, 1) == '(' then
                pos = name:find(')', 1, true)
                if pos then
                    return name:sub(1, pos+1) .. name:sub(pos+2):gsub(' ', '-')
                end
            elseif name:sub(-1) == ')' then
                pos = name:find('(', 1, true)
                if pos then

```

```

        return name:sub(1, pos-2):gsub(' ', '-') .. name:sub(pos-1
    end
    end
    return name:gsub(' ', '-')
end
parts = collection()
for before, item, after in name:gmatch('([^\[]*)(%[[^\[]*%])((^\[]*)') do
    if item:find(' ', 1, true) then
        local prefix
        local plen = item:find('|', 1, true)
        if plen then
            prefix = item:sub(1, plen)
            item = item:sub(plen + 1, -3)
        else
            prefix = item:sub(1, -3) .. '|'
            item = item:sub(3, -3)
        end
        item = prefix .. hyphenated(item, parts) .. ']'
    end
    parts:add(before:gsub(' ', '-') .. item .. after:gsub(' ', '-'))
end
if parts.n == 0 then
    -- No link like "[[...]]" was found in the original name.
    parts:add(hyphenated(name, parts))
end
return table.concat(parts)
end
return name
end
end

```

```

local function hyphenated_maybe(parms, want_name, sep, id, inout)
    -- Return s, f where
    -- s = id, possibly modified
    -- f = true if hyphenated
    -- Possible modifications: hyphenate; prepend '-'; append mid text.
    if id == nil or id == '' then
        return ''
    end
    local mid = (inout == (parms.opt_flip and 'out' or 'in')) and parms.mid or ''
    if want_name then
        if parms.opt_adjectival then
            return '-' .. hyphenated(id) .. mid, true
        end
        if parms.opt_add_s and id:sub(-1) ~= 's' then
            id = id .. 's' -- for nowiki
        end
    end
    return sep .. id .. mid
end
end

```

```

local function use_minus(text)

```

```

-- Return text with Unicode minus instead of '-', if present.
if text:sub(1, 1) == '-' then
    return MINUS .. text:sub(2)
end
return text
end

local function digit_groups(parms, text, method)
    -- Return a numbered table of groups of digits (left-to-right, in local language).
    -- Parameter method is a number or nil:
    -- 3 for 3-digit grouping (default), or
    -- 2 for 3-then-2 grouping (only for digits before decimal mark).
    local len_right
    local len_left = text:find('.', 1, true)
    if len_left then
        len_right = #text - len_left
        len_left = len_left - 1
    else
        len_left = #text
    end
    local twos = method == 2 and len_left > 5
    local groups = collection()
    local run = len_left
    local n
    if run < 4 or (run == 4 and parms.opt_comma5) then
        if parms.opt_gaps then
            n = run
        else
            n = #text
        end
    elseif twos then
        n = run % 2 == 0 and 1 or 2
    else
        n = run % 3 == 0 and 3 or run % 3
    end
    while run > 0 do
        groups:add(n)
        run = run - n
        n = (twos and run > 3) and 2 or 3
    end
    if len_right then
        if groups.n == 0 then
            groups:add(0)
        end
        if parms.opt_gaps and len_right > 3 then
            local want4 = not parms.opt_gaps3 -- true gives no gap before trailing si
            local isfirst = true
            run = len_right
            while run > 0 do
                n = (want4 and run == 4) and 4 or (run > 3 and 3 or run)
                if isfirst then

```

```

            isfirst = false
            groups[groups.n] = groups[groups.n] + 1 + n
        else
            groups:add(n)
        end
        run = run - n
    end
else
    groups[groups.n] = groups[groups.n] + 1 + len_right
end
end
local pos = 1
for i, length in ipairs(groups) do
    groups[i] = from_en(text:sub(pos, pos + length - 1))
    pos = pos + length
end
return groups
end

function with_separator(parms, text) -- for forward declaration above
-- Input text is a number in en digits with optional '.' decimal mark.
-- Return an equivalent, formatted for display:
--   with a custom decimal mark instead of '.', if wanted
--   with thousand separators inserted, if wanted
--   digits in local language
-- The given text is like '123' or '123.' or '12345.6789'.
-- The text has no sign (caller inserts that later, if necessary).
-- When using gaps, they are inserted before and after the decimal mark.
-- Separators are inserted only before the decimal mark.
-- A trailing dot (as in '123.') is removed because their use appears to
-- be accidental, and such a number should be shown as '123' or '123.0'.
-- It is useful for convert to suppress the dot so, for example, '4000.'
-- is a simple way of indicating that all the digits are significant.
if text:sub(-1) == '.' then
    text = text:sub(1, -2)
end
if #text < 4 or parms.opt_nocomma or numsep == '' then
    return from_en(text)
end
local groups = digit_groups(parms, text, group_method)
if parms.opt_gaps then
    if groups.n <= 1 then
        return groups[1] or ''
    end
    local nowrap = '<span style="white-space: nowrap">'
    local gap = '<span style="margin-left: 0.25em">'
    local close = '</span>'
    return nowrap .. groups[1] .. gap .. table.concat(groups, close .. gap, 2, groups)
end
return table.concat(groups, numsep)
end
end

```

```

-- An input value like 1.23e12 is displayed using scientific notation (1.23×1012).
-- That also makes the output use scientific notation, except for small values.
-- In addition, very small or very large output values use scientific notation.
-- Use format(fmtpower, significand, '10', exponent) where each argument is a string.
local fmtpower = '%s>x</span>%s<sup>%s</sup>'

local function with_exponent(parms, show, exponent)
    -- Return wikitext to display the implied value in scientific notation.
    -- Input uses en digits; output uses digits in local language.
    return format(fmtpower, with_separator(parms, show), from_en('10'), use_minus(from_en(tostring(show))))
end

local function make_sigfig(value, sigfig)
    -- Return show, exponent that are equivalent to the result of
    -- converting the number 'value' (where value >= 0) to a string,
    -- rounded to 'sigfig' significant figures.
    -- The returned items are:
    -- show: a string of digits; no sign and no dot;
    --       there is an implied dot before show.
    -- exponent: a number (an integer) to shift the implied dot.
    -- Resulting value = tonumber('.') .. show * 10^exponent.
    -- Examples:
    -- make_sigfig(23.456, 3) returns '235', 2 (.235 * 10^2).
    -- make_sigfig(0.0023456, 3) returns '235', -2 (.235 * 10^-2).
    -- make_sigfig(0, 3) returns '000', 1 (.000 * 10^1).
    if sigfig <= 0 then
        sigfig = 1
    elseif sigfig > maxsigfig then
        sigfig = maxsigfig
    end
    if value == 0 then
        return string.rep('0', sigfig), 1
    end
    local exp, fracpart = math.modf(log10(value))
    if fracpart >= 0 then
        fracpart = fracpart - 1
        exp = exp + 1
    end
    local digits = format('%f', 10^(fracpart + sigfig))
    if #digits > sigfig then
        -- Overflow (for sigfig=3: like 0.9999 rounding to "1000"; need "100").
        digits = digits:sub(1, sigfig)
        exp = exp + 1
    end
    assert(#digits == sigfig, 'Bug: rounded number has wrong length')
    return digits, exp
end

-- Fraction output format.
local fracfmt = {

```

```

{ -- Like {{frac}} (fraction slash).
  '<span class="frac" role="math">{SIGN}<span class="num">{NUM}</span>&fracl;<span c
  '<span class="frac" role="math">{SIGN}{WHOLE}<span class="sr-only">+</span><span c
  style = 'frac',
},
{ -- Like {{sfrac}} (stacked fraction, that is, horizontal bar).
  '<span class="sfrac tion" role="math">{SIGN}<span class="num">{NUM}</span><span cl
  '<span class="sfrac" role="math">{SIGN}{WHOLE}<span class="sr-only">+</span><span
  style = 'sfrac',
},
}

```

```

local function format_fraction(parms, inout, negative, wholestr, numstr, denstr, do_spell, style)
  -- Return wikitext for a fraction, possibly spelled.
  -- Inputs use en digits and have no sign; output uses digits in local language.
  local wikitext
  if not style then
    style = parms.opt_fraction_horizontal and 2 or 1
  end
  if wholestr == '' then
    wholestr = nil
  end
  local substitute = {
    SIGN = negative and MINUS or '',
    WHOLE = wholestr and with_separator(parms, wholestr),
    NUM = from_en(numstr),
    DEN = from_en(denstr),
  }
  wikitext = fracfmt[style][wholestr and 2 or 1]:gsub('{(%u+)}', substitute)
  if do_spell then
    if negative then
      if wholestr then
        wholestr = '-' .. wholestr
      else
        numstr = '-' .. numstr
      end
    end
    local s = spell_number(parms, inout, wholestr, numstr, denstr)
    if s then
      return s
    end
  end
  add_style(parms, fracfmt[style].style)
  return wikitext
end

```

```

local function format_number(parms, show, exponent, isnegative)
  -- Parameter show is a string or a table containing strings.
  -- Each string is a formatted number in en digits and optional '.' decimal mark.
  -- A table represents a fraction: integer, numerator, denominator;
  -- if a table is given, exponent must be nil.

```

```

-- Return t where t is a table with fields:
-- show = wikitext formatted to display implied value
--       (digits in local language)
-- is_scientific = true if show uses scientific notation
-- clean = unformatted show (possibly adjusted and with inserted '.')
--       (en digits)
-- sign = '' or MINUS
-- exponent = exponent (possibly adjusted)
-- The clean and exponent fields can be used to calculate the
-- rounded absolute value, if needed.
--
-- The value implied by the arguments is found from:
-- exponent is nil; and
-- show is a string of digits (no sign), with an optional dot;
-- show = '123.4' is value 123.4, '1234' is value 1234.0;
-- or:
-- exponent is an integer indicating where dot should be;
-- show is a string of digits (no sign and no dot);
-- there is an implied dot before show;
-- show does not start with '0';
-- show = '1234', exponent = 3 is value 0.1234*10^3 = 123.4.
--
-- The formatted result:
-- * Is for an output value and is spelled if wanted and possible.
-- * Includes a Unicode minus if isnegative and not spelled.
-- * Uses a custom decimal mark, if wanted.
-- * Has digits grouped where necessary, if wanted.
-- * Uses scientific notation if requested, or for very small or large values
--   (which forces result to not be spelled).
-- * Has no more than maxsigfig significant digits
--   (same as old template and {{#expr}}).
local xhi, xlo -- these control when scientific notation (exponent) is used
if parms.opt_scientific then
    xhi, xlo = 4, 2 -- default for output if input uses e-notation
elseif parms.opt_scientific_always then
    xhi, xlo = 0, 0 -- always use scientific notation (experimental)
else
    xhi, xlo = 10, 4 -- default
end
local sign = isnegative and MINUS or ''
local maxlen = maxsigfig
local tfrac
if type(show) == 'table' then
    tfrac = show
    show = tfrac.wholestr
    assert(exponent == nil, 'Bug: exponent given with fraction')
end
if not tfrac and not exponent then
    local integer, dot, decimals = show:match('^(%d*)(%?)(.*)')
    if integer == '0' or integer == '' then
        local zeros, figs = decimals:match('^(0*)([^\0]?.*')

```

```

        if #figs == 0 then
            if #zeros > maxlen then
                show = '0.' .. zeros:sub(1, maxlen)
            end
        elseif #zeros >= xlo then
            show = figs
            exponent = -#zeros
        elseif #figs > maxlen then
            show = '0.' .. zeros .. figs:sub(1, maxlen)
        end
    elseif #integer >= xhi then
        show = integer .. decimals
        exponent = #integer
    else
        maxlen = maxlen + #dot
        if #show > maxlen then
            show = show:sub(1, maxlen)
        end
    end
end
if exponent then
    local function zeros(n)
        return string.rep('0', n)
    end
    if #show > maxlen then
        show = show:sub(1, maxlen)
    end
    if exponent > xhi or exponent <= -xlo or (exponent == xhi and show ~= '1' .. zeros:
        -- When xhi, xlo = 10, 4 (the default), scientific notation is used if the
        -- rounded value satisfies: value >= 1e9 or value < 1e-4 (1e9 = 0.1e10),
        -- except if show is '1000000000' (1e9), for example:
        -- {{convert|1000000000|m|m|sigfig=10}} → 1,000,000,000 metres (1,000,000,
        local significand
        if #show > 1 then
            significand = show:sub(1, 1) .. '.' .. show:sub(2)
        else
            significand = show
        end
        return {
            clean = '.' .. show,
            exponent = exponent,
            sign = sign,
            show = sign .. with_exponent(parms, significand, exponent-1),
            is_scientific = true,
        }
    end
    if exponent >= #show then
        show = show .. zeros(exponent - #show) -- result has no dot
    elseif exponent <= 0 then
        show = '0.' .. zeros(-exponent) .. show
    else

```

```

        show = show:sub(1, exponent) .. '.' .. show:sub(exponent+1)
    end
end
local formatted_show
if tfrac then
    show = tostring(tfrac.value) -- to set clean in returned table
    formatted_show = format_fraction(parms, 'out', isnegative, tfrac.wholestr, tfrac.r
else
    if isnegative and show:match('^0.?0*$') then
        sign = '' -- don't show minus if result is negative but rounds to zero
    end
    formatted_show = sign .. with_separator(parms, show)
    if parms.opt_spell_out then
        formatted_show = spell_number(parms, 'out', sign .. show) or formatted_shc
    end
end
end
return {
    clean = show,
    sign = sign,
    show = formatted_show,
    is_scientific = false, -- to avoid calling __index
}
end

```

```

local function extract_fraction(parms, text, negative)
    -- If text represents a fraction, return
    -- value, altvalue, show, denominator
    -- where
    -- value is a number (value of the fraction in argument text)
    -- altvalue is an alternate interpretation of any fraction for the hands
    -- unit where "12.1+3/4" means 12 hands 1.75 inches
    -- show is a string (formatted text for display of an input value,
    -- and is spelled if wanted and possible)
    -- denominator is value of the denominator in the fraction
    -- Otherwise, return nil.
    -- Input uses en digits and '.' decimal mark (input has been translated).
    -- Output uses digits in local language and local decimal mark, if any.
    -----
    -- Originally this function accepted x+y/z where x, y, z were any valid
    -- numbers, possibly with a sign. For example '1.23e+2+1.2/2.4' = 123.5,
    -- and '2-3/8' = 1.625. However, such usages were found to be errors or
    -- misunderstandings, so since August 2014 the following restrictions apply:
    -- x (if present) is an integer or has a single digit after decimal mark
    -- y and z are unsigned integers
    -- e-notation is not accepted
    -- The overall number can start with '+' or '-' (so '12+3/4' and '+12+3/4'
    -- and '-12-3/4' are valid).
    -- Any leading negative sign is removed by the caller, so only inputs
    -- like the following are accepted here (may have whitespace):
    -- negative = false      false      true (there was a leading '-')
    -- text      = '2/3'      '+2/3'      '2/3'

```

```

-- text      = '1+2/3'      '+1+2/3'      '1-2/3'
-- text      = '12.3+1/2'  '+12.3+1/2'  '12.3-1/2'
-- Values like '12.3+1/2' are accepted, but are intended only for use
-- with the hands unit (not worth adding code to enforce that).
-----
local leading_plus, prefix, numstr, slashes, denstr =
    text:match('^%s*(%+?)%s*(.)%s*(%d+)%s*(/)%s*(%d+)%s*$')
if not leading_plus then
    -- Accept a single U+2044 fraction slash because that may be pasted.
    leading_plus, prefix, numstr, denstr =
        text:match('^%s*(%+?)%s*(.)%s*(%d+)%s*%s*(%d+)%s*$')
    slashes = '/'
end
local numerator = tonumber(numstr)
local denominator = tonumber(denstr)
if numerator == nil or denominator == nil or (negative and leading_plus ~= '') then
    return nil
end
local whole, wholestr
if prefix == '' then
    wholestr = ''
    whole = 0
else
    -- Any prefix must be like '12+' or '12-' (whole number and fraction sign);
    -- '12.3+' and '12.3-' are also accepted (single digit after decimal point)
    -- because '12.3+1/2 hands' is valid (12 hands 3½ inches).
    local num1, num2, frac_sign = prefix:match('^(%d+)(%.?%d?)%s*([+%-])$')
    if num1 == nil then return nil end
    if num2 == '' then -- num2 must be '' or like '.1' but not '.' or '.12'
        wholestr = num1
    else
        if #num2 ~= 2 then return nil end
        wholestr = num1 .. num2
    end
    if frac_sign ~= (negative and '-' or '+') then return nil end
    whole = tonumber(wholestr)
    if whole == nil then return nil end
end
local value = whole + numerator / denominator
if not valid_number(value) then return nil end
local altvalue = whole + numerator / (denominator * 10)
local style = #slashes -- kludge: 1 or 2 slashes can be used to select style
if style > 2 then style = 2 end
local wikitext = format_fraction(parms, 'in', negative, leading_plus .. wholestr, numstr,
return value, altvalue, wikitext, denominator
end

local function extract_number(parms, text, another, no_fraction)
    -- Return true, info if can extract a number from text,
    -- where info is a table with the result,
    -- or return false, t where t is an error message table.

```

```

-- Input can use en digits or digits in local language and can
-- have references at the end. Accepting references is intended
-- for use in infoboxes with a field for a value passed to convert.
-- Parameter another = true if the expected value is not the first.
-- Before processing, the input text is cleaned:
-- * Any thousand separators (valid or not) are removed.
-- * Any sign is replaced with '-' (if negative) or '' (otherwise).
--   That replaces Unicode minus with '-'.
-- If successful, the returned info table contains named fields:
-- value      = a valid number
-- altvalue   = a valid number, usually same as value but different
--             if fraction used (for hands unit)
-- singular   = true if value is 1 or -1 (to use singular form of units)
-- clean      = cleaned text with any separators and sign removed
--             (en digits and '.' decimal mark)
-- show       = text formatted for output, possibly with ref strip markers
--             (digits in local language and custom decimal mark)
-- The resulting show:
-- * Is for an input value and is spelled if wanted and possible.
-- * Has a rounded value, if wanted.
-- * Has digits grouped where necessary, if wanted.
-- * If negative, a Unicode minus is used; otherwise the sign is
--   '+' (if the input text used '+'), or is '' (if no sign in input).
text = strip(text or '')
local reference
local pos = text:find('\127', 1, true)
if pos then
    local before = text:sub(1, pos - 1)
    local remainder = text:sub(pos)
    local refs = {}
    while #remainder > 0 do
        local ref, spaces
        ref, spaces, remainder = remainder:match('^(\\127[\\^\\127]*UNIQ[\\^\\127]*%-ref|
        if ref then
            table.insert(refs, ref)
        else
            refs = {}
            break
        end
    end
    if #refs > 0 then
        text = strip(before)
        reference = table.concat(refs)
    end
end
local clean = to_en(text, parms)
if clean == '' then
    return false, { another and 'cvt_no_num2' or 'cvt_no_num' }
end
local isnegative, propersign = false, '' -- most common case
local singular, show, denominator

```

```

local value = tonumber(clean)
local altvalue
if value then
    local sign = clean:sub(1, 1)
    if sign == '+' or sign == '-' then
        propersign = (sign == '+') and '+' or MINUS
        clean = clean:sub(2)
    end
    if value < 0 then
        isnegative = true
        value = -value
    end
else
    local valstr
    for _, prefix in ipairs({ '-', MINUS, '&minus;' }) do
        -- Including '-' sets isnegative in case input is a fraction like '-2-3/4'
        local plen = #prefix
        if clean:sub(1, plen) == prefix then
            valstr = clean:sub(plen + 1)
            if valstr:match('^%s') then -- "- 1" is invalid but "-1 - 1/2" is
                return false, { 'cvt_bad_num', text }
            end
            break
        end
    end
    if valstr then
        isnegative = true
        propersign = MINUS
        clean = valstr
        value = tonumber(clean)
    end
    if value == nil then
        if not no_fraction then
            value, altvalue, show, denominator = extract_fraction(parms, clean)
        end
        if value == nil then
            return false, { 'cvt_bad_num', text }
        end
        if value <= 1 then
            singular = true -- for example, "½ mile" or "one half mile" (sing
        end
    end
end
end
if not valid_number(value) then -- for example, "1e310" may overflow
    return false, { 'cvt_invalid_num' }
end
if show == nil then
    -- clean is a non-empty string with no spaces, and does not represent a fraction,
    -- and value = tonumber(clean) is a number >= 0.
    -- If the input uses e-notation, show will be displayed using a power of ten, but
    -- we use the number as given so it might not be normalized scientific notation.

```

```

-- The input value is spelled if specified so any e-notation is ignored;
-- that allows input like 2e6 to be spelled as "two million" which works
-- because the spell module converts '2e6' to '2000000' before spelling.
local function rounded(value, default, exponent)
    local precision = parms.opt_ri
    if precision then
        if precision then
            local fmt = '%.' .. format('%d', precision) .. 'f'
            local result = fmt:format(tonumber(value) + 2e-14) -- fudge for s
            if not exponent then
                singular = (tonumber(result) == 1)
            end
            return result
        end
        return default
    end
end
singular = (value == 1)
local scientific
local significand, exponent = clean:match('^([%d.]+)[Ee]([%+%-]?%d+)')
if significand then
    show = with_exponent(parms, rounded(significand, significand, exponent), e
    scientific = true
else
    show = with_separator(parms, rounded(value, clean))
end
show = propersign .. show
if parms.opt_spell_in then
    show = spell_number(parms, 'in', propersign .. rounded(value, clean)) or s
    scientific = false
end
if scientific then
    parms.opt_scientific = true
end
end
if isnegative and (value ~= 0) then
    value = -value
    altvalue = -(altvalue or value)
end
return true, {
    value = value,
    altvalue = altvalue or value,
    singular = singular,
    clean = clean,
    show = show .. (reference or ''),
    denominator = denominator,
}

```

end

```

local function get_number(text)
    -- Return v, f where:
    -- v = nil (text is not a number)
    -- or

```

```

-- v = value of text (text is a number)
-- f = true if value is an integer
-- Input can use en digits or digits in local language or separators,
-- but no Unicode minus, and no fraction.
if text then
    local number = tonumber(to_en(text))
    if number then
        local _, fracpart = math.modf(number)
        return number, (fracpart == 0)
    end
end
end

local function gcd(a, b)
    -- Return the greatest common denominator for the given values,
    -- which are known to be positive integers.
    if a > b then
        a, b = b, a
    end
    if a <= 0 then
        return b
    end
    local r = b % a
    if r <= 0 then
        return a
    end
    if r == 1 then
        return 1
    end
    return gcd(r, a)
end

local function fraction_table(value, denominator)
    -- Return value as a string or a table:
    -- * If result is a string, there is no fraction, and the result
    --   is value formatted as a string of en digits.
    -- * If result is a table, it represents a fraction with named fields:
    --   wholestr, numstr, denstr (strings of en digits for integer, numerator, denominator).
    -- The result is rounded to the nearest multiple of (1/denominator).
    -- If the multiple is zero, no fraction is included.
    -- No fraction is included if value is very large as the fraction would
    -- be unhelpful, particularly if scientific notation is required.
    -- Input value is a non-negative number.
    -- Input denominator is a positive integer for the desired fraction.
    if value <= 0 then
        return '0'
    end
    if denominator <= 0 or value > 1e8 then
        return format('%.2f', value)
    end
    local integer, decimals = math.modf(value)

```

```

local numerator = floor((decimals * denominator) +
    0.5 + 2e-14) -- add fudge for some common cases of bad rounding
if numerator >= denominator then
    integer = integer + 1
    numerator = 0
end
local wholestr = tostring(integer)
if numerator > 0 then
    local div = gcd(numerator, denominator)
    if div > 1 then
        numerator = numerator / div
        denominator = denominator / div
    end
    return {
        wholestr = (integer > 0) and wholestr or '',
        numstr = tostring(numerator),
        denstr = tostring(denominator),
        value = value,
    }
end
return wholestr
end

local function preunits(count, preunit1, preunit2)
    -- If count is 1:
    --     ignore preunit2
    --     return p1
    -- else:
    --     preunit1 is used for preunit2 if the latter is empty
    --     return p1, p2
    -- where:
    --     p1 is text to insert before the input unit
    --     p2 is text to insert before the output unit
    --     p1 or p2 may be nil to mean "no preunit"
    -- Using '+' gives output like "5+ feet" (no space before, but space after).
local function withspace(text, wantboth)
    -- Return text with space before and, if wantboth, after.
    -- However, no space is added if there is a space or '&nbsp;' or '-'
    -- at that position ('-' is for adjectival text).
    -- There is also no space if text starts with '&'
    -- (e.g. '&deg;' would display a degree symbol with no preceding space).
    local char = text:sub(1, 1)
    if char == '&' then
        return text -- an html entity can be used to specify the exact display
    end
    if not (char == ' ' or char == '-' or char == '+') then
        text = ' ' .. text
    end
    if wantboth then
        char = text:sub(-1, -1)
        if not (char == ' ' or char == '-' or text:sub(-6, -1) == '&nbsp;') then

```

```

        text = text .. ' '
    end
    end
    return text
end
local PLUS = '+'
preunit1 = preunit1 or ''
local trim1 = strip(preunit1)
if count == 1 then
    if trim1 == '' then
        return nil
    end
    if trim1 == '+' then
        return PLUS
    end
    return withspace(preunit1, true)
end
preunit1 = withspace(preunit1)
preunit2 = preunit2 or ''
local trim2 = strip(preunit2)
if trim1 == '+' then
    if trim2 == '' or trim2 == '+' then
        return PLUS, PLUS
    end
    preunit1 = PLUS
end
if trim2 == '' then
    if trim1 == '' then
        return nil, nil
    end
    preunit2 = preunit1
elseif trim2 == '+' then
    preunit2 = PLUS
elseif trim2 == '&#32;' then -- trick to make preunit2 empty
    preunit2 = nil
else
    preunit2 = withspace(preunit2)
end
return preunit1, preunit2
end

```

```

local function range_text(range, want_name, parms, before, after, inout, options)
-- Return before .. rtext .. after
-- where rtext is the text that separates two values in a range.
local rtext, adj_text, exception
options = options or {}
if type(range) == 'table' then
-- Table must specify range text for ('off' and 'on') or ('input' and 'output'),
-- and may specify range text for 'adj=on',
-- and may specify exception = true.
rtext = range[want_name and 'off' or 'on'] or

```

```

        range(((inout == 'in') == (parms.opt_flip == true)) and 'output' (
adj_text = range['adj']
exception = range['exception']
else
    rtext = range
end
if parms.opt_adjectival then
    if want_name or (exception and parms.abbr_org == 'on') then
        rtext = adj_text or rtext:gsub(' ', '-'):gsub('&nbsp;', '-')
    end
end
if rtext == '-' and (options.spaced or after:sub(1, #MINUS) == MINUS) then
    rtext = '&nbsp;- '
end
return before .. rtext .. after
end

```

```

local function get_composite(parms, iparm, in_unit_table)
-- Look for a composite input unit. For example, {{convert|1|yd|2|ft|3|in}}
-- would result in a call to this function with
-- iparm = 3 (parms[iparm] = "2", just after the first unit)
-- in_unit_table = (unit table for "yd"; contains value 1 for number of yards)
-- Return true, iparm, unit where
-- iparm = index just after the composite units (7 in above example)
-- unit = composite unit table holding all input units,
-- or return true if no composite unit is present in parms,
-- or return false, t where t is an error message table.
local default, subinfo
local composite_units, count = { in_unit_table }, 1
local fixups = {}
local total = in_unit_table.valinfo[1].value
local subunit = in_unit_table
while subunit.subdivs do -- subdivs is nil or a table of allowed subdivisions
    local subcode = strip(parms[iparm+1])
    local subdiv = subunit.subdivs[subcode] or subunit.subdivs[(all_units[subcode] or
if not subdiv then
        break
    end
    local success
    success, subunit = lookup(parms, subcode, 'no_combination')
    if not success then return false, subunit end -- should never occur
    success, subinfo = extract_number(parms, parms[iparm])
    if not success then return false, subinfo end
    iparm = iparm + 2
    subunit.inout = 'in'
    subunit.valinfo = { subinfo }
    -- Recalculate total as a number of subdivisions.
    -- subdiv[1] = number of subdivisions per previous unit (integer > 1).
    total = total * subdiv[1] + subinfo.value
    if not default then -- set by the first subdiv with a default defined
        default = subdiv.default
    end
end

```

```

        end
        count = count + 1
        composite_units[count] = subunit
        if subdiv.unit or subdiv.name then
            fixups[count] = { unit = subdiv.unit, name = subdiv.name, valinfo = subuni
        end
    end
end
if count == 1 then
    return true -- no error and no composite unit
end
for i, fixup in pairs(fixups) do
    local unit = fixup.unit
    local name = fixup.name
    if not unit or (count > 2 and name) then
        composite_units[i].fixed_name = name
    else
        local success, alternate = lookup(parms, unit, 'no_combination')
        if not success then return false, alternate end -- should never occur
        alternate.inout = 'in'
        alternate.valinfo = fixup.valinfo
        composite_units[i] = alternate
    end
end
return true, iparm, {
    utype = in_unit_table.utype,
    scale = subunit.scale, -- scale of last (least significant) unit
    valinfo = { { value = total, clean = subinfo.clean, denominator = subinfo.denomina
    composite = composite_units,
    default = default or in_unit_table.default
}
end

```

```

local function translate_parms(parms, kv_pairs)
    -- Update fields in parms by translating each key:value in kv_pairs to terms
    -- used by this module (may involve translating from local language to English).
    -- Also, checks are performed which may display warnings, if enabled.
    -- Return true if successful or return false, t where t is an error message table.
    currency_text = nil -- local testing can hold module in memory; must clear globals
    if kv_pairs.adj and kv_pairs.sing then
        -- For enwiki (before translation), warn if attempt to use adj and sing
        -- as the latter is a deprecated alias for the former.
        if kv_pairs.adj ~= kv_pairs.sing and kv_pairs.sing ~= '' then
            add_warning(parms, 1, 'cvt_unknown_option', 'sing=' .. kv_pairs.sing)
        end
        kv_pairs.sing = nil
    end
end
kv_pairs.comma = kv_pairs.comma or config.comma -- for plwiki who want default comma=5
for loc_name, loc_value in pairs(kv_pairs) do
    local en_name = text_code.en_option_name[loc_name]
    if en_name then
        local en_value = text_code.en_option_value[en_name]
    end
end

```

```

if en_value == 'INTEGER' then -- altitude_ft, altitude_m, frac, sigfig
  en_value = nil
  if loc_value == '' then
    add_warning(parms, 2, 'cvt_empty_option', loc_name)
  else
    local minimum
    local number, is_integer = get_number(loc_value)
    if en_name == 'sigfig' then
      minimum = 1
    elseif en_name == 'frac' then
      minimum = 2
      if number and number < 0 then
        parms.opt_fraction_horizontal = true
        number = -number
      end
    else
      minimum = -1e6
    end
    if number and is_integer and number >= minimum then
      en_value = number
    else
      local m
      if en_name == 'frac' then
        m = 'cvt_bad_frac'
      elseif en_name == 'sigfig' then
        m = 'cvt_bad_sigfig'
      else
        m = 'cvt_bad_altitude'
      end
      add_warning(parms, 1, m, loc_name .. '=' .. loc_value)
    end
  end
end
elseif en_value == 'TEXT' then -- $, input, qid, qual, stylein, styleout,
  en_value = loc_value ~= '' and loc_value or nil -- accept non-empty
  if not en_value and (en_name == '$' or en_name == 'qid' or en_name == 'qual') then
    add_warning(parms, 2, 'cvt_empty_option', loc_name)
  elseif en_name == '$' then
    -- Value should be a single character like "€" for the eu
    currency_text = (loc_value == 'euro') and '€' or loc_value
  elseif en_name == 'input' then
    -- May have something like {{convert|input=}} (empty input
    -- with optional fields. In that case, want to output notl
    parms.input_text = loc_value -- keep input because parms.
  end
end
else
  en_value = en_value[loc_value]
  if en_value and en_value:sub(-1) == '?' then
    en_value = en_value:sub(1, -2)
    add_warning(parms, -1, 'cvt_deprecated', loc_name .. '=' .. loc_value)
  end
end
if en_value == nil then

```

```

        if loc_value == '' then
            add_warning(parms, 2, 'cvt_empty_option', loc_name)
        else
            add_warning(parms, 1, 'cvt_unknown_option', loc_name)
        end
    elseif en_value == '' then
        en_value = nil -- an ignored option like adj=off
    elseif type(en_value) == 'string' and en_value:sub(1, 4) == 'opt_'
        for _, v in ipairs(split(en_value, ',')) do
            local lhs, rhs = v:match('^(-)=(.+)$')
            if rhs then
                parms[lhs] = tonumber(rhs) or rhs
            else
                parms[v] = true
            end
        end
        en_value = nil
    end
end
parms[en_name] = en_value
else
    add_warning(parms, 1, 'cvt_unknown_option', loc_name .. '=' .. loc_value)
end
end
local abbr_entered = parms.abbr
local cfg_abbr = config.abbr
if cfg_abbr then
    -- Don't warn if invalid because every convert would show that warning.
    if cfg_abbr == 'on always' then
        parms.abbr = 'on'
    elseif cfg_abbr == 'off always' then
        parms.abbr = 'off'
    elseif parms.abbr == nil then
        if cfg_abbr == 'on default' then
            parms.abbr = 'on'
        elseif cfg_abbr == 'off default' then
            parms.abbr = 'off'
        end
    end
end
end
if parms.abbr then
    if parms.abbr == 'unit' then
        parms.abbr = 'on'
        parms.number_word = true
    end
    parms.abbr_org = parms.abbr -- original abbr, before any flip
elseif parms.opt_hand_hh then
    parms.abbr_org = 'on'
    parms.abbr = 'on'
else
    parms.abbr = 'out' -- default is to abbreviate output only (use symbol, not name)
end

```

```

end
if parms.opt_order_out then
    -- Disable options that do not work in a useful way with order=out.
    parms.opt_flip = nil -- override adj=flip
    parms.opt_spell_in = nil
    parms.opt_spell_out = nil
    parms.opt_spell_upper = nil
end
if parms.opt_spell_out and not abbr_entered then
    parms.abbr = 'off' -- should show unit name when spelling the output value
end
if parms.opt_flip then
    local function swap_in_out(option)
        local value = parms[option]
        if value == 'in' then
            parms[option] = 'out'
        elseif value == 'out' then
            parms[option] = 'in'
        end
    end
    swap_in_out('abbr')
    swap_in_out('lk')
    if parms.opt_spell_in and not parms.opt_spell_out then
        -- For simplicity, and because it does not appear to be needed,
        -- user cannot set an option to spell the output only.
        parms.opt_spell_in = nil
        parms.opt_spell_out = true
    end
end
if parms.opt_spell_upper then
    parms.spell_upper = parms.opt_flip and 'out' or 'in'
end
if parms.opt_table or parms.opt_tablecen then
    if abbr_entered == nil and parms.lk == nil then
        parms.opt_values = true
    end
    parms.table_align = parms.opt_table and 'right' or 'center'
end
if parms.table_align or parms.opt_sortable_on then
    parms.need_table_or_sort = true
end
local disp_joins = text_code.disp_joins
local default_joins = disp_joins['b']
parms.join_between = default_joins[3] or ';'
local disp = parms.disp
if disp == nil then -- special case for the most common setting
    parms.joins = default_joins
elseif disp == 'x' then
    -- Later, parms.joins is set from the input parameters.
else
    -- Old template does this.

```

```

    local abbr = parms.abbr
    if disp == 'slash' then
        if abbr_entered == nil then
            disp = 'slash-nbsp'
        elseif abbr == 'in' or abbr == 'out' then
            disp = 'slash-sp'
        else
            disp = 'slash-nosp'
        end
    elseif disp == 'sqbr' then
        if abbr == 'on' then
            disp = 'sqbr-nbsp'
        else
            disp = 'sqbr-sp'
        end
    end
    parms.joins = disp_joins[disp] or default_joins
    parms.join_between = parms.joins[3] or parms.join_between
    parms.wantname = parms.joins.wantname
end
if (en_default and not parms.opt_lang_local and (parms[1] or ''):find('%d')) or parms.opt_
    from_en_table = nil
end
if en_default and from_en_table then
    -- For hiwiki: localized symbol/name is defined with the US symbol/name field,
    -- and is used if output uses localized numbers.
    parms.opt_sp_us = true
end
return true
end

local function get_values(parms)
    -- If successful, update parms and return true, v, i where
    -- v = table of input values
    -- i = index to next entry in parms after those processed here
    -- or return false, t where t is an error message table.
    local valinfo = collection() -- numbered table of input values
    local range = collection() -- numbered table of range items (having, for example, 2 range
    local had_nocomma -- true if removed "nocomma" kludge from second parameter (like "tonoc
    local parm2 = strip(parms[2])
    if parm2 and parm2:sub(-7, -1) == 'nocomma' then
        parms[2] = strip(parm2:sub(1, -8))
        parms.opt_nocomma = true
        had_nocomma = true
    end
    local function extractor(i)
        -- If the parameter is not a value, try unpacking it as a range ("1-23" for "1 to
        -- However, "-1-2/3" is a negative fraction (-1 $\frac{2}{3}$ ), so it must be extracted first.
        -- Do not unpack a parameter if it is like "3-1/2" which is sometimes incorrectly
        -- used instead of "3+1/2" (and which should not be interpreted as "3 to  $\frac{1}{2}$ ").
        -- Unpacked items are inserted into the parms table.

```

```

-- The tail recursion allows combinations like "1x2 to 3x4".
local valstr = strip(parms[i]) -- trim so any '-' as a negative sign will be at :
local success, result = extract_number(parms, valstr, i > 1)
if not success and valstr and i < 20 then -- check i to limit abuse
    local lhs, sep, rhs = valstr:match('^(%S+)%s+(%S+)%s+(%S.*)')
    if lhs and not (sep == '-' and rhs:match('/')) then
        if sep:find('%d') then
            return success, result -- to reject {{convert|1 234 567|n
        end
        parms[i] = rhs
        table.insert(parms, i, sep)
        table.insert(parms, i, lhs)
        return extractor(i)
    end
    if not valstr:match('%-.*\/') then
        for _, sep in ipairs(text_code.ranges.words) do
            local start, stop = valstr:find(sep, 2, true) -- start at
            if start then
                parms[i] = valstr:sub(stop + 1)
                table.insert(parms, i, sep)
                table.insert(parms, i, valstr:sub(1, start - 1))
                return extractor(i)
            end
        end
    end
end
return success, result
end
local i = 1
local is_change
while true do
    local success, info = extractor(i) -- need to set parms.opt_nocomma before callin
    if not success then return false, info end
    i = i + 1
    if is_change then
        info.is_change = true -- value is after "±" and so is a change (significa
        is_change = nil
    end
    valinfo:add(info)
    local range_item = get_range(strip(parms[i]))
    if not range_item then
        break
    end
    i = i + 1
    range:add(range_item)
    if type(range_item) == 'table' then
        -- For range "x", if append unit to some values, append it to all.
        parms.in_range_x = parms.in_range_x or range_item.in_range_x
        parms.out_range_x = parms.out_range_x or range_item.out_range_x
        parms.abbr_range_x = parms.abbr_range_x or range_item.abbr_range_x
        is_change = range_item.is_range_change
    end
end

```

```

        end
    end
end
if range.n > 0 then
    if range.n > 30 then -- limit abuse, although 4 is a more likely upper limit
        return false, { 'cvt_invalid_num' } -- misleading message but it will do
    end
    parms.range = range
elseif had_nocomma then
    return false, { 'cvt_unknown', parm2 }
end
return true, valinfo, i
end

local function simple_get_values(parms)
    -- If input is like "{{convert|valid_value|valid_unit|...}}",
    -- return true, i, in_unit, in_unit_table
    -- i = index in parms of what follows valid_unit, if anything.
    -- The valid_value is not negative and does not use a fraction, and
    -- no options requiring further processing of the input are used.
    -- Otherwise, return nothing or return false, parm1 for caller to interpret.
    -- Testing shows this function is successful for 96% of converts in articles,
    -- and that on average it speeds up converts by 8%.
    local clean = to_en(strip(parms[1] or ''), parms)
    if parms.opt_ri or parms.opt_spell_in or #clean > 10 or not clean:match('^[@-9.]+$') then
        return false, clean
    end
    local value = tonumber(clean)
    if not value then return end
    local info = {
        value = value,
        altvalue = value,
        singular = (value == 1),
        clean = clean,
        show = with_separator(parms, clean),
    }
    local in_unit = strip(parms[2])
    local success, in_unit_table = lookup(parms, in_unit, 'no_combination')
    if not success then return end
    in_unit_table.valinfo = { info }
    return true, 3, in_unit, in_unit_table
end

local function wikidata_call(parms, operation, ...)
    -- Return true, s where s is the result of a Wikidata operation,
    -- or return false, t where t is an error message table.
    local function worker(...)
        wikidata_code = wikidata_code or require(wikidata_module)
        wikidata_data = wikidata_data or mw.loadData(wikidata_data_module)
        return wikidata_code[operation](wikidata_data, ...)
    end
    local success, status, result = pcall(worker, ...)

```

```

if success then
    return status, result
end
if parms.opt_sortable_debug then
    -- Use debug=yes to crash if an error while accessing Wikidata.
    error('Error accessing Wikidata: ' .. status, 0)
end
return false, { 'cvt_wd_fail' }
end

local function get_parms(parms, args)
    -- If successful, update parms and return true, unit where
    -- parms is a table of all arguments passed to the template
    -- converted to named arguments, and
    -- unit is the input unit table;
    -- or return false, t where t is an error message table.
    -- For special processing (not a convert), can also return
    -- true, wikitext where wikitext is the final result.
    -- The returned input unit table may be for a fake unit using the specified
    -- unit code as the symbol and name, and with bad_mcode = message code table.
    -- MediaWiki removes leading and trailing whitespace from the values of
    -- named arguments. However, the values of numbered arguments include any
    -- whitespace entered in the template, and whitespace is used by some
    -- parameters (example: the numbered parameters associated with "disp=x").
    local kv_pairs = {} -- table of input key:value pairs where key is a name; needed because
    for k, v in pairs(args) do
        if type(k) == 'number' or k == 'test' then -- parameter "test" is reserved for te
            parms[k] = v
        else
            kv_pairs[k] = v
        end
    end
end
if parms.test == 'wikidata' then
    local ulookup = function (ucode)
        -- Use empty table for parms so it does not accumulate results when used i
        return lookup({}, ucode, 'no_combination')
    end
    return wikidata_call(parms, '_listunits', ulookup)
end
local success, msg = translate_parms(parms, kv_pairs)
if not success then return false, msg end
if parms.input then
    success, msg = wikidata_call(parms, '_adjustparameters', parms, 1)
    if not success then return false, msg end
end
local success, i, in_unit, in_unit_table = simple_get_values(parms)
if not success then
    if type(i) == 'string' and i:match('^NNN+$') then
        -- Some infoboxes have examples like {{convert|NNN|m}} (3 or more "N").
        -- Output an empty string for these.
        return false, { 'cvt_no_output' }
    end
end

```

```

end
local valinfo
success, valinfo, i = get_values(parms)
if not success then return false, valinfo end
in_unit = strip(parms[i])
i = i + 1
success, in_unit_table = lookup(parms, in_unit, 'no_combination')
if not success then
    in_unit = in_unit or ''
    if parms.opt_ignore_error then -- display given unit code with no error (
        in_unit_table = '' -- suppress error message and prevent processi
    end
    in_unit_table = setmetatable({
        symbol = in_unit, name2 = in_unit, utype = in_unit,
        scale = 1, default = '', defkey = '', linkey = '',
        bad_mcode = in_unit_table }, unit_mt)
end
in_unit_table.valinfo = valinfo
end
if parms.test == 'msg' then
    -- Am testing the messages produced when no output unit is specified, and
    -- the input unit has a missing or invalid default.
    -- Set two units for testing that.
    -- LATER: Remove this code.
    if in_unit == 'chain' then
        in_unit_table.default = nil -- no default
    elseif in_unit == 'rd' then
        in_unit_table.default = "ft!X!m" -- an invalid expression
    end
end
end
in_unit_table.inout = 'in' -- this is an input unit
if not parms.range then
    local success, inext, composite_unit = get_composite(parms, i, in_unit_table)
    if not success then return false, inext end
    if composite_unit then
        in_unit_table = composite_unit
        i = inext
    end
end
end
if in_unit_table.builtin == 'mach' then
    -- As with old template, a number following Mach as the input unit is the altitude
    -- That is deprecated: should use altitude_ft=NUMBER or altitude_m=NUMBER.
    local success, info
    success = tonumber(parms[i]) -- this will often work and will give correct result
    if success then
        info = { value = success }
    else
        success, info = extract_number(parms, parms[i], false, true)
    end
end
if success then
    i = i + 1

```

```

        in_unit_table.altitude = info.value
    end
end
local word = strip(parms[i])
i = i + 1
local precision, is_bad_precision
local function set_precision(text)
    local number, is_integer = get_number(text)
    if number then
        if is_integer then
            precision = number
        else
            precision = text
            is_bad_precision = true
        end
    end
    return true -- text was used for precision, good or bad
end
end
if word and not set_precision(word) then
    parms.out_unit = parms.out_unit or word
    if set_precision(strip(parms[i])) then
        i = i + 1
    end
end
if parms.opt_adj_mid then
    word = parms[i]
    i = i + 1
    if word then -- mid-text words
        if word:sub(1, 1) == '-' then
            parms.mid = word
        else
            parms.mid = ' ' .. word
        end
    end
end
end
if parms.opt_one_preunit then
    parms[parms.opt_flip and 'preunit2' or 'preunit1'] = preunits(1, parms[i])
    i = i + 1
end
if parms.disp == 'x' then
    -- Following is reasonably compatible with the old template.
    local first = parms[i] or ''
    local second = parms[i+1] or ''
    i = i + 2
    if strip(first) == '' then -- user can enter '&#32;' rather than ' ' to avoid the
        first = ' [&nbsp;]' .. first
        second = '&nbsp;]' .. second
    end
    parms.joins = { first, second }
elseif parms.opt_two_preunits then
    local p1, p2 = preunits(2, parms[i], parms[i+1])

```

```

    i = i + 2
    if parms.preunit1 then
        -- To simplify documentation, allow unlikely use of adj=pre with disp=pre
        -- (however, an output unit must be specified with adj=pre and with disp=pre)
        parms.preunit1 = parms.preunit1 .. p1
        parms.preunit2 = p2
    else
        parms.preunit1, parms.preunit2 = p1, p2
    end
end
if precision == nil then
    if set_precision(strip(parms[i])) then
        i = i + 1
    end
end
if is_bad_precision then
    add_warning(parms, 1, 'cvt_bad_prec', precision)
else
    parms.precision = precision
end
for j = i, i + 3 do
    local parm = parms[j] -- warn if find a non-empty extraneous parameter
    if parm and parm:match('%S') then
        add_warning(parms, 1, 'cvt_unknown_option', parm)
        break
    end
end
return true, in_unit_table
end

```

```

local function record_default_precision(parms, out_current, precision)
    -- If necessary, adjust parameters and return a possibly adjusted precision.
    -- When converting a range of values where a default precision is required,
    -- that default is calculated for each value because the result sometimes
    -- depends on the precise input and output values. This function may cause
    -- the entire convert process to be repeated in order to ensure that the
    -- same default precision is used for each individual convert.
    -- If that were not done, a range like 1000 to 1000.4 may give poor results
    -- because the first output could be heavily rounded, while the second is not.
    -- For range 1000.4 to 1000, this function can give the second convert the
    -- same default precision that was used for the first.
    if not parms.opt_round_each then
        local maxdef = out_current.max_default_precision
        if maxdef then
            if maxdef < precision then
                parms.do_convert_again = true
                out_current.max_default_precision = precision
            else
                precision = out_current.max_default_precision
            end
        end
    else

```

```

        out_current.max_default_precision = precision
    end
end
return precision
end

local function default_precision(parms, inval, inclean, denominator, outvalue, in_current, out_current)
    -- Return a default value for precision (an integer like 2, 0, -2).
    -- If denominator is not nil, it is the value of the denominator in inclean.
    -- Code follows procedures used in old template.
    local fudge = 1e-14 -- {{Order of magnitude}} adds this, so we do too
    local prec, minprec, adjust
    local subunit_ignore_trailing_zero
    local subunit_more_precision -- kludge for "in" used in input like "|2|ft|6|in"
    local composite = in_current.composite
    if composite then
        subunit_ignore_trailing_zero = true -- input "|2|st|10|lb" has precision 0, not .
        if composite[#composite].exception == 'subunit_more_precision' then
            subunit_more_precision = true -- do not use standard precision with input
        end
    end
end

if denominator and denominator > 0 then
    prec = math.max(log10(denominator), 1)
else
    -- Count digits after decimal mark, handling cases like '12.345e6'.
    local exponent
    local integer, dot, decimals, expstr = inclean:match('^(%d*)(%?)(%d*)(.*)')
    local e = expstr:sub(1, 1)
    if e == 'e' or e == 'E' then
        exponent = tonumber(expstr:sub(2))
    end
    if dot == '.' then
        prec = subunit_ignore_trailing_zero and 0 or -integer:match('^0*$'):len()
    else
        prec = #decimals
    end
    if exponent then
        -- So '1230' and '1.23e3' both give prec = -1, and '0.00123' and '1.23e-3'
        prec = prec - exponent
    end
end

if in_current.istemperature and out_current.istemperature then
    -- Converting between common temperatures (°C, °F, °R, K); not keVT.
    -- Kelvin value can be almost zero, or small but negative due to precision problem
    -- Also, an input value like -300 C (below absolute zero) gives negative kelvins.
    -- Calculate minimum precision from absolute value.
    adjust = 0
    local kelvin = abs((inval - in_current.offset) * in_current.scale)
    if kelvin < 1e-8 then -- assume nonzero due to input or calculation precision problem
        minprec = 2
    else

```

```

        minprec = 2 - floor(log10(kelvin) + fudge) -- 3 sigfigs in kelvin
    end
else
    if invalue == 0 or outvalue <= 0 then
        -- We are never called with a negative outvalue, but it might be zero.
        -- This is special-cased to avoid calculation exceptions.
        return record_default_precision(parms, out_current, 0)
    end
    if out_current.exception == 'integer_more_precision' and floor(invalue) == invalue
        -- With certain output units that sometimes give poor results
        -- with default rounding, use more precision when the input
        -- value is equal to an integer. An example of a poor result
        -- is when input 50 gives a smaller output than input 49.5.
        -- Experiment shows this helps, but it does not eliminate all
        -- surprises because it is not clear whether "50" should be
        -- interpreted as "from 45 to 55" or "from 49.5 to 50.5".
        adjust = -log10(in_current.scale)
    elseif subunit_more_precision then
        -- Conversion like "{{convert|6|ft|1|in|cm}}" (where subunit is "in")
        -- has a non-standard adjust value, to give more output precision.
        adjust = log10(out_current.scale) + 2
    else
        adjust = log10(abs(invalue / outvalue))
    end
    adjust = adjust + log10(2)
    -- Ensure that the output has at least two significant figures.
    minprec = 1 - floor(log10(outvalue) + fudge)
end
if extra then
    adjust = extra.adjust or adjust
    minprec = extra.minprec or minprec
end
return record_default_precision(parms, out_current, math.max(floor(prec + adjust), minprec)
end

local function convert(parms, invalue, info, in_current, out_current)
    -- Convert given input value from one unit to another.
    -- Return output_value (a number) if a simple convert, or
    -- return f, t where
    --   f = true, t = table of information with results, or
    --   f = false, t = error message table.
    local inscale = in_current.scale
    local outscale = out_current.scale
    if not in_current.iscomplex and not out_current.iscomplex then
        return invalue * (inscale / outscale) -- minimize overhead for most common case
    end
    if in_current.invert or out_current.invert then
        -- Inverted units, such as inverse length, inverse time, or
        -- fuel efficiency. Built-in units do not have invert set.
        if (in_current.invert or 1) * (out_current.invert or 1) < 0 then
            return 1 / (invalue * inscale * outscale)
        end
    end
end

```

```

    end
    return invalue * (inscale / outscale)
elseif in_current.offset then
    -- Temperature (there are no built-ins for this type of unit).
    if info.is_change then
        return invalue * (inscale / outscale)
    end
    return (invalue - in_current.offset) * (inscale / outscale) + out_current.offset
else
    -- Built-in unit.
    local in_builtin = in_current.builtin
    local out_builtin = out_current.builtin
    if in_builtin and out_builtin then
        if in_builtin == out_builtin then
            return invalue
        end
        -- There are no cases (yet) where need to convert from one
        -- built-in unit to another, so this should never occur.
        return false, { 'cvt_bug_convert' }
    end
    if in_builtin == 'mach' or out_builtin == 'mach' then
        -- Should check that only one altitude is given but am planning to remove
        -- in_current.altitude (which can only occur when Mach is the input unit),
        -- and out_current.altitude cannot occur.
        local alt = parms.altitude_ft or in_current.altitude
        if not alt and parms.altitude_m then
            alt = parms.altitude_m / 0.3048 -- 1 ft = 0.3048 m
        end
        local spd = speed_of_sound(alt)
        if in_builtin == 'mach' then
            inscale = spd
            return invalue * (inscale / outscale)
        end
        outscale = spd
        local adjust = 0.1 / inscale
        return true, {
            outvalue = invalue * (inscale / outscale),
            adjust = log10(adjust) + log10(2),
        }
    elseif in_builtin == 'hand' then
        -- 1 hand = 4 inches; 1.2 hands = 6 inches.
        -- Decimals of a hand are only defined for the first digit, and
        -- the first fractional digit should be a number of inches (1, 2 or 3).
        -- However, this code interprets the entire fractional part as the number
        -- of inches / 10 (so 1.75 inches would be 0.175 hands).
        -- A value like 12.3 hands is exactly 12*4 + 3 inches; base default preci:
        local integer, fracpart = math.modf(invalue)
        local inch_value = 4 * integer + 10 * fracpart -- equivalent number of in
        local factor = inscale / outscale
        if factor == 4 then
            -- Am converting to inches: show exact result, and use "inches" nc

```

```

        if parms.abbr_org == nil then
            out_current.username = true
        end
        local show = format('%g', abs(inch_value)) -- show and clean are
        if not show:find('e', 1, true) then
            return true, {
                invalue = inch_value,
                outvalue = inch_value,
                clean = show,
                show = show,
            }
        end
    end
    local outvalue = (integer + 2.5 * fracpart) * factor
    local fracstr = info.clean:match('%.(.*)') or ''
    local fmt
    if fracstr == '' then
        fmt = '%.0f'
    else
        fmt = '%.' .. format('%d', #fracstr - 1) .. 'f'
    end
    return true, {
        invalue = inch_value,
        clean = format(fmt, inch_value),
        outvalue = outvalue,
        minprec = 0,
    }
end
end
return false, { 'cvt_bug_convert' } -- should never occur
end

local function user_style(parms, i)
    -- Return text for a user-specified style for a table cell, or '' if none,
    -- given i = 1 (input style) or 2 (output style).
    local style = parms[(i == 1) and 'stylein' or 'styleout']
    if style then
        style = style:gsub('""', '')
        if style ~= '' then
            if style:sub(-1) ~= ';' then
                style = style .. ';'
            end
            return style
        end
    end
end

end
return ''
end

local function make_table_or_sort(parms, invalue, info, in_current, scaled_top)
    -- Set options to handle output for a table or a sort key, or both.
    -- The text sort key is based on the value resulting from converting

```

```

-- the input to a fake base unit with scale = 1, and other properties
-- required for a conversion derived from the input unit.
-- For other modules, return the sort key in a hidden span element, and
-- the scaled value used to generate the sort key.
-- If scaled_top is set, it is the scaled value of the numerator of a per unit
-- to be combined with this unit (the denominator) to make the sort key.
-- Scaling only works with units that convert with a factor (not temperature).
local sortkey, scaled_value
if parms.opt_sortable_on then
    local base = { -- a fake unit with enough fields for a valid convert
        scale = 1,
        invert = in_current.invert and 1,
        iscomplex = in_current.iscomplex,
        offset = in_current.offset and 0,
    }
    local outvalue, extra = convert(parms, invalue, info, in_current, base)
    if extra then
        outvalue = extra.outvalue
    end
    if in_current.istemperature then
        -- Have converted to kelvin; assume numbers close to zero have a
        -- rounding error and should be zero.
        if abs(outvalue) < 1e-12 then
            outvalue = 0
        end
    end
    end
    if scaled_top and outvalue ~= 0 then
        outvalue = scaled_top / outvalue
    end
    scaled_value = outvalue
    if not valid_number(outvalue) then
        if outvalue < 0 then
            sortkey = '10000000000000000000'
        else
            sortkey = '90000000000000000000'
        end
    elseif outvalue == 0 then
        sortkey = '50000000000000000000'
    else
        local mag = floor(log10(abs(outvalue))) + 1e-14
        local prefix
        if outvalue > 0 then
            prefix = 7000 + mag
        else
            prefix = 2999 - mag
            outvalue = outvalue + 10^(mag+1)
        end
        sortkey = format('%d', prefix) .. format('%015.0f', floor(outvalue * 10^(1
    end
end
local sortspan

```

```

if sortkey and not parms.table_align then
    sortspan = parms.opt_sortable_debug and
        '<span data-sort-value="' .. sortkey .. '♠"><span style="border:1px solid'
        '<span data-sort-value="' .. sortkey .. '♠"></span>'
    parms.join_before = sortspan
end
if parms.table_align then
    local sort
    if sortkey then
        sort = ' data-sort-value="' .. sortkey .. ''
        if parms.opt_sortable_debug then
            parms.join_before = '<span style="border:1px solid">' .. sortkey .
        end
    else
        sort = ''
    end
    local style = 'style="text-align:' .. parms.table_align .. ';'
    local joins = {}
    for i = 1, 2 do
        joins[i] = (i == 1 and '' or '\n|') .. style .. user_style(parms, i) .. ''
    end
    parms.table_joins = joins
end
return sortspan, scaled_value
end

local cvt_to_hand

local function cvtround(parms, info, in_current, out_current)
    -- Return true, t where t is a table with the conversion results; fields:
    -- show = rounded, formatted string with the result of converting value in info,
    -- using the rounding specified in parms.
    -- singular = true if result (after rounding and ignoring any negative sign)
    -- is "1", or like "1.00", or is a fraction with value < 1;
    -- (and more fields shown below, and a calculated 'absvalue' field).
    -- or return false, t where t is an error message table.
    -- Input info.clean uses en digits (it has been translated, if necessary).
    -- Output show uses en or non-en digits as appropriate, or can be spelled.
    if out_current.builtin == 'hand' then
        return cvt_to_hand(parms, info, in_current, out_current)
    end
    local inval = in_current.builtin == 'hand' and info.altvalue or info.value
    local outval, extra = convert(parms, inval, info, in_current, out_current)
    if parms.need_table_or_sort then
        parms.need_table_or_sort = nil -- process using first input value only
        make_table_or_sort(parms, inval, info, in_current)
    end
    if extra then
        if not outval then return false, extra end
        inval = extra.inval or inval
        outval = extra.outval
    end
end

```

```

end
if not valid_number(outvalue) then
    return false, { 'cvt_invalid_num' }
end
local isnegative
if outvalue < 0 then
    isnegative = true
    outvalue = -outvalue
end
local precision, show, exponent
local denominator = out_current.frac
if denominator then
    show = fraction_table(outvalue, denominator)
else
    precision = parms.precision
    if not precision then
        if parms.sigfig then
            show, exponent = make_sigfig(outvalue, parms.sigfig)
        elseif parms.opt_round then
            local n = parms.opt_round
            if n == 0.5 then
                local integer, fracpart = math.modf(floor(2 * outvalue + (
                if fracpart == 0 then
                    show = format('%.0f', integer)
                else
                    show = format('%.1f', integer + fracpart)
                end
            else
                show = format('%.0f', floor((outvalue / n) + 0.5) * n)
            end
        elseif in_current.builtin == 'mach' then
            local sigfig = info.clean:gsub('^[0.]+', ''):gsub('%.', ''):len()
            show, exponent = make_sigfig(outvalue, sigfig)
        else
            local inclean = info.clean
            if extra then
                inclean = extra.clean or inclean
                show = extra.show
            end
            if not show then
                precision = default_precision(parms, invalue, inclean, in
            end
        end
    end
end
if precision then
    if precision >= 0 then
        local fudge
        if precision <= 8 then
            -- Add a fudge to handle common cases of bad rounding due to inabi
            -- to precisely represent some values. This makes the following wc

```

```

        -- {{convert|-100.1|C|K}} and {{convert|5555000|um|m|2}}.
        -- Old template uses #expr round, which invokes PHP round().
        -- LATER: Investigate how PHP round() works.
        fudge = 2e-14
    else
        fudge = 0
    end
    local fmt = '%.' .. format('%d', precision) .. 'f'
    local success
    success, show = pcall(format, fmt, outvalue + fudge)
    if not success then
        return false, { 'cvt_big_prec', tostring(precision) }
    end
else
    precision = -precision -- #digits to zero (in addition to any digits after)
    local shift = 10 ^ precision
    show = format('%.' .. precision .. 'f', outvalue/shift)
    if show ~= '0' then
        exponent = #show + precision
    end
end
end
local t = format_number(parms, show, exponent, isnegative)
if type(show) == 'string' then
    -- Set singular using match because on some systems 0.9999999999999999 is 1.0.
    if exponent then
        t.singular = (exponent == 1 and show:match('^10*$'))
    else
        t.singular = (show == '1' or show:match('^1%.0*$'))
    end
else
    t.fraction_table = show
    t.singular = (outvalue <= 1) -- cannot have 'fraction == 1', but if it were possible
end
t.raw_absvalue = outvalue -- absolute value before rounding
return true, setmetatable(t, {
    __index = function (self, key)
        if key == 'absvalue' then
            -- Calculate absolute value after rounding, if needed.
            local clean, exponent = rawget(self, 'clean'), rawget(self, 'exponent')
            local value = tonumber(clean) -- absolute value (any negative sign)
            if exponent then
                value = value * 10^exponent
            end
            rawset(self, key, value)
            return value
        end
    end })
end

function cvt_to_hand(parms, info, in_current, out_current)

```

```

-- Convert input to hands, inches.
-- Return true, t where t is a table with the conversion results;
-- or return false, t where t is an error message table.
if parms.abbrev == nil then
    out_current.username = true -- default is to show name not symbol
end
local precision = parms.precision
local frac = out_current.frac
if not frac and precision and precision > 1 then
    frac = (precision == 2) and 2 or 4
end
local out_next = out_current.out_next
if out_next then
    -- Use magic knowledge to determine whether the next unit is inches without requiring
    -- The following ensures that when the output combination "hand in" is used, the i
    -- value is rounded to match the hands value. Also, displaying say "61½" instead of
    -- is better as 61.5 implies the value is not 61.4.
    if out_next.exception == 'subunit_more_precision' then
        out_next.frac = frac
    end
end
end
-- Convert to inches; calculate hands from that.
local dummy_unit_table = { scale = out_current.scale / 4, frac = frac }
local success, outinfo = cvtround(parms, info, in_current, dummy_unit_table)
if not success then return false, outinfo end
local tfrac = outinfo.fraction_table
local inches = outinfo.raw_absvalue
if tfrac then
    inches = floor(inches) -- integer part only; fraction added later
else
    inches = floor(inches + 0.5) -- a hands measurement never shows decimals of an in
end
local hands, inches = divide(inches, 4)
outinfo.absvalue = hands + inches/4 -- supposed to be the absolute rounded value, but thi
local inchstr = tostring(inches) -- '0', '1', '2' or '3'
if precision and precision <= 0 then -- using negative or 0 for precision rounds to near
    hands = floor(outinfo.raw_absvalue/4 + 0.5)
    inchstr = ''
elseif tfrac then
    -- Always show an integer before fraction (like "15.0½") because "15½" means 15-ar
    inchstr = numdot .. format_fraction(parms, 'out', false, inchstr, tfrac.numstr, ti
else
    inchstr = numdot .. from_en(inchstr)
end
end
outinfo.show = outinfo.sign .. with_separator(parms, format('%.0f', hands)) .. inchstr
return true, outinfo
end

local function evaluate_condition(value, condition)
    -- Return true or false from applying a conditional expression to value,
    -- or throw an error if invalid.

```

```

-- A very limited set of expressions is supported:
--   v < 9
--   v * 9 < 9
-- where
--   'v' is replaced with value
--   9 is any number (as defined by Lua tonumber)
--   only en digits are accepted
--   '<' can also be '<=' or '>' or '>='
-- In addition, the following form is supported:
--   LHS and RHS
-- where
--   LHS, RHS = any of above expressions.
local function compare(value, text)
    local arithop, factor, compop, limit = text:match('^%s*v%s*([*]?)(.-)([<>]=?)(.*)$')
    if arithop == nil then
        error('Invalid default expression', 0)
    elseif arithop == '*' then
        factor = tonumber(factor)
        if factor == nil then
            error('Invalid default expression', 0)
        end
        value = value * factor
    end
    limit = tonumber(limit)
    if limit == nil then
        error('Invalid default expression', 0)
    end
    if compop == '<' then
        return value < limit
    elseif compop == '<=' then
        return value <= limit
    elseif compop == '>' then
        return value > limit
    elseif compop == '>=' then
        return value >= limit
    end
    error('Invalid default expression', 0) -- should not occur
end
local lhs, rhs = condition:match('^(.-%W)and(%W.*)')
if lhs == nil then
    return compare(value, condition)
end
return compare(value, lhs) and compare(value, rhs)
end

local function get_default(value, unit_table)
    -- Return true, s where s = name of unit's default output unit,
    -- or return false, t where t is an error message table.
    -- Some units have a default that depends on the input value
    -- (the first value if a range of values is used).
    -- If '!' is in the default, the first bang-delimited field is an

```

```

-- expression that uses 'v' to represent the input value.
-- Example: 'v < 120 ! small ! big ! suffix' (suffix is optional)
-- evaluates 'v < 120' as a boolean with result
-- 'smallsuffix' if (value < 120), or 'bigsuffix' otherwise.
-- Input must use en digits and '.' decimal mark.
local default = data_code.default_exceptions[unit_table.defkey or unit_table.symbol] or ur
if not default then
    local per = unit_table.per
    if per then
        local function a_default(v, u)
            local success, ucode = get_default(v, u)
            if not success then
                return '?' -- an unlikely error has occurred; will cause
            end
            -- Attempt to use only the first unit if a combination or output r
            -- This is not bulletproof but should work for most cases.
            -- Where it does not work, the convert will need to specify the wa
            local t = all_units[ucode]
            if t then
                local combo = t.combination
                if combo then
                    -- For a multiple like ftin, the "first" unit (ft)
                    local i = t.multiple and table_len(combo) or 1
                    ucode = combo[i]
                end
            end
            else
                -- Try for an automatically generated combination.
                local item = ucode:match('^(-)%+') or ucode:match('^(%S+)')
                if all_units[item] then
                    return item
                end
            end
            end
            return ucode
        end
        local unit1, unit2 = per[1], per[2]
        local def1 = (unit1 and a_default(value, unit1) or unit_table.vprefix or '
        local def2 = a_default(1, unit2) -- 1 because per unit of denominator
        return true, def1 .. '/' .. def2
    end
    return false, { 'cvt_no_default', unit_table.symbol }
end
if default:find('!', 1, true) == nil then
    return true, default
end
local t = split(default, '!')
if #t == 3 or #t == 4 then
    local success, result = pcall(evaluate_condition, value, t[1])
    if success then
        default = result and t[2] or t[3]
        if #t == 4 then
            default = default .. t[4]
        end
    end
end

```

```

                end
                return true, default
            end
        end
    end
    return false, { 'cvt_bad_default', unit_table.symbol }
end

local linked_pages -- to record linked pages so will not link to the same page more than once

local function unlink(unit_table)
    -- Forget that the given unit has previously been linked (if it has).
    -- That is needed when processing a range of inputs or outputs when an id
    -- for the first range value may have been evaluated, but only an id for
    -- the last value is displayed, and that id may need to be linked.
    linked_pages[unit_table.unitcode or unit_table] = nil
end

local function make_link(link, id, unit_table)
    -- Return wikilink "[[link|id]]", possibly abbreviated as in examples:
    -- [[Mile|mile]] --> [[mile]]
    -- [[Mile|miles]] --> [[mile]]s
    -- However, just id is returned if:
    -- * no link given (so caller does not need to check if a link was defined); or
    -- * link has previously been used during the current convert (to avoid overlinking).
    local link_key
    if unit_table then
        link_key = unit_table.unitcode or unit_table
    else
        link_key = link
    end
    end
    if not link or link == '' or linked_pages[link_key] then
        return id
    end
    linked_pages[link_key] = true
    -- Following only works for language en, but it should be safe on other wikis,
    -- and overhead of doing it generally does not seem worthwhile.
    local l = link:sub(1, 1):lower() .. link:sub(2)
    if link == id or l == id then
        return '[' .. id .. ']'
    elseif link .. 's' == id or l .. 's' == id then
        return '[' .. id:sub(1, -2) .. ']'s'
    else
        return '[' .. link .. '|' .. id .. ']'
    end
end

local function variable_name(clean, unit_table)
    -- For slwiki, a unit name depends on the value.
    -- Parameter clean is the unsigned rounded value in en digits, as a string.
    -- Value          Source      Example for "m"
    -- integer 1:    name1       meter (also is the name of the unit)

```

```

-- integer 2:      var{1}   metra
-- integer 3 and 4: var{2}   metri
-- integer else:   var{3}   metrov (0 and 5 or more)
-- real/fraction:  var{4}   metra
-- var{i} means the i'th field in unit_table.varname if it exists and has
-- an i'th field, otherwise name2.
-- Fields are separated with "!" and are not empty.
-- A field for a unit using an SI prefix has the prefix name inserted,
-- replacing '#' if found, or before the field otherwise.
local vname
if clean == '1' then
    vname = unit_table.name1
elseif unit_table.varname then
    local i
    if clean == '2' then
        i = 1
    elseif clean == '3' or clean == '4' then
        i = 2
    elseif clean:find('.', 1, true) then
        i = 4
    else
        i = 3
    end
    if i > 1 and varname == 'pl' then
        i = i - 1
    end
    vname = split(unit_table.varname, '!')[i]
end
if vname then
    local si_name = rawget(unit_table, 'si_name') or ''
    local pos = vname:find('#', 1, true)
    if pos then
        vname = vname:sub(1, pos - 1) .. si_name .. vname:sub(pos + 1)
    else
        vname = si_name .. vname
    end
    return vname
end
return unit_table.name2
end

local function linked_id(parms, unit_table, key_id, want_link, clean)
-- Return final unit id (symbol or name), optionally with a wikilink,
-- and update unit_table.sep if required.
-- key_id is one of: 'symbol', 'sym_us', 'name1', 'name1_us', 'name2', 'name2_us'.
local abbr_on = (key_id == 'symbol' or key_id == 'sym_us')
if abbr_on and want_link then
    local symlink = rawget(unit_table, 'symlink')
    if symlink then
        return symlink -- for exceptions that have the linked symbol built-in
    end
end

```

```

end
local multiplier = rawget(unit_table, 'multiplier')
local per = unit_table.per
if per then
    local paren1, paren2 = '', '' -- possible parentheses around bottom unit
    local unit1 = per[1] -- top unit_table, or nil
    local unit2 = per[2] -- bottom unit_table
    if abbr_on then
        if not unit1 then
            unit_table.sep = '' -- no separator in "$2/acre"
        end
        if not want_link then
            local symbol = unit_table.symbol_raw
            if symbol then
                return symbol -- for exceptions that have the symbol built
            end
        end
        if (unit2.symbol):find('.', 1, true) then
            paren1, paren2 = '(', ')'
        end
    end
    local key_id2 -- unit2 is always singular
    if key_id == 'name2' then
        key_id2 = 'name1'
    elseif key_id == 'name2_us' then
        key_id2 = 'name1_us'
    else
        key_id2 = key_id
    end
    local result
    if abbr_on then
        result = '/'
    elseif omitsep then
        result = per_word
    elseif unit1 then
        result = ' ' .. per_word .. ' '
    else
        result = per_word .. ' '
    end
    if want_link and unit_table.link then
        if abbr_on or not varname then
            result = (unit1 and linked_id(parms, unit1, key_id, false, clean)
        else
            result = (unit1 and variable_name(clean, unit1) or '') .. result
        end
        if omit_separator(result) then
            unit_table.sep = ''
        end
        return make_link(unit_table.link, result, unit_table)
    end
    if unit1 then

```

```

        result = linked_id(parms, unit1, key_id, want_link, clean) .. result
        if unit1.sep then
            unit_table.sep = unit1.sep
        end
    elseif omitsep then
        unit_table.sep = ''
    end
    return result .. paren1 .. linked_id(parms, unit2, key_id2, want_link, '1') .. pa
end
if multiplier then
    -- A multiplier (like "100" in "100km") forces the unit to be plural.
    multiplier = from_en(multiplier)
    if not omitsep then
        multiplier = multiplier .. (abbr_on and '&nbsp;' or ' ')
    end
    if not abbr_on then
        if key_id == 'name1' then
            key_id = 'name2'
        elseif key_id == 'name1_us' then
            key_id = 'name2_us'
        end
    end
end
else
    multiplier = ''
end
local id = unit_table.fixed_name or ((varname and not abbr_on) and variable_name(clean, ur
if omit_separator(id) then
    unit_table.sep = ''
end
if want_link then
    local link = data_code.link_exceptions[unit_table.linkey or unit_table.symbol] or
    if link then
        local before = ''
        local i = unit_table.customary
        if i == 1 and parms.opt_sp_us then
            i = 2 -- show "U.S." not "US"
        end
        if i == 3 and abbr_on then
            i = 4 -- abbreviate "imperial" to "imp"
        end
        local customary = text_code.customary_units[i]
        if customary then
            -- LATER: This works for language en only, but it's esoteric so ig
            local pertext
            if id:sub(1, 1) == '/' then
                -- Want unit "/USgal" to display as "/U.S. gal", not "U.S.
                pertext = '/'
                id = id:sub(2)
            elseif id:sub(1, 4) == 'per ' then
                -- Similarly want "per U.S. gallon", not "U.S. per gallon"
                pertext = 'per '
            end
        end
    end
end

```

```

        id = id:sub(5)
    else
        pertext = ''
    end
    end
    -- Omit any "US"/"U.S."/"imp"/"imperial" from start of id since tl
    local removes = (i < 3) and { 'US&nbsp;', 'US ', 'U.S.&nbsp;', 'U.
    for _, prefix in ipairs(removes) do
        local plen = #prefix
        if id:sub(1, plen) == prefix then
            id = id:sub(plen + 1)
            break
        end
    end
    end
    before = pertext .. make_link(customary.link, customary[1]) .. ' '
end
id = before .. make_link(link, id, unit_table)
end
end
return multiplier .. id
end

```

```

local function make_id(parms, which, unit_table)
    -- Return id, f where
    -- id = unit name or symbol, possibly modified
    -- f = true if id is a name, or false if id is a symbol
    -- using the value for index 'which', and for 'in' or 'out' (unit_table.inout).
    -- Result is '' if no symbol/name is to be used.
    -- In addition, set unit_table.sep = ' ' or '&nbsp;' or ''
    -- (the separator that caller will normally insert before the id).
    if parms.opt_values then
        unit_table.sep = ''
        return ''
    end
    local inout = unit_table.inout
    local info = unit_table.valinfo[which]
    local abbr_org = parms.abbr_org
    local adjectival = parms.opt_adjectival
    local lk = parms.lk
    local want_link = (lk == 'on' or lk == inout)
    local username = unit_table.username
    local singular = info.singular
    local want_name
    if username then
        want_name = true
    else
        if abbr_org == nil then
            if parms.wantname then
                want_name = true
            end
            if unit_table.usesymbol then
                want_name = false
            end
        end
    end
end

```

```

        end
    end
    if want_name == nil then
        local abbr = parms.abbr
        if abbr == 'on' or abbr == inout or (abbr == 'mos' and inout == 'out') then
            want_name = false
        else
            want_name = true
        end
    end
end
local key
if want_name then
    if lk == nil and unit_table.builtin == 'hand' then
        want_link = true
    end
    if parms.opt_use_nbsp then
        unit_table.sep = '&nbsp;';
    else
        unit_table.sep = ' '
    end
    if parms.opt_singular then
        local value
        if inout == 'in' then
            value = info.value
        else
            value = info.absvalue
        end
        if value then -- some unusual units do not always set value field
            value = abs(value)
            singular = (0 < value and value < 1.0001)
        end
    end
    if unit_table.engscale then
        -- engscale: so "|1|e3kg" gives "1 thousand kilograms" (plural)
        singular = false
    end
    key = (adjectival or singular) and 'name1' or 'name2'
    if parms.opt_sp_us then
        key = key .. '_us'
    end
else
    if unit_table.builtin == 'hand' then
        if parms.opt_hand_hh then
            unit_table.symbol = 'hh' -- LATER: might want i18n applied to this
        end
    end
    unit_table.sep = '&nbsp;';
    key = parms.opt_sp_us and 'sym_us' or 'symbol'
end
return linked_id(parms, unit_table, key, want_link, info.clean), want_name

```

end

```
local function decorate_value(parms, unit_table, which, number_word)
  -- If needed, update unit_table so values will be shown with extra information.
  -- For consistency with the old template (but different from fmtpower),
  -- the style to display powers of 10 includes "display:none" to allow some
  -- browsers to copy, for example, "10³" as "10^3", rather than as "103".
  local info
  local engscale = unit_table.engscale
  local prefix = unit_table.vprefix
  if engscale or prefix then
    info = unit_table.valinfo[which]
    if info.decorated then
      return -- do not redecorate if repeating convert
    end
    info.decorated = true
    if engscale then
      local inout = unit_table.inout
      local abbr = parms.abbr
      if (abbr == 'on' or abbr == inout) and not parms.number_word then
        info.show = info.show ..
          '<span style="margin-left:0.2em">x<span style="margin-left:
from_en('10') ..
          '</span></span><s style="display:none">^</s><sup>' ..
          from_en(tostring(engscale.exponent)) .. '</sup>'
      elseif number_word then
        local number_id
        local lk = parms.lk
        if lk == 'on' or lk == inout then
          number_id = make_link(engscale.link, engscale[1])
        else
          number_id = engscale[1]
        end
        -- WP:NUMERAL recommends "&nbsp;" in values like "12 million".
        info.show = info.show .. (parms.opt_adjectival and '-' or '&nbsp;')
      end
    end
    if prefix then
      info.show = prefix .. info.show
    end
  end
end
```

```
local function process_input(parms, in_current)
  -- Processing required once per conversion.
  -- Return block of text to represent input (value/unit).
  if parms.opt_output_only or parms.opt_output_number_only or parms.opt_output_unit_only then
    parms.joins = { '', '' }
    return ''
  end
  local first_unit
```

```

local composite = in_current.composite -- nil or table of units
if composite then
    first_unit = composite[1]
else
    first_unit = in_current
end
local id1, want_name = make_id(parms, 1, first_unit)
local sep = first_unit.sep -- separator between value and unit, set by make_id
local preunit = parms.preunit1
if preunit then
    sep = ' ' -- any separator is included in preunit
else
    preunit = ''
end
if parms.opt_input_unit_only then
    parms.joins = { ' ', ' ' }
    if composite then
        local parts = { id1 }
        for i, unit in ipairs(composite) do
            if i > 1 then
                table.insert(parts, (make_id(parms, 1, unit)))
            end
        end
        id1 = table.concat(parts, ' ')
    end
    if want_name and parms.opt_adjectival then
        return preunit .. hyphenated(id1)
    end
    return preunit .. id1
end
if parms.opt_also_symbol and not composite and not parms.opt_flip then
    local join1 = parms.joins[1]
    if join1 == ' (' or join1 == ' [' then
        parms.joins = { ' [' .. first_unit[parms.opt_sp_us and 'sym_us' or 'symbol']
    end
end
if in_current.builtin == 'mach' and first_unit.sep ~= '' then -- '' means omitsep with no
    local prefix = id1 .. '&nbsp;'
    local range = parms.range
    local valinfo = first_unit.valinfo
    local result = prefix .. valinfo[1].show
    if range then
        -- For simplicity and because more not needed, handle one range item only.
        local prefix2 = make_id(parms, 2, first_unit) .. '&nbsp;'
        result = range_text(range[1], want_name, parms, result, prefix2 .. valinfo
    end
    return preunit .. result
end
if composite then
    -- Simplify: assume there is no range, and no decoration.
    local mid = (not parms.opt_flip) and parms.mid or ''

```

```

    local sep1 = '&nbsp;';
    local sep2 = ' '
    if parms.opt_adjectival and want_name then
        sep1 = '-'
        sep2 = '-'
    end
    if omitsep and sep == '' then
        -- Testing the id of the most significant unit should be sufficient.
        sep1 = ''
        sep2 = ''
    end
    local parts = { first_unit.valinfo[1].show .. sep1 .. id1 }
    for i, unit in ipairs(composite) do
        if i > 1 then
            table.insert(parts, unit.valinfo[1].show .. sep1 .. (make_id(parms
            end
        end
    end
    return table.concat(parts, sep2) .. mid
end
local add_unit = (parms.abbr == 'mos') or
    parms[parms.opt_flip and 'out_range_x' or 'in_range_x'] or
    (not want_name and parms.abbr_range_x)
local range = parms.range
if range and not add_unit then
    unlink(first_unit)
end
local id = range and make_id(parms, range.n + 1, first_unit) or id1
local extra, was_hyphenated = hyphenated_maybe(parms, want_name, sep, id, 'in')
if was_hyphenated then
    add_unit = false
end
local result
local valinfo = first_unit.valinfo
if range then
    for i = 0, range.n do
        local number_word
        if i == range.n then
            add_unit = false
            number_word = true
        end
        decorate_value(parms, first_unit, i+1, number_word)
        local show = valinfo[i+1].show
        if add_unit then
            show = show .. first_unit.sep .. (i == 0 and id1 or make_id(parms,
        end
        if i == 0 then
            result = show
        else
            result = range_text(range[i], want_name, parms, result, show, 'in'
        end
    end
end
end

```

```

else
    decorate_value(parms, first_unit, 1, true)
    result = valinfo[1].show
end
return result .. preunit .. extra
end

local function process_one_output(parms, out_current)
    -- Processing required for each output unit.
    -- Return block of text to represent output (value/unit).
    local inout = out_current.inout -- normally 'out' but can be 'in' for order=out
    local id1, want_name = make_id(parms, 1, out_current)
    local sep = out_current.sep -- set by make_id
    local preunit = parms.preunit2
    if preunit then
        sep = '' -- any separator is included in preunit
    else
        preunit = ''
    end
    if parms.opt_output_unit_only then
        if want_name and parms.opt_adjectival then
            return preunit .. hyphenated(id1)
        end
        return preunit .. id1
    end
    if out_current.builtin == 'mach' and out_current.sep ~= '' then -- '' means omitsep with
        local prefix = id1 .. '&nbsp;'
        local range = parms.range
        local valinfo = out_current.valinfo
        local result = prefix .. valinfo[1].show
        if range then
            -- For simplicity and because more not needed, handle one range item only.
            result = range_text(range[1], want_name, parms, result, prefix .. valinfo[1].show)
        end
        return preunit .. result
    end
    local add_unit = (parms[parms.opt_flip and 'in_range_x' or 'out_range_x'] or
        (not want_name and parms.abbr_range_x)) and
        not parms.opt_output_number_only
    local range = parms.range
    if range and not add_unit then
        unlink(out_current)
    end
    local id = range and make_id(parms, range.n + 1, out_current) or id1
    local extra, was_hyphenated = hyphenated_maybe(parms, want_name, sep, id, inout)
    if was_hyphenated then
        add_unit = false
    end
    local result
    local valinfo = out_current.valinfo
    if range then

```

```

    for i = 0, range.n do
        local number_word
        if i == range.n then
            add_unit = false
            number_word = true
        end
        decorate_value(parms, out_current, i+1, number_word)
        local show = valinfo[i+1].show
        if add_unit then
            show = show .. out_current.sep .. (i == 0 and id1 or make_id(parms
        end
        if i == 0 then
            result = show
        else
            result = range_text(range[i], want_name, parms, result, show, ino
        end
    end
end
else
    decorate_value(parms, out_current, 1, true)
    result = valinfo[1].show
end
if parms.opt_output_number_only then
    return result
end
return result .. preunit .. extra
end

```

```

local function make_output_single(parms, in_unit_table, out_unit_table)
    -- Return true, item where item = wikitext of the conversion result
    -- for a single output (which is not a combination or a multiple);
    -- or return false, t where t is an error message table.
    if parms.opt_order_out and in_unit_table.unitcode == out_unit_table.unitcode then
        out_unit_table.valinfo = in_unit_table.valinfo
    else
        out_unit_table.valinfo = collection()
        for _, v in ipairs(in_unit_table.valinfo) do
            local success, info = cvtround(parms, v, in_unit_table, out_unit_table)
            if not success then return false, info end
            out_unit_table.valinfo:add(info)
        end
    end
end
return true, process_one_output(parms, out_unit_table)
end

```

```

local function make_output_multiple(parms, in_unit_table, out_unit_table)
    -- Return true, item where item = wikitext of the conversion result
    -- for an output which is a multiple (like 'ftin');
    -- or return false, t where t is an error message table.
    local inout = out_unit_table.inout -- normally 'out' but can be 'in' for order=out
    local multiple = out_unit_table.multiple -- table of scaling factors (will not be nil)
    local combos = out_unit_table.combination -- table of unit tables (will not be nil)

```

```

local abbr = parms.abbr
local abbr_org = parms.abbr_org
local disp = parms.disp
local want_name = (abbr_org == nil and (disp == 'or' or disp == 'slash')) or
                  not (abbr == 'on' or abbr == inout or abbr == 'mos')
local want_link = (parms.lk == 'on' or parms.lk == inout)
local mid = parms.opt_flip and parms.mid or ''
local sep1 = '&nbsp;'
local sep2 = ''
if parms.opt_adjectival and want_name then
    sep1 = '-'
    sep2 = '-'
end
local do_spell = parms.opt_spell_out
parms.opt_spell_out = nil -- so the call to cvtround does not spell the value
local function make_result(info, isfirst)
    local fmt, outvalue, sign
    local results = {}
    for i = 1, #combos do
        local tfrac, thisvalue, strforce
        local out_current = combos[i]
        out_current.inout = inout
        local scale = multiple[i]
        if i == 1 then -- least significant unit ('in' from 'ftin')
            local decimals
            out_current.frac = out_unit_table.frac
            local success, outinfo = cvtround(parms, info, in_unit_table, out_
            if not success then return false, outinfo end
            if isfirst then
                out_unit_table.valinfo = { outinfo } -- in case output va
            end
            sign = outinfo.sign
            tfrac = outinfo.fraction_table
            if outinfo.is_scientific then
                strforce = outinfo.show
                decimals = ''
            elseif tfrac then
                decimals = ''
            else
                local show = outinfo.show -- number as a string in local
                local p1, p2 = show:find(numdot, 1, true)
                decimals = p1 and show:sub(p2 + 1) or '' -- text after nu
            end
            fmt = '%.' .. ulen(decimals) .. 'f' -- to reproduce precision
            if decimals == '' then
                if tfrac then
                    outvalue = floor(outinfo.raw_absvalue) -- integer
                else
                    outvalue = floor(outinfo.raw_absvalue + 0.5) -- 1
                end
            end
        else
    end
end

```

```

        outvalue = outinfo.absvalue
    end
end
if scale then
    outvalue, thisvalue = divide(outvalue, scale)
else
    thisvalue = outvalue
end
local id
if want_name then
    if varname then
        local clean
        if strforce or tfrac then
            clean = '.1' -- dummy value to force name for fl
        else
            clean = format(fmt, thisvalue)
        end
        id = variable_name(clean, out_current)
    else
        local key = 'name2'
        if parms.opt_adjectival then
            key = 'name1'
        elseif tfrac then
            if thisvalue == 0 then
                key = 'name1'
            end
        elseif parms.opt_singular then
            if 0 < thisvalue and thisvalue < 1.0001 then
                key = 'name1'
            end
        else
            if thisvalue == 1 then
                key = 'name1'
            end
        end
        id = out_current[key]
    end
else
    id = out_current['symbol']
end
if i == 1 and omit_separator(id) then
    -- Testing the id of the least significant unit should be sufficient
    sep1 = ''
    sep2 = ''
end
if want_link then
    local link = out_current.link
    if link then
        id = make_link(link, id, out_current)
    end
end
end

```

```

        local strval
        local spell_inout = (i == #combos or outvalue == 0) and inout or '' -- t
        if strforce and outvalue == 0 then
            sign = '' -- any sign is in strforce
            strval = strforce -- show small values in scientific notation; w
        elseif tfrac then
            local wholestr = (thisvalue > 0) and tostring(thisvalue) or nil
            strval = format_fraction(parms, spell_inout, false, wholestr, tfrac
        else
            strval = (thisvalue == 0) and from_en('0') or with_separator(parms
            if do_spell then
                strval = spell_number(parms, spell_inout, strval) or strval
            end
        end
        table.insert(results, strval .. sep1 .. id)
        if outvalue == 0 then
            break
        end
        fmt = '%.0f' -- only least significant unit can have a non-integral value
    end
    local reversed, count = {}, #results
    for i = 1, count do
        reversed[i] = results[count + 1 - i]
    end
    return true, sign .. table.concat(reversed, sep2)
end
local valinfo = in_unit_table.valinfo
local success, result = make_result(valinfo[1], true)
if not success then return false, result end
local range = parms.range
if range then
    for i = 1, range.n do
        local success, result2 = make_result(valinfo[i+1])
        if not success then return false, result2 end
        result = range_text(range[i], want_name, parms, result, result2, inout, {s
    end
end
return true, result .. mid
end

local function process(parms, in_unit_table, out_unit_table)
    -- Return true, s, outunit where s = final wikitext result,
    -- or return false, t where t is an error message table.
    linked_pages = {}
    local success, bad_output
    local bad_input_mcode = in_unit_table.bad_mcode -- nil if input unit is a valid convert
    local out_unit = parms.out_unit
    if out_unit == nil or out_unit == '' or type(out_unit) == 'function' then
        if bad_input_mcode or parms.opt_input_unit_only then
            bad_output = ''
        else

```

```

        local getdef = type(out_unit) == 'function' and out_unit or get_default
        success, out_unit = getdef(in_unit_table.valinfo[1].value, in_unit_table)
        parms.out_unit = out_unit
        if not success then
            bad_output = out_unit
        end
    end
end
if not bad_output and not out_unit_table then
    success, out_unit_table = lookup(parms, out_unit, 'any_combination')
    if success then
        local mismatch = check_mismatch(in_unit_table, out_unit_table)
        if mismatch then
            bad_output = mismatch
        end
    else
        bad_output = out_unit_table
    end
end
local lhs, rhs
local flipped = parms.opt_flip and not bad_input_mcode
if bad_output then
    rhs = (bad_output == '') and '' or message(parms, bad_output)
elseif parms.opt_input_unit_only then
    rhs = ''
else
    local combos -- nil (for 'ft' or 'ftin'), or table of unit tables (for 'm ft')
    if not out_unit_table.multiple then -- nil/false ('ft' or 'm ft'), or table of fr
        combos = out_unit_table.combination
    end
    local frac = parms.frac -- nil or denominator of fraction for output values
    if frac then
        -- Apply fraction to the unit (if only one), or to non-SI units (if a comb
        -- except that if a precision is also specified, the fraction only applie
        -- the hand unit; that allows the following result:
        -- {{convert|156|cm|in hand|1|frac=2}} → 156 centimetres (61.4 in; 15.1½ h
        -- However, the following is handled elsewhere as a special case:
        -- {{convert|156|cm|hand in|1|frac=2}} → 156 centimetres (15.1½ hands; 61½
        if combos then
            local precision = parms.precision
            for _, unit in ipairs(combos) do
                if unit.builtin == 'hand' or (not precision and not unit.frac)
                then
                    unit.frac = frac
                end
            end
        else
            out_unit_table.frac = frac
        end
    end
end
local outputs = {}
local imax = combos and #combos or 1 -- 1 (single unit) or number of unit tables

```

```

    if imax == 1 then
        parms.opt_order_out = nil -- only useful with an output combination
    end
    if not flipped and not parms.opt_order_out then
        -- Process left side first so any duplicate links (from lk=on) are suppressed
        -- on right. Example: {{convert|28|e9pc|e9ly|abbr=off|lk=on}}
        lhs = process_input(parms, in_unit_table)
    end
    for i = 1, imax do
        local success, item
        local out_current = combos and combos[i] or out_unit_table
        out_current.inout = 'out'
        if i == 1 then
            if imax > 1 and out_current.builtin == 'hand' then
                out_current.out_next = combos[2] -- built-in hand can input
            end
            if parms.opt_order_out then
                out_current.inout = 'in'
            end
        end
        if out_current.multiple then
            success, item = make_output_multiple(parms, in_unit_table, out_current)
        else
            success, item = make_output_single(parms, in_unit_table, out_current)
        end
        if not success then return false, item end
        outputs[i] = item
    end
    if parms.opt_order_out then
        lhs = outputs[1]
        table.remove(outputs, 1)
    end
    local sep = parms.table_joins and parms.table_joins[2] or parms.join_between
    rhs = table.concat(outputs, sep)
end
if flipped or not lhs then
    local input = process_input(parms, in_unit_table)
    if flipped then
        lhs = rhs
        rhs = input
    else
        lhs = input
    end
end
if parms.join_before then
    lhs = parms.join_before .. lhs
end
local wikitext
if bad_input_mcode then
    if bad_input_mcode == '' then
        wikitext = lhs
    end
end

```

```

        else
            wikitext = lhs .. message(parms, bad_input_mcode)
        end
elseif parms.table_joins then
    wikitext = parms.table_joins[1] .. lhs .. parms.table_joins[2] .. rhs
else
    wikitext = lhs .. parms.joins[1] .. rhs .. parms.joins[2]
end
if parms.warnings and not bad_input_mcode then
    wikitext = wikitext .. parms.warnings
end
return true, get_styles(parms) .. wikitext, out_unit_table
end

local function main_convert(frame)
    -- Do convert, and if needed, do it again with higher default precision.
    local parms = { frame = frame } -- will hold template arguments, after translation
    set_config(frame.args)
    local success, result = get_parms(parms, frame:getParent().args)
    if success then
        if type(result) ~= 'table' then
            return tostring(result)
        end
        local in_unit_table = result
        local out_unit_table
        for _ = 1, 2 do -- use counter so cannot get stuck repeating convert
            success, result, out_unit_table = process(parms, in_unit_table, out_unit_table)
            if success and parms.do_convert_again then
                parms.do_convert_again = false
            else
                break
            end
        end
    end
end

-- If input=x gives a problem, the result should be just the user input
-- (if x is a property like P123 it has been replaced with '').
-- An unknown input unit would display the input and an error message
-- with success == true at this point.
-- Also, can have success == false with a message that outputs an empty string.
if parms.input_text then
    if success and not parms.have_problem then
        return result
    end
    local cat
    if parms.tracking then
        -- Add a tracking category using the given text as the category sort key.
        -- There is currently only one type of tracking, but in principle multiple
        -- items could be tracked, using different sort keys for convenience.
        cat = wanted_category('tracking', parms.tracking)
    end
    return parms.input_text .. (cat or '')
end

```

```

end
return success and result or message(parms, result)
end

local function _unit(unitcode, options)
-- Helper function for Module:Val to look up a unit.
-- Parameter unitcode must be a string to identify the wanted unit.
-- Parameter options must be nil or a table with optional fields:
--   value = number (for sort key; default value is 1)
--   scaled_top = nil for a normal unit, or a number for a unit which is
--               the denominator of a per unit (for sort key)
--   si = { 'symbol', 'link' }
--         (a table with two strings) to make an SI unit
--         that will be used for the look up
--   link = true if result should be [[linked]]
--   sort = 'on' or 'debug' if result should include a sort key in a
--         span element ('debug' makes the key visible)
--   name = true for the name of the unit instead of the symbol
--   us = true for the US spelling of the unit, if any
-- Return nil if unitcode is not a non-empty string.
-- Otherwise return a table with fields:
--   text = requested symbol or name of unit, optionally linked
--   scaled_value = input value adjusted by unit scale; used for sort key
--   sortspan = span element with sort key like that provided by {{ntsh}},
--             calculated from the result of converting value
--             to a base unit with scale 1.
--   unknown = true if the unitcode was not known
unitcode = strip(unitcode)
if unitcode == nil or unitcode == '' then
    return nil
end
set_config({})
linked_pages = {}
options = options or {}
local parms = {
    abbr = options.name and 'off' or 'on',
    lk = options.link and 'on' or nil,
    opt_sp_us = options.us and true or nil,
    opt_ignore_error = true, -- do not add pages using this function to 'what links I
    opt_sortable_on = options.sort == 'on' or options.sort == 'debug',
    opt_sortable_debug = options.sort == 'debug',
}
if options.si then
-- Make a dummy table of units (just one unit) for lookup to use.
-- This makes lookup recognize any SI prefix in the unitcode.
local symbol = options.si[1] or '?'
parms.unittable = { [symbol] = {
    _name1 = symbol,
    _name2 = symbol,
    _symbol = symbol,
    utype = symbol,

```

```

        scale = symbol == 'g' and 0.001 or 1,
        prefixes = 1,
        default = symbol,
        link = options.si[2],
    }}
end
local success, unit_table = lookup(parms, unitcode, 'no_combination')
if not success then
    unit_table = setmetatable({
        symbol = unitcode, name2 = unitcode, utype = unitcode,
        scale = 1, default = '', defkey = '', linkey = '' }, unit_mt)
end
local value = tonumber(options.value) or 1
local clean = tostring(abs(value))
local info = {
    value = value,
    altvalue = value,
    singular = (clean == '1'),
    clean = clean,
    show = clean,
}
unit_table.inout = 'in'
unit_table.valinfo = { info }
local sortspan, scaled_value
if options.sort then
    sortspan, scaled_value = make_table_or_sort(parms, value, info, unit_table, option
end
return {
    text = make_id(parms, 1, unit_table),
    sortspan = sortspan,
    scaled_value = scaled_value,
    unknown = not success and true or nil,
}
end

return { convert = main_convert, _unit = _unit }

```

## Modul:Convert/text/sandbox

This page defines text used by [Module:Convert](#). All documentation (from [Module:Convert/doc](#)) is at that module. The text includes messages and categories output by the module, and parameters used as input.

This is a separate module to simplify translation for use on another wiki. For example, see **translation\_table** and the other tables in [bn:Module:Convert/text](#). Documentation is at [Template:Convert/Transwiki guide](#).

Any changes should first be tested at [Module:Convert/text/sandbox](#)—see [Template:Convert/testcases#Sandbox testcases](#).

---

```

-- Text used by Module:Convert for enwiki.
-- This is a separate module to simplify translation for use on another wiki.
-- See [[:en:Template:Convert/Transwiki guide]] if copying to another wiki.

-- Some units accept an SI prefix before the unit code, such as "kg" for kilogram.
local SIPrefixes = {
    -- The prefix field is what the prefix should be, if different from the prefix used.
    ['Y'] = { exponent = 24, name = 'yotta',           },
    ['Z'] = { exponent = 21, name = 'zetta',          },
    ['E'] = { exponent = 18, name = 'exa' ,           },
    ['P'] = { exponent = 15, name = 'peta' ,          },
    ['T'] = { exponent = 12, name = 'tera' ,          },
    ['G'] = { exponent = 9,  name = 'giga' ,           },
    ['M'] = { exponent = 6,  name = 'mega' ,           },
    ['k'] = { exponent = 3,  name = 'kilo' ,           },
    ['h'] = { exponent = 2,  name = 'hecto' ,          },
    ['da'] = { exponent = 1, name = 'deca' , name_us = 'deka' },
    ['d'] = { exponent = -1, name = 'deci' ,           },
    ['c'] = { exponent = -2, name = 'centi' ,          },
    ['m'] = { exponent = -3, name = 'milli' ,          },
    ['μ'] = { exponent = -6, name = 'micro' ,           }, -- key = 'GREEK SMALL LETTER MU'
    ['µ'] = { exponent = -6, name = 'micro', prefix = 'µ' }, -- key = 'MICRO SIGN' (U+00B5)
    ['u'] = { exponent = -6, name = 'micro', prefix = 'µ' }, -- not an SI prefix, but all
    ['n'] = { exponent = -9, name = 'nano' ,           },
    ['p'] = { exponent = -12, name = 'pico' ,           },
    ['f'] = { exponent = -15, name = 'femto' ,          },
    ['a'] = { exponent = -18, name = 'atto' ,           },
    ['z'] = { exponent = -21, name = 'zepto' ,          },
    ['y'] = { exponent = -24, name = 'yocto' ,          },
}

-- Some units can be qualified with one of the following prefixes, when linked.
local customary_units = {
    { "US", link = "United States customary units" },
    { "U.S.", link = "United States customary units" },
    { "imperial", link = "Imperial units" },
    { "imp", link = "Imperial units" },
}

-- Names when using engineering notation (a prefix of "eN" where N is a number; example "e6km").
-- key = { "name", link = "article title", exponent = numeric_key_value }
-- If lk=on and link is defined, the name of the number will appear as a link.
local eng_scales = {
    ["3"] = { "thousand", exponent = 3 },
    ["6"] = { "million", exponent = 6 },
    ["9"] = { "billion", link = "1000000000 (number)", exponent = 9 },
    ["12"] = { "trillion", link = "1000000000000 (number)", exponent = 12 },
    ["15"] = { "quadrillion", link = "1000000000000000 (number)", exponent = 15 },
}

```

```

local all_categories = {
    unit = "[[Category:Convert errors]]",
    option = "[[Category:Convert errors]]",
    warning = '[[Category:Convert invalid options]]',
    tracking = '[[Category:Convert tracking]]',
}

-- For some error messages, the following puts the wanted style around
-- each unit code marked like '...%{ft%}...'.
local unitcode_regex = '%%([{}])'
local unitcode_replace = { ['{'] = '', ['}'] = '' } -- no longer need the more elaborate substi

-- All messages that may be displayed if a problem occurs.
local all_messages = {
    -- Message format string: $1=title, $2=text, $3=category, $4=anchor.
    -- Each displayed message starts with "Convert:" so can easily locate by searching article
    cvt_format = '<sup class="noprint Inline-Template" style="white-space:nowrap;"><i>[[Help:
    cvt_format2 = '<sup class="noprint Inline-Template" style="white-space:nowrap;">[[Help:Cor
    cvt_format_preview = '<strong class="error">Error in convert: $1 [[Help:Convert messages#
    -- Each of following messages is a table:
    -- { [1] = 'title',          -- mouseover title text
    --   [2] = 'text',           -- link text displayed in article
    --   [3] = 'category key',   -- key to lookup category in all_categories
    --   [4] = 'anchor',         -- anchor for link to relevant section on help page
    --   regex = gsub_regex,
    --   replace = gsub_table,
    -- }
    --                                     Mouseover title text                               Link text
    cvt_bad_input      = { 'input "$1" must be a number and unit'      , 'invalid input'
    cvt_bad_num        = { 'Value "$1" must be a number'                , 'invalid number'
    cvt_big_prec       = { 'Precision "$1" is too large'                 , 'precision too large'
    cvt_invalid_num    = { 'Number has overflowed'                       , 'number overflow'
    cvt_no_num         = { 'Needs the number to be converted'           , 'needs a number'
    cvt_no_num2        = { 'Needs another number for a range'           , 'needs another numbe
    cvt_bad_altitude  = { '"$1" needs an integer'                       , 'invalid altitude'
    cvt_bad_frac       = { '"$1" needs an integer above 1'              , 'invalid fraction'
    cvt_bad_prec       = { 'Precision "$1" must be an integer'          , 'invalid precision'
    cvt_bad_sigfig     = { '"$1" needs a positive integer'              , 'invalid sigfig'
    cvt_empty_option   = { 'Ignored empty option "$1"'                  , 'empty option'
    cvt_deprecated     = { 'Option "$1" is deprecated'                  , '*'
    cvt_no_spell       = { 'Spelling is not available'                   , 'bug, ask for help'
    cvt_unknown_option = { 'Ignored invalid option "$1"'                 , 'invalid option'
    cvt_wd_fail        = { 'Unable to access Wikidata'                   , 'wikidata problem'
    cvt_bad_default    = { 'Unit "$1" has an invalid default'           , 'bug, ask for help'
    cvt_bad_unit       = { 'Unit "$1" is invalid here'                   , 'unit invalid here'
    cvt_no_default     = { 'Unit "$1" has no default output unit'       , 'bug, ask for help'
    cvt_no_unit        = { 'Needs name of unit'                          , 'needs unit name'
    cvt_unknown        = { 'Unit name "$1" is not known'                 , 'unknown unit'
    cvt_should_be     = { '$1'                                           , 'ambiguous unit'
    cvt_mismatch       = { 'Cannot convert "$1" to "$2"'                 , 'unit mismatch'
    cvt_bug_convert    = { 'Bug: Cannot convert between specified units' , 'bug, ask for help'
    cvt_lookup         = { 'Unit "$1" is incorrectly defined'           , 'bug, ask for help'

```

```

}

-- Text to join input value/unit with output value/unit.
local disp_joins = {
    -- [1]=before output, [2]=after output, [3]=between outputs in a combination; default " ; '
    -- [wantname] gives default abbr=off
    ["or"]      = { " or "      , "" , " or " , wantname = true },
    ["sqbr-sp"] = { " ["      , "]" },
    ["sqbr-nbsp"] = { "&nbsp;[" , "]" },
    ["comma"]    = { " , "      , "" , " , " },
    ["slash-sp"] = { " / "      , "" , wantname = true },
    ["slash-nbsp"] = { "&nbsp;/ " , "" , wantname = true },
    ["slash-nosp"] = { "/"      , "" , wantname = true },
    ["b"]        = { " ("      , ")" },
    ["(or)"]     = { " ("      , ")" , " or " },
    ["br"]       = { "<br />" , "" , wantname = true },
    ["br()"]     = { "<br />(" , ")" , wantname = true },
}

```

```

-- Text to separate values in a range.
local range_types = {
    -- Specifying a table requires either:
    -- * "off" and "on" values (for "abbr=off" and "abbr=on"), or
    -- * "input" and "output" values (for LHS and RHS);
    -- other fields are optional.
    -- When "adj=on|abbr=off" applies, spaces in range text are replaced with hyphens.
    -- With "exception = true", that also occurs with "adj=on|abbr=on".
    -- If "adj" is defined here, that text (unchanged) is used with "adj=on".
    ["+"]       = " + ",
    [","]       = ",&nbsp;",
    [", and"]   = ", and ",
    [", or"]    = ", or ",
    ["by"]      = " by ",
    ["-"]       = "-",
    ["to about"] = " to about ",
    ["and"]     = { off = " and " , on = " and " , exception = true },
    ["and(-)"] = { input = " and " , output = "-" },
    ["or"]      = { off = " or " , on = " or " , exception = true },
    ["to"]      = { off = " to " , on = " to " , exception = true },
    ["to(-)"]   = { input = "&nbsp;to " , output = "-" },
    ["+/-"]     = { off = "&nbsp;±&nbsp;" , on = "&nbsp;±&nbsp;" , adj = "&nbsp;±&nbsp;" , is_range = true },
    ["by(x)"]   = { input = " by " , output = " x&nbsp;" , out_range_x = true },
    ["x"]       = { off = " by " , on = " x&nbsp;" , abbr_range_x = true },
    ["xx"]      = "&nbsp;x&nbsp;",
    ["*"]       = "x",
    ["/"]       = "&thinsp;/&thinsp;" , -- for a table of high/low temperatures with {{convert|
}

```

```

local range_aliases = {
    -- ["alternative name for a range"] = "standard range name"
    ["-"]       = "-",
}

```

```

["&ndash;"] = "-",
["x"]      = "x",
["&times;"] = "x",
["±"]      = "+/-",
["&plusmn;"] = "+/-",
}

-- Convert accepts range text delimited with whitespace, for example, {{convert|1 to 2|ft}}.
-- In addition, the following "words" are accepted without spaces, for example, {{convert|1-2|ft}}
-- Words must be in correct order for searching, for example, 'x' after 'xx'.
local range_words = { '-', '- ', 'xx', 'x', '*' }

local ranges = {
    types = range_types,
    aliases = range_aliases,
    words = range_words,
}

-- Valid option names.
local en_option_name = {
    -- ["local text for option name"] = "en name used in this module"
    ["$"] = "$",
    ["abbr"] = "abbr",
    ["adj"] = "adj",
    ["altitude_ft"] = "altitude_ft",
    ["altitude_m"] = "altitude_m",
    ["comma"] = "comma",
    ["debug"] = "debug",
    ["disp"] = "disp",
    ["frac"] = "frac",
    ["input"] = "input",
    ["lang"] = "lang",
    ["lk"] = "lk",
    ["order"] = "order",
    ["qid"] = "qid",
    ["qual"] = "qual",
    ["qualifier"] = "qual",
    ["round"] = "round",
    ["sigfig"] = "sigfig",
    ["sing"] = "adj", -- "sing" is an old alias for "adj"
    ["sortable"] = "sortable",
    ["sp"] = "sp",
    ["spell"] = "spell",
    ["stylein"] = "stylein",
    ["styleout"] = "styleout",
    ["tracking"] = "tracking",
}

-- Valid option values.
-- Convention: parms.opt_xxx refers to an option that is set here
-- (not intended to be set by the template which invokes this module).

```

```

-- Example: At enwiki, "abbr" includes:
-- ["values"] = "opt_values"
-- As a result, if the template uses abbr=values, Module:Convert sets:
--   parms["opt_values"] = true
--   parms["abbr"] = nil
-- Therefore parms.abbr will be nil, or will have one of the listed values
-- that do not start with "opt_".
-- An option value of form "xxx?" is the same as "xxx" but shows the input as deprecated.
local en_option_value = {
    ["$"] = 'TEXT',          -- TEXT should be a currency symbol that will be used
    ["abbr"] = {
        -- ["local text for option value"] = "en value used in this module"
        ["def"] = "",        -- ignored (some wrapper templates call convert w
        ["h"] = "on",        -- abbr=on + use "h" for hand unit (default)
        ["hh"] = "opt_hand_hh", -- abbr=on + use "hh" for hand unit
        ["in"] = "in",       -- use symbol for LHS unit
        ["none"] = "off",    -- old name for "off"
        ["off"] = "off",     -- use name for all units
        ["on"] = "on",       -- use symbol for all units
        ["out"] = "out",     -- use symbol for RHS unit (default)
        ["unit"] = "unit",   -- abbr=on but abbreviate units only: e6km → milli
        ["values"] = "opt_values", -- show only input and output numbers, not units
        ["~"] = "opt_also_symbol", -- show input unit symbol as well as name
    },
    ["adj"] = {
        ["mid"] = "opt_adjectival, opt_adj_mid", -- adj=on with user-specified text after
        ["off"] = "", -- ignored (off is the default)
        ["on"] = "opt_adjectival", -- unit name is singular and hyphenated
        ["pre"] = "opt_one_preunit", -- user-specified text before input unit
        ["ri0"] = "opt_ri=0", -- round input with precision = 0
        ["ri1"] = "opt_ri=1", -- round input with precision = 1
        ["ri2"] = "opt_ri=2", -- round input with precision = 2
        ["ri3"] = "opt_ri=3", -- round input with precision = 3
    },
    ["altitude_ft"] = 'INTEGER',
    ["altitude_m"] = 'INTEGER',
    ["comma"] = {
        ["5"] = "opt_comma5", -- only use numsep grouping if 5 or more digits
        ["gaps"] = "opt_gaps", -- use gaps, not numsep, to separate groups of dig
        ["gaps3"] = "opt_gaps, opt_gaps3", -- group only in threes rather than default of
        ["off"] = "opt_nocomma", -- no numsep in input or output numbers
    },
    ["debug"] = {
        ["yes"] = "opt_sortable_debug", -- make the normally hidden sort key visible
    },
    ["disp"] = {
        ["5"] = "opt_round=5?", -- round output value to nearest 5
        ["b"] = "b", -- join: '(...)'
        ["(or)"] = "(or)", -- join: '(...)' with 'or' between outputs in a c
        ["br"] = "br", -- join: '<br />'
        ["br()"] = "br()", -- join: '<br />(...)'
    }
}

```

```

["comma"] = "comma",          -- join: ','
["flip"] = "opt_flip",       -- reverse order of input/output
["number"] = "opt_output_number_only", -- display output value (not input, and no
["or"] = "or",               -- join: 'or'
["out"] = "opt_output_only",
["output number only"] = "opt_output_number_only",
["output only"] = "opt_output_only",
["preunit"] = "opt_two_preunits", -- user-specified text before input and output
["sqbr"] = "sqbr",          -- join: '[...]'
["table"] = "opt_table",    -- output is suitable for a table cell with align=
["tablecen"] = "opt_tablecen", -- output is suitable for a table cell with align=
["unit"] = "opt_input_unit_only", -- display input symbol/name (not output, and no
["unit or text"] = "opt_input_unit_only, opt_ignore_error", -- display input symbol
["unit2"] = "opt_output_unit_only",
["x"] = "x",                -- join: <first>...<second> (user-specified text)
},
["frac"] = 'INTEGER',
["input"] = 'TEXT',         -- TEXT should be value<space><unitcode> or <wikidata
["lang"] = {                -- language for output digits (both en and local digit
  ["en"] = "opt_lang_en",   -- use en digits for numbers, regardless of local
  ["local"] = "opt_lang_local", -- use local digits for numbers (default, although
},
["lk"] = {
  ["in"] = "in",           -- link LHS unit name or symbol
  ["off"] = "off",        -- do not link: same as default except for hand ur
  ["on"] = "on",          -- link all unit names or symbols (but not twice i
  ["out"] = "out",        -- link RHS unit name or symbol
},
["order"] = {
  ["flip"] = "opt_flip",   -- reverse order of input/output
  ["out"] = "opt_order_out", -- do not show input; instead, use order in output
},
["qid"] = 'TEXT',          -- TEXT should be a Wikidata Q item identifier
["qual"] = 'TEXT',        -- TEXT should be a Wikidata Q item identifier
["round"] = {
  ["0.5"] = "opt_round=0.5", -- round output value to nearest 0.5
  ["5"] = "opt_round=5",     -- round output value to nearest 5
  ["10"] = "opt_round=10",  -- round output value to nearest 10 (same as but c
  ["25"] = "opt_round=25",  -- round output value to nearest 25
  ["50"] = "opt_round=50",  -- round output value to nearest 50
  ["each"] = "opt_round_each", -- using default precision in a range, round each
},
["sigfig"] = 'INTEGER',
["sortable"] = {
  ["off"] = "",            -- ignored (off is the default)
  ["on"] = "opt_sortable_on", -- output sort key for use in a sortable table, ba
  ["debug"] = "opt_sortable_on, opt_sortable_debug", -- |sortable=debug is the same
},
["sp"] = {
  ["us"] = "opt_sp_us",    -- use U.S. spelling (like "meter" instead of defa
},

```

```

["spell"] = {
    -- only English spelling is supported; not scientific
    ["in"] = "opt_spell_in", -- spell input value in words
    ["In"] = "opt_spell_in, opt_spell_upper", -- spell input value in v
    ["on"] = "opt_spell_in, opt_spell_out", -- spell input and output
    ["On"] = "opt_spell_in, opt_spell_out, opt_spell_upper", -- same, with first letter
},
["stylein"] = 'TEXT',
["styleout"] = 'TEXT',
["tracking"] = 'TEXT',
}

local titles = {
    ["frac"] = "Fraction/styles.css",
    ["sfrac"] = "Sfrac/styles.css",
}

return {
    SIprefixes = SIprefixes,
    all_categories = all_categories,
    all_messages = all_messages,
    currency = { ['$'] = true, ['£'] = true, ['€'] = true, ['₹'] = true, ['¥'] = true,
    customary_units = customary_units,
    disp_joins = disp_joins,
    en_option_name = en_option_name,
    en_option_value = en_option_value,
    eng_scales = eng_scales,
    ranges = ranges,
    titles = titles,
}

```