

Toggle menu
Blue Gold Program Wiki

Navigation

- [Main page](#)
- [Recent changes](#)
- [Random page](#)
- [Help about MediaWiki](#)

Tools

- [What links here](#)
- [Related changes](#)
- [Special pages](#)
- [Permanent link](#)
- [Page information](#)

Personal tools

- [Log in](#)

personal-extra

Toggle search

Search

Random page

Views

- [View](#)
- [View source](#)
- [History](#)
- [PDF Export](#)

Actions

Module:Citation/CS1/Date validation

From Blue Gold Program Wiki

< [Module:Citation/CS1](#)

The printable version is no longer supported and may have rendering errors. Please update your browser bookmarks and please use the default browser print function instead.

```
{{#lst:Module:Citation/CS1/doc|header}}
```

This module contains routines that support the [Citation Style 1](#) and [Citation Style 2](#) date formats for citations on Wikipedia. In particular, this module contains a suite of functions that validate date formats and content for the variety of date-holding parameters associated with cs1|2 citations.

```
{{#lst:Module:Citation/CS1/doc|module_components_table}}
```

```
--[[-----< F O R W A R D   D E C L A R A T I O N S >-----  
-----  
]]
```

```
local is_set, in_array;  
-- imported functions from selected Module:Citation/CS1/Utilities  
local cfg;  
-- table of tables imported from selected Module:Citation/CS1/Configuration
```

```
--[[-----< F I L E - S C O P E   D E C L A R A T I O N S  
>-----
```

File-scope variables are declared here

```
]]  
  
local lang_object = mw.getContentLanguage();  
-- used by is_valid_accessdate(), is_valid_year(), date_name_xlate(); TODO:  
move to ~/Configuration?  
local year_limit;  
-- used by is_valid_year()
```

```
--[=[-----< I S _ V A L I D _ A C C E S S D A T E >-----  
-----
```

returns true if:

Wikipedia start date <= accessdate < today + 2 days

Wikipedia start date is 2001-01-15T00:00:00 UTC which is 979516800 seconds after 1970-01-01T00:00:00 UTC (the start of Unix time)

accessdate is the date provided in |accessdate= at time 00:00:00 UTC

today is the current date at time 00:00:00 UTC plus 48 hours

if today is 2015-01-01T00:00:00 then

adding 24 hours gives 2015-01-02T00:00:00 – one second more than today

adding 24 hours gives 2015-01-03T00:00:00 – one second more than tomorrow

This function does not work if it is fed month names for languages other than English. Wikimedia #time: parser

apparently doesn't understand non-English date month names. This function will always return false when the date contains a non-English month name because good1 is false after the call to lang.formatDate(). To get around that call this function with YYYY-MM-DD format dates.

```
] = ]
```

```
local function is_valid_accessdate (accessdate)
    local good1, good2;
    local access_ts, tomorrow_ts;
-- to hold unix time stamps representing the dates

    good1, access_ts = pcall (lang_object.formatDate, lang_object, 'U',
accessdate ); -- convert accessdate value to unix
timesatmp
    good2, tomorrow_ts = pcall (lang_object.formatDate, lang_object, 'U',
'today + 2 days' ); -- today midnight + 2 days is one second more than
all day tomorrow
    if good1 and good2 then
-- lang.formatDate() returns a timestamp in the local script which which
tonumber() may not understand
        access_ts = tonumber (access_ts) or
lang_object:parseFormattedNumber (access_ts); --
convert to numbers for the comparison;
        tomorrow_ts = tonumber (tomorrow_ts) or
lang_object:parseFormattedNumber (tomorrow_ts);
    else
        return false;
-- one or both failed to convert to unix time stamp
    end

    if 979516800 <= access_ts and access_ts < tomorrow_ts then
-- Wikipedia start date <= accessdate < tomorrow's date
        return true;
    else
        return false;
-- accessdate out of range
    end
end
```

```
--[[-----< I S _ V A L I D _ E M B A R G O _ D A T E >-----
-----
```

returns true and date value if that value has proper dmy, mdy, ymd format.

returns false and 9999 (embargoed forever) when date value is not proper format; assumes that when |embargo= is set, the editor intended to embargo a pmc but |embargo= does not hold a single date.

```

]]

local function is_valid_embargo_date (v)
    if v:match ('^%d%d%d%d%-%d%d%-%d%d$') or
-- ymd
        v:match ('^%d%d?%s+%a+%s+%d%d%d%d$') or
-- dmy
        v:match ('^%a+%s+%d%d?%s*,%s*%d%d%d%d$') then
-- mdy
            return true, v;
        end
    return false, '9999';
-- if here not good date so return false and set embargo date to long time in
future
end

--[[-----< G E T _ M O N T H _ N U M B E R >-----
-----

returns a number according to the month in a date: 1 for January, etc.
Capitalization and spelling must be correct.
If not a valid month, returns 0

]]

local function get_month_number (month)
    return cfg.date_names['local'].long[month] or
cfg.date_names['local'].short[month] or -- look for local
names first
        cfg.date_names['en'].long[month] or
cfg.date_names['en'].short[month] or -- failing that,
look for English names
        0;
-- not a recognized month name
end

--[[-----< G E T _ S E A S O N _ N U M B E R >-----
-----

returns a number according to the sequence of seasons in a year: 21 for
Spring, etc. Capitalization and spelling
must be correct. If not a valid season, returns 0.
    21-24 = Spring, Summer, Autumn, Winter, independent of "Hemisphere"

returns 0 when <param> is not |date=

Season numbering is defined by Extended Date/Time Format (EDTF) Specification
(https://www.loc.gov/standards/datetime/)
which became part of ISO 8601 in 2019. See '$Sub-year groupings'. The

```

standard defines various divisions using numbers 21-41. cs1|2 only supports generic seasons. EDTF does support the distinction between north and south hemisphere seasons but cs1|2 has no way to make that distinction.

These additional divisions not currently supported:

25-28 = Spring - Northern Hemisphere, Summer- Northern Hemisphere, Autumn - Northern Hemisphere, Winter - Northern Hemisphere

29-32 = Spring – Southern Hemisphere, Summer– Southern Hemisphere, Autumn – Southern Hemisphere, Winter - Southern Hemisphere

33-36 = Quarter 1, Quarter 2, Quarter 3, Quarter 4 (3 months each)

37-39 = Quadrimester 1, Quadrimester 2, Quadrimester 3 (4 months each)

40-41 = Semestral 1, Semestral-2 (6 months each)

]]

```
local function get_season_number (season, param)
```

```
  if 'date' ~= param then
```

```
    return 0;
```

```
-- season dates only supported by |date=
  end
```

```
  return cfg.date_names['local'].season[season] or
```

```
-- look for local names first
```

```
    cfg.date_names['en'].season[season] or
```

```
-- failing that, look for English names
```

```
    0;
```

```
-- not a recognized season name
```

```
end
```

```
--[[-----< G E T _ Q U A R T E R _ N U M B E R >-----
-----
```

returns a number according to the sequence of quarters in a year: 33 for first quarter, etc. Capitalization and spelling must be correct. If not a valid quarter, returns 0.

33-36 = Quarter 1, Quarter 2, Quarter 3, Quarter 4 (3 months each)

returns 0 when <param> is not |date=

Quarter numbering is defined by Extended Date/Time Format (EDTF) Specification (<https://www.loc.gov/standards/datetime/>)

which became part of ISO 8601 in 2019. See '\$Sub-year groupings'. The standard defines various divisions using numbers 21-41. cs1|2 only supports generic seasons and quarters.

These additional divisions not currently supported:

37-39 = Quadrimester 1, Quadrimester 2, Quadrimester 3 (4 months each)

40-41 = Semestral 1, Semestral-2 (6 months each)

```

]]

local function get_quarter_number (quarter, param)
    if 'date' ~= param then
        return 0;
    -- quarter dates only supported by |date=
    end
    quarter = mw.ustring.gsub (quarter, ' +', ' ');
    -- special case replace multiple space chars with a single space char
    return cfg.date_names['local'].quarter[quarter] or
    -- look for local names first
        cfg.date_names['en'].quarter[quarter] or
    -- failing that, look for English names
        0;
    -- not a recognized quarter name
end

--[[-----< GET _ P R O P E R _ N A M E _ N U M B E R
>-----

returns a non-zero number if date contains a recognized proper-name.
Capitalization and spelling must be correct.

returns 0 when <param> is not |date=

]]

local function get_proper_name_number (name, param)
    if 'date' ~= param then
        return 0;
    -- proper-name dates only supported by |date=
    end
    return cfg.date_names['local'].named[name] or
    -- look for local names dates first
        cfg.date_names['en'].named[name] or
    -- failing that, look for English names
        0;
    -- not a recognized named date
end

--[[-----< GET _ E L E M E N T _ N U M B E R <-----
-----

returns true if month or season or quarter or proper name is valid (properly
spelled, capitalized, abbreviated)

]]

local function get_element_number (element, param)

```

```

        local num;
        local funcs = {get_month_number, get_season_number,
get_quarter_number, get_proper_name_number};          -- list of functions to
execute in order
        for _, func in ipairs (funcs) do
-- spin through the function list
                num = func (element, param);
-- call the function and get the returned number
                if 0 ~= num then
-- non-zero when valid month season quarter
                        return num;
-- return that number
                end
        end
        return nil;
-- not valid
end

```

```

--[[-----< I S _ V A L I D _ Y E A R >-----
-----

```

Function gets current year from the server and compares it to year from a citation parameter. Years more than one year in the future are not acceptable.

```

]]

```

```

local function is_valid_year (year)
    if not is_set(year_limit) then
        year_limit = tonumber(os.date("%Y"))+1;
-- global variable so we only have to fetch it once
    end

```

```

        year = tonumber (year) or lang_object:parseFormattedNumber (year);
-- convert to numbers for the comparison;
        return year and (year <= year_limit) or false;
end

```

```

--[[-----< I S _ V A L I D _ D A T E >-----
-----

```

Returns true if day is less than or equal to the number of days in month and year is no farther into the future than next year; else returns false.

Assumes Julian calendar prior to year 1582 and Gregorian calendar thereafter. Accounts for Julian calendar leap years before 1582 and Gregorian leap years after 1582. Where the two calendars overlap (1582 to approximately

1923) dates are assumed to be Gregorian.

```
]]

local function is_valid_date (year, month, day)
local days_in_month = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
local month_length;
    if not is_valid_year(year) then
-- no farther into the future than next year
        return false;
    end
    month = tonumber(month);
-- required for YYYY-MM-DD dates
    if (2==month) then
-- if February
        month_length = 28;
-- then 28 days unless
        if 1582 > tonumber(year) then
-- Julian calendar
            if 0==(year%4) then
-- is a leap year?
                month_length = 29;
-- if leap year then 29 days in February
            end
        else
-- Gregorian calendar
            if (0==(year%4) and (0~=(year%100) or 0==(year%400)))
-- is a leap year?
                month_length = 29;
-- if leap year then 29 days in February
            end
        end
    else
        month_length=days_in_month[month];
    end

    if tonumber (day) > month_length then
        return false;
    end
    return true;
end
```

```
--[[-----< I S _ V A L I D _ M O N T H _ R A N G E _ S T
Y L E >-----
```

Months in a range are expected to have the same style: Jan–Mar or October–December but not February–Mar or Jul–August. There is a special test for May because it can be either short or long form.

Returns true when style for both months is the same


```

]]

local function is_valid_month_range_style (month1, month2)
local len1 = month1:len();
local len2 = month2:len();
    if len1 == len2 then
        return true;
-- both months are short form so return true
    elseif 'May' == month1 or 'May'== month2 then
        return true;
-- both months are long form so return true
    elseif 3 == len1 or 3 == len2 then
        return false;
-- months are mixed form so return false
    else
        return true;
-- both months are long form so return true
    end
end

--[[-----< I S _ V A L I D _ M O N T H _ S E A S O N _ R
A N G E >-----]]

```

Check a pair of months or seasons to see if both are valid members of a month or season pair.

Month pairs are expected to be left to right, earliest to latest in time.

All season ranges are accepted as valid because there are publishers out there who have published a Summer–Spring YYYY issue so ... ok

```

]]

local function is_valid_month_season_range(range_start, range_end, param)
    local range_start_number = get_month_number (range_start);
    local range_end_number;

    if 0 == range_start_number then
-- is this a month range?
        range_start_number = get_season_number (range_start, param);
-- not a month; is it a season? get start season number
        range_end_number = get_season_number (range_end, param);
-- get end season number

        if (0 ~= range_start_number) and (0 ~= range_end_number) then
            return true;
-- any season pairing is accepted
        end
        return false;
-- range_start and/or range_end is not a season
    end
end

```

```

        end
-- here when range_start is a month
        range_end_number = get_month_number (range_end);
-- get end month number
        if range_start_number < range_end_number then
-- range_start is a month; does range_start precede range_end?
            if is_valid_month_range_style (range_start, range_end) then
-- do months have the same style?
                return true;
-- proper order and same style
            end
        end
        return false;
-- range_start month number is greater than or equal to range end number; or
range end isn't a month
end

```

```

--[[-----< M A K E _ C O I N S _ D A T E >-----
-----

```

This function receives a table of date parts for one or two dates and an empty table reference declared in Module:Citation/CS1. The function is called only for |date= parameters and only if the |date=<value> is determined to be a valid date format. The question of what to do with invalid date formats is not answered here.

The date parts in the input table are converted to an ISO 8601 conforming date string:

```

single whole dates:          yyyy-mm-dd
month and year dates:       yyyy-mm
year dates:                  yyyy
ranges:                       yyyy-mm-dd/yyyy-mm-dd
                               yyyy-mm/yyyy-mm
                               yyyy/yyyy

```

Dates in the Julian calendar are reduced to year or year/year so that we don't have to do calendar conversion from Julian to Proleptic Gregorian.

The input table has:

```

year, year2 – always present; if before 1582, ignore months and days
if present
month, month2 – 0 if not provided, 1-12 for months, 21-24 for
seasons; 99 Christmas
day, day2 – 0 if not provided, 1-31 for days

```

the output table receives:

```

rftdate:          an IS8601 formatted date
rftchron:         a free-form version of the date, usually without
year which is in rftdate (season ranges and proper-name dates)

```

```

        rftssn:                one of four season keywords: winter, spring,
summer, fall (lowercase)
        rftquarter:           one of four values: 1, 2, 3, 4

]]

local function make_C0inS_date (input, tC0inS_date)
    local date;
-- one date or first date in a range
    local date2 = '';
-- end of range date
-- start temporary Julian / Gregorian calendar uncertainty detection
    local year = tonumber(input.year);
-- this temporary code to determine the extent of sources dated to the
Julian/Gregorian
    local month = tonumber(input.month);
-- interstice 1 October 1582 – 1 January 1926
    local day = tonumber (input.day);
    if (0 ~= day) and
-- day must have a value for this to be a whole date
        (((1582 == year) and (10 <= month) and (12 >= month)) or
-- any whole 1582 date from 1 october to 31 December or
            ((1926 == year) and (1 == month) and (1 ==
input.day)) or
                -- 1 January 1926 or
                ((1582 < year) and (1925 >= year))) then
-- any date 1 January 1583 – 31 December 1925
            tC0inS_date.inter_cal_cat = true;
-- set category flag true
    end
-- end temporary Julian / Gergorian calendar uncertainty detection
    if 1582 > tonumber(input.year) or 20 < tonumber(input.month) then
-- Julian calendar or season so &rft.date gets year only
        date = input.year;
        if 0 ~= input.year2 and input.year ~= input.year2 then
-- if a range, only the second year portion when not the same as range start
year
            date = string.format ('%.4d/%.4d',
tonumber(input.year), tonumber(input.year2))
-- assemble the
date range
        end
        if 20 < tonumber(input.month) then
-- if season or proper-name date
            local season = {[24]='winter', [21]='spring',
[22]='summer', [23]='fall', [33]='1', [34]='2', [35]='3', [36]='4',
[98]='Easter', [99]='Christmas'};
-- seasons lowercase, no autumn;
proper-names use title case
            if 0 == input.month2 then
-- single season date
                if 40 < tonumber(input.month) then
                    tC0inS_date.rftchron =
season[input.month];
-- proper-name

```



```

        date2 = string.format ('/%s-%.2d', input.year2,
tonumber(input.month2));    -- year and month
        else
            date2 = string.format ('/%s', input.year2);
-- just year
        end
    end
    tC0inS_date.rftdate = date .. date2;
-- date2 has the '/' separator
    return;
end

```

```

--[[-----< P A T T E R N S >-----
-----

```

this is the list of patterns for date formats that this module recognizes. Approximately the first half of these patterns represent formats that might be reformatted into another format. Those that might be reformatted have 'indicator' letters that identify the content of the matching capture: 'd' (day), 'm' (month), 'a' (anchor year), 'y' (year); second day, month, year have a '2' suffix.

These patterns are used for both date validation and for reformatting. This table should not be moved to ~/Configuration because changes to this table require changes to check_date() and to reformatter() and reformat_date()

```

]]

```

```

local patterns = {
-- year-initial numerical year-month-day
    ['ymd'] = {'^(%d%d%d)%-(%d)%-(%d)$', 'y', 'm', 'd'},
-- month-initial: month day, year
    ['Mdy'] = {'^(%D-) +([1-9]%d?), +((%d%d%d)%a?)$', 'm', 'd', 'a',
'y'},
-- month-initial day range: month day-day, year; days are separated by endash
    ['Md-dy'] = {'^(%D-) +([1-9]%d?)[%--]([1-9]%d?), +((%d%d%d)%a?)$',
'm', 'd', 'd2', 'a', 'y'},
-- day-initial: day month year
    ['dMy'] = {'^([1-9]%d?) +(%D-) +((%d%d%d)%a?)$', 'd', 'm', 'a',
'y'},
-- year-initial: year month day; day: 1 or 2 two digits, leading zero
allowed; not supported at en.wiki
--    ['yMd'] = {'^((%d%d%d)%a?) +(%D-) +(%d%d?)$', 'a', 'y', 'm', 'd'},
-- day-range-initial: day-day month year; days are separated by endash
    ['d-dMy'] = {'^([1-9]%d?)[%--]([1-9]%d?) +(%D-) +((%d%d%d)%a?)$',
'd', 'd2', 'm', 'a', 'y'},
-- day initial month-day-range: day month - day month year; uses spaced
endash

```

```
    ['dM-dMy'] = {'^([1-9]d?) +(%D-) +[%--] +([1-9]d?) +(%D-)
+((%d%d%d)d)%a?)}$', 'd', 'm', 'd2', 'm2', 'a', 'y'},
-- month initial month-day-range: month day – month day, year; uses spaced
endash
```

```
    ['Md-Mdy'] = {'^(%D-) +([1-9]d?) +[%--] +(%D-) +([1-9]d?),
+((%d%d%d)d)%a?)}$', 'm', 'd', 'm2', 'd2', 'a', 'y'},
-- day initial month-day-year-range: day month year - day month year; uses
spaced endash
```

```
    ['dMy-dMy'] = {'^([1-9]d?) +(%D-) +(%d%d%d)d +[%--] +([1-9]d?)
+(%D-) +((%d%d%d)d)%a?)}$', 'd', 'm', 'y', 'd2', 'm2', 'a', 'y2'},
-- month initial month-day-year-range: month day, year – month day, year;
uses spaced endash
```

```
    ['Mdy-Mdy'] = {'^(%D-) +([1-9]d?), +(%d%d%d)d +[%--] +(%D-)
+([1-9]d?), +((%d%d%d)d)%a?)}$', 'm', 'd', 'y', 'm2', 'd2', 'a', 'y2'},
```

```
-- these date formats cannot be converted, per se, but month name can be
rendered short or long
```

```
-- month/season year - month/season year; separated by spaced endash
```

```
    ['My-My'] = {'^(%D-) +(%d%d%d)d +[%--] +(%D-) +((%d%d%d)d)%a?)}$',
'm', 'y', 'm2', 'a', 'y2'},
```

```
-- month/season range year; months separated by endash
```

```
    ['M-My'] = {'^(%D-)[%--](%D-) +((%d%d%d)d)%a?)}$', 'm', 'm2', 'a',
'y'},
```

```
-- month/season year or proper-name year; quarter year when First Quarter
YYYY etc
```

```
    ['My'] = {'^([^-]d-) +((%d%d%d)d)%a?)}$', 'm', 'a', 'y'},
```

```
-- this way because endash is a member of %D; %D- will match January–March
2019 when it shouldn't
```

```
-- these date formats cannot be converted
```

```
--    ['Q,y'] = {'^(Q%a* +[1-4]), +((%d%d%d)d)%a?)}$',
```

```
-- Quarter n, yyyy
```

```
    ['Sy4-y2'] = {'^(%D-) +((%d%d)%d)d [%--] ((%d%d)%a?)}$',
```

```
-- special case Winter/Summer year-year (YYYY-YY); year separated with
unspaced endash
```

```
    ['Sy-y'] = {'^(%D-) +(%d%d%d)d [%--] ((%d%d%d)d)%a?)}$',
```

```
-- special case Winter/Summer year-year; year separated with unspaced endash
```

```
    ['y-y'] = {'^(%d%d%d)d? [%--] ((%d%d%d)d)%a?)}$',
```

```
-- year range: YYY-YYY or YYY-YYYY or YYYY-YYYY; separated by unspaced
endash; 100-9999
```

```
    ['y4-y2'] = {'^((%d%d)%d)d [%--] ((%d%d)%a?)}$',
```

```
-- year range: YYYY-YY; separated by unspaced endash
```

```
    ['y'] = {'^((%d%d%d)d)%a?)}$',
```

```
-- year; here accept either YYY or YYYY
```

```
}
```

```
--[[-----< C H E C K _ D A T E >-----
-----
```

Check date format to see that it is one of the formats approved by WP:DATESNO or WP:DATERANGE. Exception: only allowed range separator is endash. Additionally, check the date to see that it is a real date: no 31 in 30-day months; no 29 February when not a leap year. Months, both long-form and three character abbreviations, and seasons must be spelled correctly. Future years beyond next year are not allowed.

If the date fails the format tests, this function returns false and does not return values for anchor_year and C0inS_date. When this happens, the date parameter is used in the C0inS metadata and the CITEREF identifier gets its year from the year parameter if present otherwise CITEREF does not get a date value.

Inputs:

date_string - date string from date-holding parameters (date, year, accesdate, embargo, archivedate, etc.)

Returns:

false if date string is not a real date; else
true, anchor_year, C0inS_date
anchor_year can be used in CITEREF anchors
C0inS_date is ISO 8601 format date; see make_C0inS_date()

]]

```
local function check_date (date_string, param, tC0inS_date)
    local year;
-- assume that year2, months, and days are not used;
    local year2=0;
-- second year in a year range
    local month=0;
    local month2=0;
-- second month in a month range
    local day=0;
    local day2=0;
-- second day in a day range
    local anchor_year;
    local coins_date;

    if date_string:match (patterns['ymd'][1]) then
-- year-initial numerical year month day format
        year, month, day=date_string:match (patterns['ymd'][1]);
        if 12 < tonumber(month) or 1 > tonumber(month) or 1582 >
tonumber(year) or 0 == tonumber(day) then return false; end -- month
or day number not valid or not Gregorian calendar
        anchor_year = year;
-- elseif mw.usttring.match(date_string, patterns['Q,y'][1]) then
-- quarter n, year; here because much the same as Mdy
-- month, anchor_year, year=mw.usttring.match(date_string,
```

```

patterns['Q,y'][1]);
--         if not is_valid_year(year) then return false; end
--         month = get_quarter_number (month, param);
-- get quarter number or nil
--         if not month then return false; end
-- not valid whatever it is

        elseif mw.ustring.match(date_string, patterns['Mdy'][1]) then
-- month-initial: month day, year
        month, day, anchor_year, year=mw.ustring.match(date_string,
patterns['Mdy'][1]);
        month = get_month_number (month);
        if 0 == month then return false; end
-- return false if month text isn't one of the twelve months
        elseif mw.ustring.match(date_string, patterns['Md-dy'][1]) then
-- month-initial day range: month day–day, year; days are separated by endash
        month, day, day2, anchor_year,
year=mw.ustring.match(date_string, patterns['Md-dy'][1]);
        if tonumber(day) >= tonumber(day2) then return false; end
-- date range order is left to right: earlier to later; dates may not be the
same;
        month = get_month_number (month);
        if 0 == month then return false; end
-- return false if month text isn't one of the twelve months
        month2=month;
-- for metadata
        year2=year;

        elseif mw.ustring.match(date_string, patterns['dMy'][1]) then
-- day-initial: day month year
        day, month, anchor_year, year=mw.ustring.match(date_string,
patterns['dMy'][1]);
        month = get_month_number (month);
        if 0 == month then return false; end
-- return false if month text isn't one of the twelve months

--[[ NOT supported at en.wiki
        elseif mw.ustring.match(date_string, patterns['yMd'][1]) then
-- year-initial: year month day; day: 1 or 2 two digits, leading zero allowed
        anchor_year, year, month, day=mw.ustring.match(date_string,
patterns['yMd'][1]);
        month = get_month_number (month);
        if 0 == month then return false; end
-- return false if month text isn't one of the twelve months
-- end NOT supported at en.wiki ]]

        elseif mw.ustring.match(date_string, patterns['d-dMy'][1]) then
-- day-range-initial: day–day month year; days are separated by endash
        day, day2, month, anchor_year,
year=mw.ustring.match(date_string, patterns['d-dMy'][1]);
        if tonumber(day) >= tonumber(day2) then return false; end

```



```

-- date range order is left to right: earlier to later; dates may not be the
same;
        month = get_month_number (month);
        if 0 == month then return false; end
-- return false if month text isn't one of the twelve months
        month2=month;
-- for metadata
        year2=year;

        elseif mw.ustring.match(date_string, patterns['dM-dMy'][1]) then
-- day initial month-day-range: day month - day month year; uses spaced
endash
        day, month, day2, month2, anchor_year,
year=mw.ustring.match(date_string, patterns['dM-dMy'][1]);
        if (not is_valid_month_season_range(month, month2)) or not
is_valid_year(year) then return false; end        -- date range order is left
to right: earlier to later;
        month = get_month_number (month);
-- for metadata
        month2 = get_month_number (month2);
        year2=year;

        elseif mw.ustring.match(date_string, patterns['Md-Mdy'][1]) then
-- month initial month-day-range: month day - month day, year; uses spaced
endash
        month, day, month2, day2, anchor_year,
year=mw.ustring.match(date_string, patterns['Md-Mdy'][1]);
        if (not is_valid_month_season_range(month, month2, param)) or
not is_valid_year(year) then return false; end
        month = get_month_number (month);
-- for metadata
        month2 = get_month_number (month2);
        year2=year;

        elseif mw.ustring.match(date_string, patterns['dMy-dMy'][1]) then
-- day initial month-day-year-range: day month year - day month year; uses
spaced endash
        day, month, year, day2, month2, anchor_year,
year2=mw.ustring.match(date_string, patterns['dMy-dMy'][1]);
        if tonumber(year2) <= tonumber(year) then return false; end
-- must be sequential years, left to right, earlier to later
        if not is_valid_year(year2) or not
is_valid_month_range_style(month, month2) then return false; end
-- year2 no more than one year in the future; months same style
        month = get_month_number (month);
-- for metadata
        month2 = get_month_number (month2);

        elseif mw.ustring.match(date_string, patterns['Mdy-Mdy'][1]) then
-- month initial month-day-year-range: month day, year - month day, year;
uses spaced endash

```

```

        month, day, year, month2, day2, anchor_year,
year2=mw.ustring.match(date_string, patterns['Mdy-Mdy'][1]);
        if tonumber(year2) <= tonumber(year) then return false; end
-- must be sequential years, left to right, earlier to later
        if not is_valid_year(year2) or not
is_valid_month_range_style(month, month2) then return false; end
-- year2 no more than one year in the future; months same style
        month = get_month_number (month);
-- for metadata
        month2 = get_month_number (month2);

        elseif mw.ustring.match(date_string, patterns['Sy4-y2'][1]) then
-- special case Winter/Summer year-year (YYYY-YY); year separated with
unspaced endash
        local century;
        month, year, century, anchor_year,
year2=mw.ustring.match(date_string, patterns['Sy4-y2'][1]);
        if 'Winter' ~= month and 'Summer' ~= month then return false
end;
        -- 'month' can only be Winter or Summer
        anchor_year=year..'-'..anchor_year;
-- assemble anchor_year from both years
        year2 = century..year2;
-- add the century to year2 for comparisons
        if 1 ~= tonumber(year2) - tonumber(year) then return false;
end
        -- must be sequential years, left to right,
earlier to later
        if not is_valid_year(year2) then return false; end
-- no year farther in the future than next year
        month = get_season_number (month, param);

        elseif mw.ustring.match(date_string, patterns['Sy-y'][1]) then
-- special case Winter/Summer year-year; year separated with unspaced endash
        month, year, anchor_year, year2=mw.ustring.match(date_string,
patterns['Sy-y'][1]);
        if 'Winter' ~= month and 'Summer' ~= month then return false
end;
        -- 'month' can only be Winter or Summer
        anchor_year=year..'-'..anchor_year;
-- assemble anchor_year from both years
        if 1 ~= tonumber(year2) - tonumber(year) then return false;
end
        -- must be sequential years, left to right,
earlier to later
        if not is_valid_year(year2) then return false; end
-- no year farther in the future than next year
        month = get_season_number (month, param);
-- for metadata

        elseif mw.ustring.match(date_string, patterns['My-My'][1]) then
-- month/season year - month/season year; separated by spaced endash
        month, year, month2, anchor_year,
year2=mw.ustring.match(date_string, patterns['My-My'][1]);
        anchor_year=year..'-'..anchor_year;

```

```

-- assemble anchor_year from both years
    if tonumber(year) >= tonumber(year2) then return false; end
-- left to right, earlier to later, not the same
    if not is_valid_year(year2) then return false; end
-- no year farther in the future than next year
    if 0 ~= get_month_number(month) and 0 ~=
get_month_number(month2) and is_valid_month_range_style(month, month2) then
-- both must be month year, same month style
        month = get_month_number(month);
        month2 = get_month_number(month2);
    elseif 0 ~= get_season_number(month, param) and 0 ~=
get_season_number(month2, param) then -- both must be season year, not
mixed
        month = get_season_number(month, param);
        month2 = get_season_number(month2, param);
    else
        return false;
    end

    elseif mw.ustring.match(date_string, patterns['M-My'][1]) then
-- month/season range year; months separated by endash
        month, month2, anchor_year,
year=mw.ustring.match(date_string, patterns['M-My'][1]);
        if (not is_valid_month_season_range(month, month2, param)) or
(not is_valid_year(year)) then return false; end
        if 0 ~= get_month_number(month) then
-- determined to be a valid range so just check this one to know if month or
season
            month = get_month_number(month);
            month2 = get_month_number(month2);
        else
            month = get_season_number(month, param);
            month2 = get_season_number(month2, param);
        end
        year2=year;
    elseif mw.ustring.match(date_string, patterns['My'][1]) then
-- month/season/quarter/proper-name year
        month, anchor_year, year=mw.ustring.match(date_string,
patterns['My'][1]);
        if not is_valid_year(year) then return false; end
        month = get_element_number (month, param);
-- get month season quarter proper-name number or nil
        if not month then return false; end
-- not valid whatever it is

    elseif mw.ustring.match(date_string, patterns['y-y'][1]) then
-- Year range: YYY-YYY or YYY-YYYY or YYYY-YYYY; separated by unspaced
endash; 100-9999
        year, anchor_year, year2=mw.ustring.match(date_string,
patterns['y-y'][1]);
        anchor_year=year..'-'..anchor_year;

```

```

-- assemble anchor year from both years
    if tonumber(year) >= tonumber(year2) then return false; end
-- left to right, earlier to later, not the same
    if not is_valid_year(year2) then return false; end
-- no year farther in the future than next year

    elseif mw.ustring.match(date_string, patterns['y4-y2'][1]) then
-- Year range: YYYY-YY; separated by unspaced endash
    local century;
    year, century, anchor_year,
year2=mw.ustring.match(date_string, patterns['y4-y2'][1]);
    anchor_year=year..'-'..anchor_year;
-- assemble anchor year from both years
    if 13 > tonumber(year2) then return false; end
-- don't allow 2003-05 which might be May 2003
    year2 = century..year2;
-- add the century to year2 for comparisons
    if tonumber(year) >= tonumber(year2) then return false; end
-- left to right, earlier to later, not the same
    if not is_valid_year(year2) then return false; end
-- no year farther in the future than next year

    elseif mw.ustring.match (date_string, patterns['y'][1]) then
-- year; here accept either YYYY or YYYY
    anchor_year, year=mw.ustring.match (date_string,
patterns['y'][1]);
    if false == is_valid_year(year) then
        return false;
    end

    else
        return false;
-- date format not one of the MOS:DATE approved formats
    end

    if 'access-date' == param then
-- test accesdate here because we have numerical date parts
    if 0 ~= year and 0 ~= month and 0 ~= day and
-- all parts of a single date required
    0 == year2 and 0 == month2 and 0 == day2 then
-- none of these; accesdate must not be a range
    if not is_valid_accesdate (year..'-'
'..month..'-'..day) then
        return false;
-- return false when accesdate out of bounds
    end

    else
        return false;
-- return false when accesdate is a range of two dates
    end

end

```

```

    local result=true;
-- check whole dates for validity; assume true because not all dates will go
through this test
    if 0 ~= year and 0 ~= month and 0 ~= day and 0 == year2 and 0 ==
month2 and 0 == day2 then          -- YMD (simple whole date)
        result=is_valid_date(year,month,day);

        elseif 0 ~= year and 0 ~= month and 0 ~= day and 0 == year2 and 0 ==
month2 and 0 ~= day2 then          -- YMD-d (day range)
            result=is_valid_date(year,month,day);
            result=result and is_valid_date(year,month,day2);

            elseif 0 ~= year and 0 ~= month and 0 ~= day and 0 == year2 and 0 ~=
month2 and 0 ~= day2 then          -- YMD-md (day month range)
                result=is_valid_date(year,month,day);
                result=result and is_valid_date(year,month2,day2);

                elseif 0 ~= year and 0 ~= month and 0 ~= day and 0 ~= year2 and 0 ~=
month2 and 0 ~= day2 then          -- YMD-ymd (day month year range)
                    result=is_valid_date(year,month,day);
                    result=result and is_valid_date(year2,month2,day2);
                end
            if false == result then return false; end

            if nil ~= tC0inS_date then
-- this table only passed into this function when testing |date= parameter
values
                make_C0inS_date ({year=year, month=month, day=day,
year2=year2, month2=month2, day2=day2}, tC0inS_date);          -- make an ISO
8601 date string for C0inS
                end
                return true, anchor_year;
-- format is good and date string represents a real date
end

```

```

--[[-----< D A T E S >-----
-----

```

Cycle the date-holding parameters in passed table `date_parameters_list` through `check_date()` to check compliance with `MOS:DATE`. For all valid dates, `check_date()` returns `true`. The `|date=` parameter test is unique, it is the only date holding parameter from which values for `anchor_year` (used in CITEREF identifiers) and `C0inS_date` (used in the `C0inS` metadata) are derived. The `|date=` parameter is the only date-holding parameter that is allowed to contain the no-date keywords "n.d." or "nd" (without quotes).

Unlike most error messages created in this module, only one error message is created by this function. Because all of the date holding parameters are

processed serially,
a single error message is created as the dates are tested.

```
]]

local function dates(date_parameters_list, tC0inS_date)
    local anchor_year;          -- will return as nil if the date
being tested is not |date=
    local C0inS_date;          -- will return as nil if the date
being tested is not |date=
    local embargo_date;
-- if embargo date is a good dmy, mdy, ymd date then holds original value
else reset to 9999
    local error_message = "";
    local good_date = false;
    for k, v in pairs(date_parameters_list) do
-- for each date-holding parameter in the list
        if is_set(v.val) then
-- if the parameter has a value
            v.val = mw.ustr.gsub (v.val, '%d',
cfg.date_names.local_digits);    -- translate 'local' digits to Western
0-9
                if v.val:match("^c%. [1-9]%d%d%d?%a?$") then
-- special case for c. year or with or without CITEREF disambiguator - only
|date= and |year=
                    local year = v.val:match("c%.
([1-9]%d%d%d?)%a?");          -- get the year portion
so it can be tested
                    if 'date'==k then
                        anchor_year, C0inS_date =
v.val:match("((c%. [1-9]%d%d%d?)%a?");    -- anchor year and C0inS_date
only from |date= parameter
                        good_date = is_valid_year(year);
                    elseif 'year'==k then
                        good_date = is_valid_year(year);
                    end
                elseif 'date'==k then
-- if the parameter is |date=
                    if v.val:match("^n%.d%.%a?$") then
-- if |date=n.d. with or without a CITEREF disambiguator
                        good_date, anchor_year, C0inS_date =
true, v.val:match("((n%.d%.)%a?");    --"n.d."; no error when date
parameter is set to no date
                    elseif v.val:match("^nd%a?$") then
-- if |date=nd with or without a CITEREF disambiguator
                        good_date, anchor_year, C0inS_date =
true, v.val:match("((nd)%a?");    --"nd"; no error when date
parameter is set to no date
                    else
                        good_date, anchor_year, C0inS_date =
check_date (v.val, k, tC0inS_date);    -- go test the date

```

```

        end
        elseif 'year'==k then
-- if the parameter is |year= it should hold only a year value
            if v.val:match("^[1-9]%d%d%d?%a?$") then
-- if |year= 3 or 4 digits only with or without a CITEREF disambiguator
                good_date, anchor_year, COinS_date =
true, v.val:match("((%d+)%a?");
            end
        elseif 'embargo'==k then
-- if the parameter is |embargo=
            good_date = check_date (v.val, k);
-- go test the date
            if true == good_date then
-- if the date is a valid date
                good_date, embargo_date =
is_valid_embargo_date (v.val); -- is |embargo= date a single dmy, mdy,
or ymd formatted date? yes:returns embargo; no: returns 9999
            end
        else
-- any other date-holding parameter
            good_date = check_date (v.val, k);
-- go test the date
        end
        if false==good_date then
-- assemble one error message so we don't add the tracking category multiple
times
            if is_set(error_message) then
-- once we've added the first portion of the error message ...
                error_message=error_message .. ", ";
-- ... add a comma space separator
            end
            error_message=error_message .. "&#124;" ..
v.name .. "="; -- add the failed parameter
        end
    end
end
end
return anchor_year, embargo_date, error_message;
-- and done
end

```

```

--[[-----< Y E A R _ D A T E _ C H E C K >-----
-----

```

Compare the value provided in |year= with the year value(s) provided in |date=. This function returns a numeric value:

- 0 - year value does not match the year value in date
- 1 - (default) year value matches the year value in date or one of the year values when date contains two years
- 2 - year value matches the year value in date when date is in the form YYYY-MM-DD and year is disambiguated (|year=YYYYx)

```

]]

local function year_date_check (year_string, date_string)
    local year;
    local date1;
    local date2;
    local result = 1;
-- result of the test; assume that the test passes
    year = year_string:match ('(%d%d%d%d?)');

    if date_string:match ('%d%d%d%d%-%d%d%-%d%d') and year_string:match
('%d%d%d%d%a') then      --special case where both date and year are
required YYYY-MM-DD and YYYYx
        date1 = date_string:match ('(%d%d%d%d)');
        year = year_string:match ('(%d%d%d%d)');
        if year ~= date1 then
            result = 0;
-- years don't match
        else
            result = 2;
-- years match; but because disambiguated, don't add to maint cat
        end
        elseif date_string:match ("%d%d%d%d??.-%d%d%d%d?") then
-- any of the standard range formats of date with two three- or four-digit
years
            date1, date2 = date_string:match ("(%d%d%d%d?).-
(%d%d%d%d?)");
            if year ~= date1 and year ~= date2 then
                result = 0;
            end

            elseif mw.ustring.match(date_string, "%d%d%d%d[%--]%d%d") then
-- YYYY-YY date ranges
            local century;
            date1, century, date2 = mw.ustring.match(date_string,
"((%d%d)%d%d)[%--]+(%d%d)");
            date2 = century..date2;
-- convert YY to YYYY
            if year ~= date1 and year ~= date2 then
                result = 0;
            end

            elseif date_string:match ("%d%d%d%d?") then
-- any of the standard formats of date with one year
            date1 = date_string:match ("(%d%d%d%d?)");
            if year ~= date1 then
                result = 0;
            end
        else
            result = 0;
-- no recognizable year in date

```



```
        end
        return result;
end
```

```
--[[-----< R E F O R M A T T E R >-----
-----
```

reformat 'date' into new format specified by format_param if pattern_idx (the current format of 'date') can be reformatted. Does the grunt work for reformat_dates().

The table re_formats maps pattern_idx (current format) and format_param (desired format) to a table that holds:

- format string used by string.format()
- identifier letters ('d', 'm', 'y', 'd2', 'm2', 'y2') that serve as indexes into a table t{} that holds captures from mw.ustring.match() for the various date parts specified by patterns[pattern_idx][1]

Items in patterns{} have the general form:

- ['ymd'] = {'^(%d%d%d%d)%-(%d%d)%-(%d%d)\$', 'y', 'm', 'd'}, where:
 - ['ymd'] is pattern_idx
 - patterns['ymd'][1] is the match pattern with captures for mw.ustring.match()
 - patterns['ymd'][2] is an indicator letter identifying the content of the first capture
 - patterns['ymd'][3] ... the second capture etc

when a pattern matches a date, the captures are loaded into table t{} in capture order using the identifier characters as indexes into t{} For the above, a ymd date is in t{} as:
t.y = first capture (year), t.m = second capture (month), t.d = third capture (day)

To reformat, this function is called with the pattern_idx that matches the current format of the date and with format_param set to the desired format. This function loads table t{} as described and then calls string.format() with the format string specified by re_format[pattern_idx][format_param][1] using values taken from t{} according to the capture identifier letters specified by patterns[pattern_idx][format_param][n] where n is 2..

```
]]
```

```
local re_formats = {
    ['ymd'] = {
-- date format is ymd; reformat to:
        ['mdy'] = {'%s %s, %s', 'm', 'd', 'y'},
-- |df=mdy
```

```

        ['dmy'] = {'%s %s %s', 'd', 'm', 'y'},
-- |df=dmy
        ['yMd'] = {'%s %s %s', 'y', 'm', 'd'},
-- |df=yMd; not supported at en.wiki
    },
    ['Mdy'] = {
-- date format is Mdy; reformat to:
        ['mdy'] = {'%s %s, %s', 'm', 'd', 'y'},
-- for long/short reformatting
        ['dmy'] = {'%s %s %s', 'd', 'm', 'y'},
-- |df=dmy
        ['ymd'] = {'%s-%s-%s', 'y', 'm', 'd'},
-- |df=ymd
        ['yMd'] = {'%s %s %s', 'y', 'm', 'd'},
-- |df=yMd; not supported at en.wiki
    },
    ['dMy'] = {
-- date format is dMy; reformat to:
        ['dmy'] = {'%s %s %s', 'd', 'm', 'y'},
-- for long/short reformatting
        ['mdy'] = {'%s %s, %s', 'm', 'd', 'y'},
-- |df=mdy
        ['ymd'] = {'%s-%s-%s', 'y', 'm', 'd'},
-- |df=ymd
        ['yMd'] = {'%s %s %s', 'y', 'm', 'd'},
-- |df=yMd; not supported at en.wiki
    },
    ['Md-dy'] = {
-- date format is Md-dy; reformat to:
        ['mdy'] = {'%s %s-%s, %s', 'm', 'd', 'd2', 'y'},
-- for long/short reformatting
        ['dmy'] = {'%s-%s %s %s', 'd', 'd2', 'm', 'y'},
-- |df=dmy -> d-dMy
    },
    ['d-dMy'] = {
-- date format is d-d>y; reformat to:
        ['dmy'] = {'%s-%s %s %s', 'd', 'd2', 'm', 'y'},
-- for long/short reformatting
        ['mdy'] = {'%s %s-%s, %s', 'm', 'd', 'd2', 'y'},
-- |df=mdy -> Md-dy
    },
    ['dM-dMy'] = {
-- date format is dM-dMy; reformat to:
        ['dmy'] = {'%s %s - %s %s %s', 'd', 'm', 'd2', 'm2', 'y'},
-- for long/short reformatting
        ['mdy'] = {'%s %s - %s %s, %s', 'm', 'd', 'm2', 'd2', 'y'},
-- |df=mdy -> Md-Mdy
    },
    ['Md-Mdy'] = {
-- date format is Md-Mdy; reformat to:
        ['mdy'] = {'%s %s - %s %s, %s', 'm', 'd', 'm2', 'd2', 'y'},

```

```

-- for long/short reformatting
    ['dmy'] = {'%s %s - %s %s %s', 'd', 'm', 'd2', 'm2', 'y'},
-- |df=dmy -> dM-dMy
    },
    ['dMy-dMy'] = {
-- date format is dMy-dMy; reformat to:
    ['dmy'] = {'%s %s %s - %s %s %s', 'd', 'm', 'y', 'd2', 'm2',
'y2'},
        -- for long/short reformatting
    ['mdy'] = {'%s %s, %s - %s %s, %s', 'm', 'd', 'y', 'm2',
'd2', 'y2'},
        -- |df=mdy -> Mdy-Mdy
    },
    ['Mdy-Mdy'] = {
-- date format is Mdy-Mdy; reformat to:
    ['mdy'] = {'%s %s, %s - %s %s, %s', 'm', 'd', 'y', 'm2',
'd2', 'y2'},
        -- for long/short reformatting
    ['dmy'] = {'%s %s %s - %s %s %s', 'd', 'm', 'y', 'd2', 'm2',
'y2'},
        -- |df=dmy -> dMy-dMy
    },
    ['My-My'] = {
-- these for long/short reformatting
    ['any'] = {'%s %s - %s %s', 'm', 'y', 'm2', 'y2'},
-- dmy/mdy agnostic
    },
    ['M-My'] = {
-- these for long/short reformatting
    ['any'] = {'%s-%s %s', 'm', 'm2', 'y'},
-- dmy/mdy agnostic
    },
    ['My'] = {
-- these for long/short reformatting
    ['any'] = {'%s %s', 'm', 'y'},
-- dmy/mdy agnostic
    },
    ['yMd'] = {
-- not supported at en.wiki
    ['mdy'] = {'%s %s, %s', 'm', 'd', 'y'},
-- |df=mdy
    ['dmy'] = {'%s %s %s', 'd', 'm', 'y'},
-- |df=dmy
    ['ymd'] = {'%s-%s-%s', 'y', 'm', 'd'},
-- |df=ymd
    },
}

```

```

local function reformatter (date, pattern_idx, format_param, mon_len)
    if not in_array (pattern_idx, {'ymd', 'Mdy', 'Md-dy', 'dMy', 'yMd',
'd-dMy', 'dM-dMy', 'Md-Mdy', 'dMy-dMy', 'Mdy-Mdy', 'My-My', 'M-My', 'My'})
then
        return;
-- not in this set of date format patterns then not a reformattable date

```

```

        end
        if 'ymd' == format_param and in_array (pattern_idx, {'ymd', 'Md-dy',
'd-dMy', 'dM-dMy', 'Md-Mdy', 'dMy-dMy', 'Mdy-Mdy', 'My-My', 'M-My', 'My'})
then
            return;
-- ymd date ranges not supported at en.wiki; no point in reformatting ymd to
ymd
        end

        if in_array (pattern_idx, {'My', 'M-My', 'My-My'}) then
-- these are not dmy/mdy so can't be 'reformatted' into either
            format_param = 'any';
-- so format-agnostic
        end

-- if 'yMd' == format_param and in_array (pattern_idx, {'yMd', 'Md-dy',
'd-dMy', 'dM-dMy', 'Md-Mdy', 'dMy-dMy', 'Mdy-Mdy'}) then -- not
supported at en.wiki
        if 'yMd' == format_param then
-- not supported at en.wiki
            return;
-- not a reformattable date
        end
        local c1, c2, c3, c4, c5, c6, c7;
-- these hold the captures specified in patterns[pattern_idx][1]
        c1, c2, c3, c4, c5, c6, c7 = mw.ustring.match (date,
patterns[pattern_idx][1]); -- get the captures

        local t = {
-- table that holds k/v pairs of date parts from the captures and
patterns[pattern_idx][2..]
            [patterns[pattern_idx][2]] = c1;
-- at minimum there is always one capture with a matching indicator letter
            [patterns[pattern_idx][3] or 'x'] = c2;
-- patterns can have a variable number of captures; each capture requires an
indicator letter;
            [patterns[pattern_idx][4] or 'x'] = c3;
-- where there is no capture, there is no indicator letter so n in
patterns[pattern_idx][n] will be nil;
            [patterns[pattern_idx][5] or 'x'] = c4;
-- the 'x' here spoofs an indicator letter to prevent 'table index is nil'
error
            [patterns[pattern_idx][6] or 'x'] = c5;
            [patterns[pattern_idx][7] or 'x'] = c6;
            [patterns[pattern_idx][8] or 'x'] = c7;
        };

        if t.a then
-- if this date has an anchor year capture
            t.y = t.a;
-- use the anchor year capture when reassembling the date

```

```

end

    if tonumber(t.m) then
-- if raw month is a number (converting from ymd)
        if 's' == mon_len then
-- if we are to use abbreviated month names
            t.m = cfg.date_names['inv_local_s'][tonumber(t.m)];
-- convert it to a month name
        else
            t.m = cfg.date_names['inv_local_l'][tonumber(t.m)];
-- convert it to a month name
        end
        t.d = t.d:gsub ('0(%d)', '%1');
-- strip leading '0' from day if present
        elseif 'ymd' == format_param then
-- when converting to ymd
            if 1582 > tonumber(t.y) then
-- ymd format dates not allowed before 1582
                return;
            end
            t.m = string.format ('%02d', get_month_number (t.m));
-- make sure that month and day are two digits
            t.d = string.format ('%02d', t.d);
        elseif mon_len then
-- if mon_len is set to either 'short' or 'long'
            for _, mon in ipairs ({'m', 'm2'}) do
-- because there can be two month names, check both
                if t[mon] then
                    t[mon] = get_month_number (t[mon]);
-- get the month number for this month (is length agnostic)
                    if 0 == t[mon] then return; end
-- seasons and named dates can't be converted
                    t[mon] = (('s' == mon_len) and
cfg.date_names['inv_local_s'][t[mon]]) or
cfg.date_names['inv_local_l'][t[mon]];          -- fetch month name according
to length
                end
            end
        end
    end

    local new_date = string.format
(re_formats[pattern_idx][format_param][1],      -- format string
    t[re_formats[pattern_idx][format_param][2]],
-- named captures from t{}
    t[re_formats[pattern_idx][format_param][3]],
    t[re_formats[pattern_idx][format_param][4]],
    t[re_formats[pattern_idx][format_param][5]],
    t[re_formats[pattern_idx][format_param][6]],
    t[re_formats[pattern_idx][format_param][7]],
    t[re_formats[pattern_idx][format_param][8]]
);

```

```
        return new_date;
end
```

```
--[[-----< R E F O R M A T _ D A T E S >-----
-----
```

Reformats existing dates into the format specified by format.

format is one of several manual keywords: dmy, dmy-all, mdy, mdy-all, ymd, ymd-all. The -all version includes access- and archive-dates; otherwise these dates are not reformatted.

This function allows automatic date formatting. In ~/Configuration, the article source is searched for one of the {{use xxx dates}} templates. If found, xxx becomes the global date format as xxx-all. If |csl-dates= in {{use xxx dates}} has legitimate value then that value determines how csl|2 dates will be rendered. Legitimate values for |csl-dates= are:

- l - all dates are rendered with long month names
- ls - publication dates use long month names; access-/archive-dates use abbreviated month names
- ly - publication dates use long month names; access-/archive-dates rendered in ymd format
- s - all dates are rendered with abbreviated (short) month names
- sy - publication dates use abbreviated month names; access-/archive-dates rendered in ymd format
- y - all dates are rendered in ymd format

the format argument for automatic date formatting will be the format specified by {{use xxx dates}} with the value supplied by |csl-dates so one of: xxx-l, xxx-ls, xxx-ly, xxx-s, xxx-sy, xxx-y, or simply xxx (|csl-dates= empty, omitted, or invalid) where xxx shall be either of dmy or mdy.

dates are extracted from date_parameters_list, reformatted (if appropriate), and then written back into the list in the new format. Dates in date_parameters_list are presumed here to be valid (no errors). This function returns true when a date has been reformatted, false else. Actual reformatting is done by reformatter().

```
]]
```

```
local function reformat_dates (date_parameters_list, format)
    local all = false;
-- set to false to skip access- and archive-dates
    local len_p = 'l';
-- default publication date length shall be long
    local len_a = 'l';
```



```

new_date =
reformatter (param_val.val, pattern_idx, format, len_p);
end
if new_date then
-- set when date was reformatted
date_parameters_list[param_name].val = new_date;      -- update date in
date list
result = true;
-- and announce that changes have been made
end
end
end
end
end
end
end
return result;
-- declare boolean result and done
end

```

```

--[[-----< D A T E _ H Y P H E N _ T O _ D A S H >-----
-----

```

Loops through the list of date-holding parameters and converts any hyphen to an ndash. Not called if the cs1|2 template has any date errors.

Modifies the date_parameters_list and returns true if hyphens are replaced, else returns false.

```

]]

local function date_hyphen_to_dash (date_parameters_list)
  local result = false;
  local n;
  for param_name, param_val in pairs(date_parameters_list) do
-- for each date-holding parameter in the list
    if is_set (param_val.val) then
      if not mw.ustr.match (param_val.val, '%d%d%d%d%-
%d%d%-%d%d') then      -- for those that are not ymd dates (ustr because
here digits may not be western)
        param_val.val, n = param_val.val:gsub ('%- ',
'-');
        -- replace any hyphen with ndash
        if 0 ~= n then
          date_parameters_list[param_name].val
= param_val.val;      -- update the list
          result = true;
        end
      end
    end
  end
end
end
end

```



```

        return result;
-- so we know if any hyphens were replaced
end

```

```

--[[-----< D A T E _ N A M E _ X L A T E >-----
-----

```

Attempts to translate English month names to local-language month names using names supplied by MediaWiki's date parser function. This is simple name-for-name replacement and may not work for all languages.

if xlat_dig is true, this function will also translate western (English) digits to the local language's digits. This will also translate ymd dates.

```

]]

```

```

local function date_name_xlate (date_parameters_list, xlt_dig)
    local xlate;
    local mode;
-- long or short month names
    local modified = false;
    local date;
    for param_name, param_val in pairs(date_parameters_list) do
-- for each date-holding parameter in the list
        if is_set(param_val.val) then
-- if the parameter has a value
            date = param_val.val;
            for month in mw.ustr.gmatch (date, '%a+') do
-- iterate through all dates in the date (single date or date range)
                if cfg.date_names.en.long[month] then
                    mode = 'F';
-- English name is long so use long local name
                elseif cfg.date_names.en.short[month] then
                    mode = 'M';
-- English name is short so use short local name
                else
                    mode = nil;
-- not an English month name; could be local language month name or an
English season name
                    end
                    if mode then
-- might be a season
                        xlate = lang_object:formatDate(mode,
'1' .. month);
-- translate the month name to this
local language
                        date = mw.ustr.gsub (date, month,
xlate);
-- replace the English with the
translation

```

```

= date;
                                date_parameters_list[param_name].val
                                -- save the translated date
                                modified = true;
                                end
                                end
                                if xlt_dig then
-- shall we also translate digits?
                                date = date:gsub ('%d',
cfg.date_names.xlate_digits);                                -- translate digits from
western to 'local digits'
                                date_parameters_list[param_name].val = date;
-- save the translated date
                                modified = true;
                                end
                                end
                                end
                                end
                                return modified;
                                end

```

```

--[[-----< S E T _ S E L E C T E D _ M O D U L E S >-----
-----

```

Sets local imported functions table to same (live or sandbox) as that used by the other modules.

```

]]

```

```

local function set_selected_modules (cfg_table_ptr, utilities_page_ptr)
    is_set = utilities_page_ptr.is_set;
-- import functions from selected Module:Citation/CS1/Utilities module
    in_array = utilities_page_ptr.in_array;
-- import functions from selected Module:Citation/CS1/Utilities module
    cfg = cfg_table_ptr;
-- import tables from selected Module:Citation/CS1/Configuration
end

```

```

--[[-----< E X P O R T E D   F U N C T I O N S >-----
-----

```

```

]]

```

```

return {
-- return exported functions
    dates = dates,
    year_date_check = year_date_check,
    reformat_dates = reformat_dates,
    date_hyphen_to_dash = date_hyphen_to_dash,
    date_name_xlate = date_name_xlate,
    set_selected_modules = set_selected_modules

```

}

accreted sediment in a river course or estuary, including both lateral (point-bars) and medial (braid-bars). Chars (or sand bars) emerge as islands within the river channel (island chars) or as attached land to the riverbanks (attached chars), create new opportunities for temporary settlements and agriculture.

Retrieved from

"https://www.bluegoldwiki.com/index.php?title=Module:Citation/CS1/Date_validation&oldid=3522"

Namespaces

- [Module](#)
- [Discussion](#)

Variants

This page was last edited on 23 August 2020, at 06:04.

Blue Gold Program Wiki

The wiki version of the Lessons Learnt Report of the Blue Gold program, documents the experiences of a technical assistance (TA) team working in a development project implemented by the Bangladesh Water Development Board (BWDB) and the Department of Agricultural Extension (DAE) over an eight+ year period from March 2013 to December 2021. The wiki lessons learnt report (LLR) is intended to complement the BWDB and DAE project completion reports (PCRs), with the aim of recording lessons learnt for use in the design and implementation of future interventions in the coastal zone.

- [Privacy policy](#)
- [About Blue Gold Program Wiki](#)
- [Disclaimers](#)

Developed and maintained by Big Blue Communications for Blue Gold Program



[Blue Gold Program Wiki](#)