Basic Symmetry454 and Symmetry010 Calendar Arithmetic

- The 52/293 Northward Equinox Leap Cycle
- The 69/389 North Solstice Leap Cycle
- Inter-conversion of dates with other calendars
- How to write computer programs for implementing the Symmetry454 and Symmetry010 calendars

Home Page on the Web: <http://individual.utoronto.ca/kalendis/>

Created by Dr. Irvin L. Bromberg

University of Toronto, Canada

Version 12.08(328), Aug 24, 2019 (Symmetry454) = Aug 22, 2019 (Symmetry010) = Aug 21, 2019 (Gregorian)

Table of Contents

The Symmetry454 and Symmetry010 calendars

This document primarily discusses the arithmetic of the symmetrical 4+**5**+4 **weeks** per quarter structure of the Symmetry454 calendar, including all arithmetic formulas and functions that are needed for computer implementation. Some people prefer the nearly equal month lengths of the 30+**31**+30 **days** per quarter Symmetry010 calendar structure, so the arithmetic for that variant is also included.

Unless otherwise specified, the discussion and formulas applies equally to both Symmetry calendar variants, even though the text will almost always refer only to the Symmetry454 calendar.

Where a topic or formula applies only to the recommended Symmetry454 calendar, it is labeled with "4+**5**+4", whereas that which applies only to the Symmetry010 calendar variant is labeled with "30+**31**+30".

There are very few places where their arithmetic differs.

Leap week appended to December, or stand-alone after December

The recommended handling for the Symmetry454 leap week is to append it as a 5th week of December, where it is no more exceptional that the preceding 5-week month of November.

Appending a leap week to the December of the Symmetry010 calendar makes that month 30+7=37 days, which is quite exceptional when compared to the normal 30 or 31 days per month. Therefore, some people prefer to see the leap week stand alone at the end of the year, like a 13th "mini-month", dubbed "*Irvember*" by Yoel Berznoger of Thornhill, Canada. Either way, the leap week is always the 53rd ordinal week of the calendar year.

The arithmetic herein shows how to handle the leap week either way. The recommended implementation for Symmetry454 has the leap week appended to December, whereas the recommended implementation for Symmetry010 has the leap week standing alone after December (to avoid having 37 days in December).

Warning: "Stick to the Script"

Symmetry454 and Symmetry010 calendar arithmetic is very simple, but there is a tendency for those who are programming their first implementation of these calendars to immediately cut corners that may suffice for a limited range of dates, or to skip thorough validation of their implementation.

Please don't deviate from the arithmetic outlined herein. Please "stick to the script". Don't try to invent your own arithmetic using novel expressions. There is no reason to do so, because this arithmetic is in the public domain, royalty free. The algorithm steps documented herein were carefully designed for efficiency, simplicity, and clarity of program code, and were thoroughly validated. Cutting corners will most likely result in harder-to-read programs that are more difficult to maintain and troubleshoot. In all probability a novel expression intended to "simplify" the arithmetic documented herein will actually prove to function erroneously under specific circumstances. It is just not worth wasting the time on the trouble that will make for you.

For most of the arithmetic expressions documented herein, simple examples are provided that you can use to verify your understanding, but that alone is not enough. For every algorithm that you implement, it is essential that you thoroughly validate its correct operation as a package. For calendrical calculations, such validations must check **thousands** of dates, either sequentially or distributed randomly in time. Every calendar date must convert to a unique fixed day number (see the heading "**Fixed day numbers, astronomy and modern computer operating systems**", below), and every fixed day number must convert to a unique calendar date, with zero "holes", which are unused fixed day numbers, or unused but otherwise valid calendar dates.

To automatically validate your implementation, set up a loop that iteratively converts either a sequentially or randomly chosen **fixed day number** to a calendar date and then back to a fixed day number. If the fixed day number that comes back differs from the original then the implemented algorithm is defective.

Then set up another loop that iteratively converts either a sequentially or randomly chosen **calendar date** to a fixed day number, taking care never to choose an illegal calendar date (month number too high, day number too high for the selected month, or date in a leap week for a year that has no leap week), and then use your algorithm to convert the fixed day number back to a calendar date. If the date that comes back differs from the original then the implemented algorithm is defective.

Verification of calendar arithmetic: the *Kalendis* freeware computer program

The *Kalendis* calendar calculator for Windows is freely available from the Symmetry454 web site at <http://individual.utoronto.ca/kalendis/>. *Kalendis* can list Symmetry454 or Symmetry010 leap years, interconverts dates to or from a variety of other calendars, and the user can experiment with built-in options.

For those who choose to implement the arithmetic documented herein, *Kalendis* is a useful tool for verifying proper error-free operation.

Understanding Floor $\lfloor n \rfloor$ and Ceiling $\lceil n \rceil$ Brackets: *ceiling*( ), *floor*( ), *quotient*( )

This document uses symbolic brackets with "feet" or with "hands". The brackets with "feet" only on the bottom are standard mathematical symbols for the *floor*(*n*) function, which gives the greatest integer that is less than or equal to *n*. In most computer programming languages this is identical to the INT(*n*) function, but for negative values of n this is not identical to the FIX(*n*) or TRUNC(*n*) functions. If your computer programming language has an INT(*n*) function that unfortunately truncates fractions the way that the FIX(*n*) or TRUNC(*n*) functions do then you need to use or implement a proper *floor*(*n*) function instead of INT(*n*).

When the floor brackets surround a fraction, as in $\lfloor n/d \rfloor$, divide the numerator n by the denominator d, discard the remainder and keep the quotient. Sometimes this is written as *quotient*($n \div d$) or *quotient*(n, d).

Calendrical calculations make frequent use of the *floor*(*n*) or *quotient*(*n*) functions.

Conversely the brackets with "hands" only on the top are standard mathematical symbols for the *ceiling*(*n*) function, which gives the least integer that is greater than or equal to *n*.

Note that $\lfloor n \rfloor \equiv -\lceil -n \rceil$ and the converse $\lceil n \rceil \equiv -\lfloor -n \rfloor$ is also true.

In many computer programming languages, displayed floating point numeric precision has fewer significant figures than internal precision. For example Microsoft Visual Basic uses up to 14 significant figures for displaying or printing double precision floating point values but 18 significant figures are carried internally through the floating point computation hardware. This can cause a number to look like an integer when displayed, for example 5, but it could really have 4.9999999999999998 as its internal representation, and then when applying the INT(*n*) function to it the value returned will be 4 instead of the expected 5. There is a high risk of such errors if single precision floating point calculations are employed — not recommended!

One way to avoid this problem in Visual Basic is to explicitly convert the number to its string representation, which will round its value to that which can be externally represented, then convert it back again to a double precision floating point number, automatically without the extra internal significant figures. After that a value that looks like an integer will behave like an integer when applying the INT(*n*) function to it.

Another way to avoid this problem is to carry out all calculations using exact arbitrary precision.

Functions for calculating remainders: *modulus*( ) and *amod*( )

Although many computer languages have a built-in *x* MOD *y* operator for returning the remainder from dividing *x* by *y*, many, including Microsoft's Visual Basic, return the wrong results when *x* is negative, or when either operand is not an integer. Therefore, I will use instead the following remainder functions (see reference 1):

$$modulus(x , y) = x - y \left\lfloor \frac{x}{y} \right\rfloor$$

The *modulus* function divides *x* by *y* and returns the remainder. We don't call this the *remainder*( ) function because in mathematics it has other applications in *modulo* arithmetic, where it isn't merely a remainder from a division operation. If *x* is divisible by *y* without any remainder then the *modulus*( ) function returns zero.

The *amod* (adjusted *modulus*) function is the same as the *modulus* function except that if *x* is divisible by *y* without any remainder then the *amod*( ) function returns the divisor *y* instead of zero.

$$amod(x , y) = y + modulus(x , -y)$$

In computer languages that support type declaration of parameters and return values, declare both parameters and the return value of both *modulus*( ) and *amod*( ) as double-precision floating point (or better).

Conversion of Symmetry454 or Symmetry010 dates to or from other calendars

In *Calendrical Calculations: Ultimate Edition* (CCUE), published by Cambridge University Press in late 2018, authors Edward M. Reingold and Nachum Dershowitz extensively documented computer algorithms for interconversion of almost every calendar used today or in the past (see references 1 and 2). The text includes calendar arithmetic functions in mathematical notation and the appendix lists source code in the LISP computer programming language, which they employ because of its standard arbitrary precision calculation engine. A CDROM disc including the source code in the Java and Mathematica computer programming languages accompanied the Second Edition ("Millennium") of this book. For further information, errata, and access to their *Calendrica* Java applet see <http://calendarists.com/>.

In the discussion of Symmetry454 calendar arithmetic below I have tried to follow the style of CCUE. I highly recommend reading the following CCUE chapters: *Introduction*, *The Gregorian Calendar*, *The Julian Calendar*, *The ISO Calendar*, *The Ecclesiastical Calendars*, *Time and Astronomy*, and the applicable errata on-line at their web site. Where a formula could have been written in multiple ways I have tried to select the method that I considered easiest to understand even if a less obvious formula might require fewer lines or computes faster.

The Symmetry454 and Symmetry010 calendar epoch: *SymEpoch*

The Symmetry454 and Symmetry010 calendars share the same epoch as the Gregorian calendar, starting on Monday, January 1, 1 AD. This is the same epoch as that of the ISO calendar and the Revised Julian calendar.

Symmetry454 and Symmetry010 calendar years prior to the epoch

Like the Gregorian and ISO calendars, and in keeping with astronomical usage, the year before the year 1 (one) is defined as 0 (zero) and the years prior to that as –1, –2, –3, *etc*. In computer systems, years before year zero should be stored and processed as negative year numbers. **See CCUE for implementation details, specifically to ensure that any programmed remainder** *(mod* or *modulus)* **and adjusted remainder** (*amod*) **functions work properly with negative values.** The *modulus* and *amod* functions as shown above do work properly.

Fixed day numbers, astronomy and modern computer operating systems

The focus of CCUE is on conversion of any date on any calendar to or from a *rata die*, or fixed day number (ordinal day#). Gregorian January 1, 1 AD, the first day of the first millennium, is defined as fixed day number one, a Monday. All other dates are counted by days forward as positive numbers or backward as negative

numbers relative to that epoch. The day before the epoch was fixed day number zero. CCUE defines a *moment* as a fixed day number that has a fractional part representing the portion of the day elapsed since midnight.

Astronomers also use a fixed day numbering system, most commonly the *Julian Day* (JD) or the *Modified Julian Day* (MJD). The JD epoch was noon on Gregorian Monday, November 24, –4713, which corresponds to the CCUE moment -1,721,424.5. To convert a JD to a CCUE moment simply add the JD epoch moment. To convert a CCUE moment to a JD, simply subtract the JD epoch moment. The MJD epoch was 2,400,000.5 days after the JD epoch, which was midnight on Gregorian Wednesday, November 17, 1858, corresponding to the CCUE fixed day number 678,576. Some prefer the MJD because it relates directly to civil time and because the day numbers are smaller for dates near the present, allowing higher resolution of fractional moments within days. To convert an MJD moment to a CCUE moment, simply add the MJD epoch. To convert a CCUE moment to an MJD, simply subtract the MJD epoch.

The Windows operating systems use a similar internal day counter to represent dates, termed a date serial number. Officially, the Windows date serial number 1 was Gregorian January 1, 1900. All versions of Windows, however, make the mistake of treating the year 1900 as a leap year, so Windows reports date serial number 60 as the wrong date February 29, 1900 instead of March 1, 1900. Therefore to match Windows date serial numbers one must consider the Windows date serial number 1 to have been Gregorian December 31, 1899, which was CCUE fixed day number 693,595. Like CCUE, Windows represents a date/time moment as the Windows day number with a fractional part that is proportional to the time elapsed since midnight. To convert a CCUE moment to or from a Windows moment, just subtract or add 693,594, respectively. Negative or zero Windows date serial numbers are illegal, however, so there is no internal representation for dates prior to the Windows epoch, and because of the Windows leap year 1900 bug it is only possible to convert valid Gregorian dates to Windows date serial numbers for dates on or after March 1, 1900, which was Windows date serial number 61 and CCUE fixed day number 693,655.

The RainingData PICK / D3 / Advanced Revelation and derivative operating / database management systems represent dates as the number of days elapsed since Gregorian December 31, 1967. Thus Gregorian (or Symmetry454) January 1st, 1968 was PICK day number 1, corresponding to CCUE fixed day number 718,432. This system represents prior dates by zero or negative numbers.

Other epochs are easy to determine using the freeware *Kalendis* program. At any time, *Kalendis* version 9.241(963) or later allows the user to switch "on-the-fly" to employ any of several built-in fixed day numbering epochs, or the user can choose any arbitrary date as the fixed day numbering epoch.

To convert between the Symmetry454 calendar and **internal dates** used by any computer operating system such as discussed above you don't need any of the functions discussed in CCUE (unless your application needs to work beyond the Gregorian calendar date range that the operating system supports). The functions presented herein are sufficient, together with minor inter-epoch conversions, examples of which were given above.

Customizing the fixed day numbering epoch

On many computer systems **it is possible to use Symmetry454 calendar arithmetic directly with the existing internal date format, without converting any dates**, by simply setting an appropriate fixed day number for the calendar epoch, designated *SymEpoch*.

SymEpoch is always identical to *GregorianEpoch*, provided that the calendar year always starts on Monday, so if the fixed day numbering scheme is changed then *GregorianEpoch* must match, otherwise Gregorian arithmetic will malfunction. Even though they are identical, the calculations documented herein explicitly use *GregorianEpoch* where the context demands it.

A few examples are shown in the following table, where the value that appears in the *SymEpoch* column is the CCUE *rata die* of the epoch date (the *WeekdayAdjust* column is explained at the end of this section), sorted from the remote past to near the present era:

Fixed day numbering scheme	*SymEpoch*	*WeekdayAdjust*
Julian Day (JD) number	1721426	6
Day number of Hebrew calendar	1373429	0
CCUE *rata die*	1	0
Day number of Western Bahá'í calendar	-673220	4
Modified Julian Day (MJD) number	-678575	4
Windows OS date serial numbers	-693593	1
Pick OS date	-718430	0
Days since start of 3rd millennium	-730484	0

Any arbitrary date could be employed as the fixed day numbering epoch, simply by assigning the appropriate value to *SymEpoch*. *Kalendis* can be of assistance in determining the appropriate value to assign, and as of version 9.241(963) *Kalendis* allows the user to choose for its fixed day numbering scheme any of the fixed day numbering epochs tabulated above or any arbitrary date.

The *FixedToWeekdayNum*( ) function

The rightmost *WeekdayAdjust* column in the table in the previous section is a small integer value (0 to 6) that must be subtracted from any fixed day number before dividing it by 7 in order to correctly calculate the traditional weekday number (Sunday = 0, Monday=1, … Saturday = 6):

$$FixedToWeekdayNum(\ FixedDate\ ) = modulus(\ floor(\ FixedDate\ ) - WeekdayAdjust,\ 7\ )$$

Obviously, if the *WeekdayAdjust* equals zero and the application has no requirement to vary the fixed day numbering epoch, then it could be omitted from the *FixedToWeekdayNum*( ) function. If you prefer to number weekdays as Sunday=1, Monday=2, … Saturday=7 then simply add +1 to the result before returning it from the *FixedToWeekdayNum*( ) function.

The *WeekdayAdjust* value only needs to be **calculated once**, whenever the fixed day numbering epoch is set. To determine its value, use a reference a date that has a known weekday, conveniently the epoch of the Symmetry454 or Gregorian calendars, which was a Monday:

$$WeekdayAdjust = modulus(\ SymEpoch - 1,\ 7\ )$$

Don't store dates in any separated format

Storing or manipulating dates in any separated format, either as text or as discrete year, month and day numeric values, is considered very poor computer programming practice because:

- it limits the way that dates can be entered, displayed and printed
- dates may be ambiguous, for example when the year is stored as only two digits
- fixed day numbers or moments stored in binary format consume the least amount of memory or storage
- calendar arithmetic is cumbersome when using text or separated values (see below)
- storing fixed day numbers enables multiple users on the same system to simultaneously use their choice of calendar for date input or display, while internally all dates are stored as inter-convertible values.

Calendar arithmetic

Computing a person's age in days is accomplished by converting the birth date and later date on any calendars to their respective fixed day numbers, then simply subtracting fixed day number of birth from the fixed day

number of the later date. To express the age in years, divide the age in days by the number of days in the calendar mean year and then either truncate the fraction or round to the precision desired.

To compute when a term investment will mature or a when loan will terminate, convert the start date to its fixed day number, add the duration of the agreement in days, then convert that result back to a calendar date.

Fixed day numbers are also useful when plotting charts of any measurement *vs*. date, to ensure that the date axis is a linear time scale (directly proportional to the number of elapsed days or longer time units).

The difference in days and fraction of a day between any two moments is computed by simple subtraction, although this does not take into account any leap seconds that may have been inserted between those moments.

Although such calculations are tedious by hand, computers execute them within less than a microsecond.

Overview of Symmetry454 and Symmetry010 calendar arithmetic

Symmetry454 and Symmetry010 calendar arithmetic functions boil down to three main calculations:

1. Find where the calendar year starts: the Symmetry454 = Symmetry010 New Year Day.
2. Optionally determine if the year is a Symmetry454 = Symmetry010 leap year (usually not necessary).
3. Compute the parameters that are set by the calendar structure, which are same for all Symmetry454 and Symmetry010 years: ordinal day number, ordinal week number, quarters, months, weeks, days, weekdays. Some of these parameters are optional, depending on the application requirements.

Overview of conversions to / from the Symmetry454 or Symmetry010 calendars

To interconvert dates between the Symmetry454 or Symmetry010 calendar and any other calendar one needs only a few of the functions published in CCUE (or their equivalents) plus a *FixedToSym*( ) function and a *SymToFixed*( ) function, both presented below. For example, to convert a Symmetry454 or Symmetry010 date (*SymYear*, *SymMonth* , *SymDay*) to a Gregorian date, use my *SymToFixed*( ) function to convert the Symmetry454 or Symmetry010 date to the fixed day number, then use the CCUE *gregorian-from-fixed*( ) function to arrive at the Gregorian date (*year*, *month*, *day*). To go in the reverse direction, use the CCUE *fixed-from-gregorian*( ) function to convert the Gregorian date to a fixed day number, and then use my *FixedToSym*( ) function to convert the fixed day number to the corresponding Symmetry454 or Symmetry010 date.

We define *SymMonth* as a simple positive integer enumeration: *January* = 1, *February* = 2, *March* = 3, *April* = 4, *May* = 5, *June* = 6, *July* = 7, *August* = 8, *September* = 9, *October* = 10, *November* = 11, *December* = 12. Normally the leap week is appended to December, but it can optionally stand alone after December as a 13th mini-month, in which can we can define *Irvember* = 13.

Note that my function naming style is the converse of CCUE. I use the function naming convention *ThatToThis* instead of *this-from-that*, to keep the function name shorter and programmatically more convenient.

To convert a Gregorian date to a fixed date, which can then be converted to a Symmetry454 or Symmetry010 date, it is convenient to calculate how many days have elapsed from the Gregorian epoch to the day before the Gregorian year started, then add in the ordinal day number within the Gregorian year.

The *PriorElapsedDays*( ) function returns the number of calendar days that have elapsed from the Gregorian epoch until the beginning of the New Year Day of the specified Gregorian year number:

$$PriorElapsedDays(\ GregYear\ ) \ = \ GregorianEpoch \ + \ priorYear \ \times 365 \ + \left\lfloor \frac{priorYear}{4} \right\rfloor - \left\lfloor \frac{priorYear}{100} \right\rfloor + \left\lfloor \frac{priorYear}{400} \right\rfloor - 1$$

where *priorYear* = *GregYear* – 1 and *GregorianEpoch* was defined under the heading "**Customizing the fixed day numbering epoch**", above.

The *PriorElapsedDays*( ) function allows 365 days for each prior elapsed year, plus one day for each prior 4-year sub-cycle to account for leap years, minus one day for each prior centurial year (because most were not leap years), and finally plus one day for every 400 years (full Gregorian cycle) to account for the prior centurial years that were leap years. For example:

$$\textit{PriorElapsedDays}(2009) = \textit{GregorianEpoch} + 2008 \times 365 + \left\lfloor \frac{2008}{4} \right\rfloor - \left\lfloor \frac{2008}{100} \right\rfloor + \left\lfloor \frac{2008}{400} \right\rfloor - 1$$

$$= \textit{GregorianEpoch} + 732920 + \lfloor 502 \rfloor - \lfloor 20.08 \rfloor + \lfloor 5.02 \rfloor - 1$$

$$= \textit{GregorianEpoch} + 732920 + 502 - 20 + 5 - 1$$

$$= \textit{GregorianEpoch} + 733406$$

If *GregorianEpoch* = 1 then the result is 733407, indicating that 733407 days elapsed from the epoch until the start of Gregorian New Year Day 2009.

One could optionally implement a simple *GregorianNewYearDay*(*GregYear*) function that will return the fixed day number of the New Year Day of the specified Gregorian year, simply by omitting the final "– 1" from the expression shown above for the *PriorElapsedDays*( ) function.

Next compute the ordinal day number within the Gregorian year (see reference 1), and then apply a one- or two-day deduction if the *month* is after February:

$$\textit{GregorianOrdinalDay} = \left\lfloor \frac{367 \times \textit{month} - 362}{12} \right\rfloor + \textit{day}$$

```
IF month > 2 THEN
    IF isGregorianLeapYear( GregYear ) THEN
        GregorianOrdinalDay = GregorianOrdinalDay – 1
    ELSE
        GregorianOrdinalDay = GregorianOrdinalDay – 2
    END IF
END IF
```

For example, the ordinal day number of July 14th in a non-leap Gregorian year is:

$$\left\lfloor \frac{367 \times 7 - 362}{12} \right\rfloor + 14 = \left\lfloor \frac{2207}{12} \right\rfloor + 14 = \left\lfloor 183 + \frac{11}{12} \right\rfloor + 14 = 183 + 14 = 197$$

minus 2 days because the date is after February in a non-leap year = the 195th day of the year.

A simpler, albeit abstruse alternative, which by "pretending" that the Gregorian New Year Day is March 1st avoids adjusting for the *month* being after February and also avoids checking if it is a Gregorian leap year, is openly available on the internet, see the erratum for page 56 of *Calendrical Calculations: The Millennium Edition*, showing a simplified *alt-fixed-from-gregorian* function.

Symmetrical leap cycles and the Symmetry454 and Symmetry010 leap rule

The single-step Symmetry454 and Symmetry010 calendar leap rule inherently and automatically distributes leap years at intervals that are symmetrically arranged and as smoothly spread as possible.

Symmetrically arranging the leap years means that the leap status (non-leap or leap) of year *n* in each leap cycle is the same as the leap status of the symmetrical year occurring *n* years prior to the beginning of the next cycle. This arrangement yields the advantage that mean equinox or solstice timing always falls at the cycle average in the first year of each cycle (provided that the leap week is appended to the end of the calendar year or is inserted somewhere after the target equinox or solstice). This feature simplifies astronomical performance evaluations: to carry out long-term astronomical drift analysis of a symmetrical leap cycle it is only necessary to assess the first year of each cycle, then smoothly interpolate from cycle-to-cycle.

Spreading the cycle leap years as smoothly as possible minimizes the short-term calendar date equinox or solstice "jitter" or "wobble" range.

The following very simple *isSymLeapYear*( ) function returns TRUE if the specified *SymYear* is a leap year, or FALSE if *SymYear* is a non-leap year:

$$isSymLeapYear(\ SymYear\ ) = modulus(\ L \times SymYear + K,\ C\ ) < L$$

where C is the number of years per cycle = 293 (a prime number), L is the number of leap years per cycle = 52, and $K = (C\text{-}1) / 2 = 146$. The K coefficient ensures that leap years are symmetrically arranged, and the *modulus* operation ensures that leap years are as smoothly spread as possible.

The quantity *modulus*($L \times SymYear + K$, C) is also known as the *accumulator*.
So we can say that the year is a leap year if its *accumulator* is less than L.
Leap year intervals are either 6 or 5 years, with 6-year intervals being about twice as frequent.
The *accumulator* is also useful for predicting whether the interval to the next leap year will be 6 or 5 years.
The following expression yields the number of longer (6-year) inter-leap intervals per cycle:

$$R = C - L \times S$$

where S is the shorter leap interval length (including one leap year) = 5 years in the case of any leap week calendar, and R is the number of longer (S + 1 years) inter-leap intervals per cycle. If for a given year its *accumulator* is < R then the next leap year will be S + 1 years later, otherwise the next leap year will be S years later. This is useful for logical efficiency when generating a list of leap years. The number of shorter (S years) inter-leap intervals per cycle = $L - R$.

Substituting constants for variables:

$$isSymLeapYear(\ SymYear\ ) = modulus(\ 52 \times SymYear + 146,\ 293\ ) < 52$$

$R = 293 - 52 \times 5 = 33$, so there are $52 - 33 = 19$ short inter-leap intervals, $33{:}19 \approx 2{:}1$ as expected

For example:

$$isSymLeapYear(\ 2009\ ) = modulus(\ 52 \times 2009 + 146,\ 293\ ) < 52$$

$$isSymLeapYear(\ 2009\ ) = modulus(\ 104614,\ 293\ ) < 52$$

104614 / 293 = 357+$^{13}/_{293}$ so the *accumulator* is 13, which is < 52 and the function result is TRUE, so the year 2009 was a leap year in the 52/293 leap cycle. Because 13 is also < 33, the number of long inter-leap intervals per 293-year cycle, the next leap year was 6 years later = 2015.

For the symmetrical 293-year cycle, leap year intervals match sub-cycle patterns of (5+6+6) = 17 or (5+6) = 11 years, which symmetrically group to 17+11+17 = 45 or 17+17+11+17+17 = 79 years. The overall symmetrical grouping for this cycle is 45+79+45+79+45 = 293 years. The total number of days per full 293-year cycle is exactly equal to 294 non-leap years, so the *average* interval between leap weeks is exactly 294 weeks.

The middle year of each 293-year cycle is a non-leap year because the cycle has an even number of leap years.

The 293-year leap cycle is ideal for approximating the mean **northward equinox** (March equinox, boreal vernal equinox, northern hemisphere spring equinox) for the past 5 millennia and the next **4 to 5 millennia**, keeping the average equinox near midnight in Jerusalem at the beginning of the 80th ordinal day of the calendar year (Symmetry454 March 17th or Symmetry010 March 19th).

Alternatively, a symmetrical smoothly spread 389-year leap cycle (389 is a full reptend prime number) with C = 389, L = 69, and K = 194 is ideal for approximating the mean **north solstice** (June solstice, boreal summer solstice, northern hemisphere summer solstice) for the past millennium and the next **10 to 11 millennia**, keeping the average solstice near midnight in Jerusalem at the beginning of the 174th ordinal day of the calendar year (Symmetry454 June 20th or Symmetry010 June 22nd):

$$isSymLeapYear(\ SymYear\ ) = modulus(\ 69 \times SymYear + 194,\ 389\ ) < 69$$

$R = 389 - 69 \times 5 = 44$, so there are $69 - 44 = 25$ short inter-leap intervals, $44{:}25 \approx 2{:}1$ as expected

For example:

$$isSymLeapYear(\ 2009\ ) = modulus(\ 69 \times 2009 + 194,\ 389\ ) < 69$$

$$= modulus(\ 138815,\ 389\ ) < 69$$

138815 / 389 = 356+$^{331}/_{389}$ so the *accumulator* is 331, which is **not** less than 69 and the function result is FALSE, meaning that the year 2009 wasn't a leap year in the 69/389 leap cycle. Nevertheless, the large *accumulator* 331 is only 58 away from 389, suggesting that a leap year was next, let's check that:

$$isSymLeapYear(\ 2010\ ) = modulus(\ 69 \times 2010 + 194,\ 389\ ) < 69$$

$$= modulus(\ 138884,\ 389\ ) < 69$$

138884 / 389 = 357+$^{11}/_{389}$ so the *accumulator* is 11, which is less than 69 and the function result is TRUE, meaning that the year 2010 was indeed a leap year in the 69/389 leap cycle. Since 11 is also less than 44, the number of long inter-leap intervals per 389-year cycle, the next leap year will be 6 years later = 2016.

For the symmetrical 389-year cycle, leap year intervals match sub-cycle patterns of (5+6+6) = 17 or (5+6) = 11 years, which symmetrically group to 17+11+17 = 45 or 17+17+11+17+17 = 79 or to the longer 17+17+11+17+17+17+11+17+17 = 141 years. The overall symmetrical grouping for this cycle is 79+45+141+45+79 = 389 years.

The middle year of each 389-year cycle is a leap year because the cycle has an odd number of leap years.

With any reasonably accurate leap week calendar cycle, 6-year leap intervals occur about twice as frequently as 5-year leap intervals. The reason for this is explained on my "Solar Calendar Leap Rules" web page at

<http://individual.utoronto.ca/kalendis/leap/> under the heading "Patterns in the Intervals Between Leap Years".

A quick way to evaluate astronomical drift using *Kalendis* version 9.709(1429) or later is as follows:

1. Select the built-in locale "Jerusalem, Israel".

2. Set the fixed day numbering epoch to the Gregorian epoch using Options → Fixed Day Numbering… → Gregorian epoch (*rata die*).

3. Choose the desired symmetrical leap cycle in the Symmetry calendar window. If you want other than the 293-year cycle then mark the "Experiment" checkbox to enable choosing alternatives, such as the 389-year cycle.

4. Enter 1 as the Fixed Day Number. This changes the displayed date to Monday, January 1, 1 (AD), which was the beginning of the first leap cycle at the calendar epoch (assuming "Start On" = Monday).

5. Use the "Next" menu to advance to your choice of northward equinox or north solstice within the year. Note the Sun Longitude displayed nearby, which should be 0.0° or 90.0°, respectively.

6. Use the "Previous" or "Next" menu to jump to the same point in the first year of the previous or next leap cycle for the selected symmetrical leap rule.

    a. If using the 293-year leap cycle with the Sym454 calendar then the applicable "Previous" and "Next" menu command will be "52/293 Sym454 Leap Cycle", but the caption for this command will change if you select a different leap cycle or calendar, such as Sym010 or an experimental variant.

    b. In the first year of each 52/293 leap cycle, the northward equinox lands on March 16th for Sym454 or March 18th for Sym010, near the end of the 79th ordinal day number for either calendar.

    c. In the first year of each 69/389 leap cycle the north solstice lands on June 19th for Sym454 or June 21st for Sym010, near the end of the 173rd ordinal day number for either calendar.

    d. The time of the equinox or solstice is shown as Universal Time in the "Univ Time" field. If you click on this field, *Kalendis* will copy this time to the clipboard as Universal Time, Standard Time for the northward equinox or Daylight Saving Time for the north solstice, Local Mean Time, and Local Apparent Time. Note that the moment for the first year of each leap cycle should be near midnight Jerusalem Local Mean Time.

    e. If the moment is after midnight in Jerusalem it will still be before midnight at the Prime Meridian, and *Kalendis* uses Universal Time for its calendar dates. Thus in terms of Jerusalem Local Mean Time, the northward equinox of the first year of each 293-year leap cycle lands near the beginning of the 80th ordinal day number (Sym454 March 17th, Sym010 March 19th), and the north solstice of the first year of each 389-year leap cycle lands near the beginning of the 174th ordinal day number (Sym454 June 20th, Sym010 June 22nd).

7. Watch the Sun Longitude: for each degree difference from the original value at the epoch the calendar has drifted about a day.

8. You can quickly continue jumping backwards or forwards by whole leap cycles simply by pressing Ctrl-A to activate the Step → Again command. In a matter of a few seconds you can find the first year of the cycle where the calendar drift deviates excessively.

Using *Kalendis* as outlined above with the 293-year leap cycle and starting from the northward equinox (0°) in year 1 AD then jumping forward by 293-year cycles, a deviation of more than a degree from 0° isn't reached until the cycle that will begin in 8791 AD, when it will reach 1.1°.

Carrying out the same procedure with the 389-year leap cycle and starting from the north solstice (90°) in year 1 AD then jumping forward by 389-year cycles, a deviation of more than a degree from 90° is never reached within the allowable year range supported by *Kalendis*, which stops at 12000 AD, in fact the solar longitude is still displayed as 90.0° in the year 11671 AD, the last cycle that *Kalendis* can reach the beginning of.

Some additional definitions and properties of smoothly spread symmetrical leap cycles include:

1) There are an odd number of years per cycle.
2) The middle year of each cycle is a leap year only if there are an odd number of leap years per cycle.
3) $NewYearMoment = epoch + 365 \times (Y - 1) + F \times (Y - 1)$, where *epoch* is the start of calendar year 1, Y is the given year number, and F is the fraction of the calendar mean year that is in excess of 365 days. Unless the calculation is carried out using exact arbitrary precision, don't simplify this expression to: $NewYearMoment = epoch + (Y - 1) \times (365 + F)$.
4) It is simpler to use the elapsed year count $E = Y - 1$, in which case $NewYearMoment = epoch + 365 \times E + F \times E$. As above, unless the calculation is carried out using exact arbitrary precision, don't simplify this expression to: $NewYearMoment = epoch + E \times (365 + F)$.
5) Although the mean *NewYearMoment* expression would be the same for a non-symmetrical leap cycle, only in the case of a symmetrical leap cycle does it exactly equal the start of calendar year 1 and the start of the first year of every cycle.
6) Any given year Y begins on day $epoch + D \times (Y - 1) + X \times floor([L \times (Y - 1) + K] / C)$, where D = the number of days in a non-leap year, X = the number of days in a leap unit (=7 for leap week calendars), and a leap year has $D + X$ days. Starting from the calendar *epoch*, this expression adds the non-leap calendar year length for each elapsed year and plus the length of the leap unit for each elapsed leap year.
7) It is simpler to use the elapsed year count $E = Y - 1$, in which case year Y begins on day $epoch + D \times E + X \times floor([L \times E + K] / C)$.
8) The maximum within-cycle "wobble" W of the mean equinox or solstice or new year moment is $\pm X \times (C - 1) / (2 \times C)$ days, where X = the number of days in the calendar leap unit (=7 for leap week calendars), provided that the leap year intervals are as smoothly spread as possible, so the drift of the earliest and latest mean equinox or solstice can simply be plotted as $\pm W$ days from the drift of the average, where the average is represented by the first year of each leap cycle, interpolating from cycle-to-cycle.

For more information about smoothly spread symmetrical leap cycles (having an odd number of years per cycle), as well as *almost* symmetrical cycles (having an even number of years per cycle), please see the topic "Smoothly Spread Symmetrical Leap Cycles" on the web page entitled "Solar Calendar Leap Rules" at <http://individual.utoronto.ca/kalendis/leap/>.

The symmetrical leap cycle concept, rules, and arithmetic are largely due to K.E.V. (Karl) Palmen, formerly of the Rutherford Appleton Laboratory in the United Kingdom (retired in 2018), primarily based on correspondence with the CALNDR LISTSERV during 2007-2008, in threads concerned with what he called

'Helios' and 'quasi-Helios' cycles. Karl originally suggested the use of symmetrical leap rules for the Symmetry454 calendar, however, back in 2004, in direct email correspondence with this author.

Generating a list of Symmetry454 leap years

It is a simple matter to iteratively check a range of years and list those for which the *isSymLeapYear*( ) function returns TRUE. There is no need to check each individual year, however, because each time that a leap year is found the loop can inspect the *Accumulator* and if it is < *R* then jump 6 years otherwise jump 5 years directly to the next leap year. The *R* threshold need be calculated only once before starting the loop:

$$R = C - L \times S$$

where $S = 5$ is the number of years in a short inter-leap interval (including one leap year), R is the number of longer ($S + 1$ years) inter-leap intervals per cycle, C is the number of years per cycle, and L is the number of leap years per cycle.

Kalendis has a built-in "Report" menu that has a command for exporting a list of Symmetry454 leap years for the currently selected leap rule. The user can choose to export this list as a web page (.htm), tab-delimited text (.tab), or comma-separated values (.csv). After exporting, *Kalendis* opens the list using the user's default application for that file type. Typically the **`.htm`** file type will open in the user's default web browser. Most users will find it convenient to configure Windows to open the **`.tab`** or **`.csv`** files in a spreadsheet or statistics application. To change this setting, right-click on the exported file using "My Computer" or "Windows Explorer" and choose the "Open With…" command, choose the program to use for opening that file type and mark the "Always use this program to open these files" checkbox, then click the "OK" button.

Finding the Symmetry454 New Year Day: the *SymNewYearDay*( ) function

This simple function returns the fixed day number of the New Year Day of any specified Symmetry454 year:

$$SymNewYearDay(\ SymYear\ ) = SymEpoch + 364 \times E + 7 \times \left\lfloor \frac{L \times E + K}{C} \right\rfloor$$

where $E = SymYear - 1$, C = years per cycle, L = leap years per cycle, $K = (C\text{-}1)/2$ and *SymEpoch* is the fixed day number of January 1 of the year 1 AD, as previously explained under the heading "**Customizing the fixed day numbering epoch**".

Thus for the **293-year northward equinox cycle** we have:

$$SymNewYearDay(\ SymYear\ ) = SymEpoch + 364 \times E + 7 \times \left\lfloor \frac{52 \times E + 146}{293} \right\rfloor$$

If *SymEpoch* was on Monday, then every New Year Day is also on Monday.

If *SymEpoch* = *GregorianEpoch* = 1 then the 52/293 cycle starts year 2010 on:

$$SymNewYearDay(\ 2010\ ) = 1 + 364 \times 2009 + 7 \times \left\lfloor \frac{52 \times 2009 + 146}{293} \right\rfloor$$

$$= 731277 + 7 \times \left\lfloor \frac{104614}{293} \right\rfloor$$

$$= 731277 + 7 \times \left\lfloor 357 + \frac{13}{293} \right\rfloor$$

$$= 731277 + 7 \times 357$$

$$= \text{fixed day number } 733776$$

and for the **389-year north solstice cycle** we have:

$$SymNewYearDay(\ SymYear\ ) = SymEpoch + 364 \times E + 7 \times \left\lfloor \frac{69 \times E + 194}{389} \right\rfloor$$

So the 69/389 cycle starts year 2010 on:

$$SymNewYearDay(\ 2010\ ) = 1 + 364 \times 2009 + 7 \times \left\lfloor \frac{69 \times 2009 + 194}{389} \right\rfloor$$

$$= 731277 + 7 \times \left\lfloor \frac{138815}{389} \right\rfloor$$

$$= 731277 + 7 \times \left\lfloor 356 + \frac{331}{389} \right\rfloor$$

$$= 731277 + 7 \times 356$$

$$= \text{fixed day number } 733769$$

The 69/389 New Year Day is 7 days prior to the 52/293 New Year Day in year 2010 because the 69/389 cycle makes 2009 a non-leap year but the 52/293 cycle appends a leap week to year 2009.

If one prefers to work with smaller fixed day numbers then choose a fixed day numbering epoch near the target era. For example, I like to use the first day of the 3rd millennium = January 1, 2001 AD as my fixed day numbering epoch, so that near present era dates will have small fixed day numbers. This date was a Monday, is the same date on the Gregorian and Symmetry454 of Symmetry010 calendars (using either the 293- or 389-year cycle), and was the first day of a Gregorian 400-year cycle. If that date is the fixed day numbering epoch then *SymEpoch* = -730484 as was shown in the **Fixed Day Number Scheme** table above, indicating that the Gregorian and Symmetry epoch was 730484 days prior to January 1, 2001. Using *SymEpoch* = -730484 the 52/293 New Year Day 2010 becomes:

$$SymNewYearDay(\ 2010\ ) = -730484 + 364 \times 2009 + 7 \times \left\lfloor \frac{52 \times 2009 + 146}{293} \right\rfloor$$

$$= 792 + 7 \times 357 = 3291$$

and as above the 69/389 cycle starts the year one week earlier:

$$SymNewYearDay(\ 2010\ ) = -730484 + 364 \times 2009 + 7 \times \left\lfloor \frac{69 \times 2009 + 194}{389} \right\rfloor$$

$$= 792 + 7 \times 356 = 3284$$

For more information about symmetrical leap cycles please see the heading "Smoothly Spread Symmetrical Leap Cycles" on my "Solar Calendar Leap Rules" web page at <http://individual.utoronto.ca/kalendis/leap/>. That web page also more fully documents the 293- and 389-year leap cycles as well as a collection of other interesting leap cycles, offers tools for evaluating the astronomical drift of various leap cycles (see the heading "Dynamic Demonstration of Mean Solar Calendar Drift Rates") and for finding fixed arithmetic leap cycles that meet specified criteria (see the heading "Automatic Fixed Leap Cycle Finder" and the heading "Mediant Fractions, Farey Pairs, and Ford Circles".

Calculating the calendar mean year

A calendar mean year is the full-cycle average length of the calendar year.

The calendar mean year for any fixed arithmetic leap cycle = *DaysPerCycle* / *YearsPerCycle*.

For a leap week calendar, *DaysPerCycle* = (*YearsPerCycle* × 364) + (*LeapWeeksPerCycle* × 7).

The mean year of the 293-year northward equinox leap cycle = (293 × 364 + 52 × 7) / 293 ≡ 365+$^{71}/_{293}$ days ≡ 365 days 5 hours 48 minutes 56+$^{152}/_{293}$ seconds or about 365.242321 days, which is an excellent average match to the astronomical mean northward equinoctial year for the next 4-5 millennia. The fraction $^{71}/_{293}$ indicates that such a leap week cycle is the mean year equivalent of a leap **day** cycle having 71 leap years per 293-year cycle.

The mean year of the 389-year north solstice leap cycle = (389 × 364 + 69 × 7) / 389 ≡ 365+$^{94}/_{389}$ days ≡ 365 days 5 hours 47 minutes 58+$^{58}/_{389}$ seconds or about 365.241645 days, which is an excellent match to the astronomical mean north solstitial year now and for the next 10-11 millennia. The fraction $^{94}/_{389}$ indicates that such a leap week cycle is the mean year equivalent of a leap **day** cycle having 94 leap years per 389-year cycle.

Compare the above with the appreciably longer mean year of the Gregorian calendar ≡ 365+$^{97}/_{400}$ days ≡ 365 days 5 hours 49 minutes 12 seconds ≡ 365.2425 days.

Alternatively, for a leap day calendar the mean year = 365 + *LeapDaysPerCycle* / *YearsPerCycle*

and for a leap week calendar the mean year = 364 + 7 × *LeapWeeksPerCycle* / *YearsPerCycle*

When working with floating point decimal numbers, it is more precise to calculate just the *fraction* of the mean year that is in excess of 365 days, because in the above expressions 3 significant figures are unavoidably used for the 365 days to the left of the decimal point:

for a leap day calendar the mean year fraction = *LeapDaysPerCycle* / *YearsPerCycle*

and for a leap week calendar the mean year fraction = 7 × *LeapWeeksPerCycle* / *YearsPerCycle* – 1

For example, for the 52/293 cycle the expression 7 × *LeapWeeksPerCycle* / *YearsPerCycle* yields 1.24232081911263 as a double precision floating point number, which is the mean number of days in excess of a 364-day common year, and the final subtraction of one day yields the mean fraction of a day in excess of a 365-day year. This value is only approximate, due to the limitations of double precision floating point calculation, because the exact decimal fraction of $^{71}/_{293}$ has 146 repeating digits.

For more information about the lengths of the astronomical seasons and mean equinoctial and solstitial years, please see my "The Lengths of the Seasons" web page at <http://individual.utoronto.ca/kalendis/seasons.htm>, which explains why it is presently impossible to use a simple fixed arithmetic leap cycle as an approximation for the astronomical mean southward equinox (September equinox, boreal autumnal equinox, northern hemisphere autumn equinox) or the south solstice (December solstice, boreal winter solstice, northern hemisphere winter solstice).

The *DaysBeforeMonth*( ) function

The *DaysBeforeMonth*( ) function calculates the number of days elapsed in the same year prior to a specified month. It returns zero for January. If the leap week stands alone as a 13th "mini-month" after December then it

returns 364 for "Irvember". This function is one of the few that differ for the Symmetry454 *vs* Symmetry010 calendar arithmetic, as indicated:

4+**5**+4 $$DaysBeforeMonth(\ SymMonth\ ) = 28\times(SymMonth-1)+7\times\left\lfloor\frac{SymMonth}{3}\right\rfloor$$

4+**5**+4 $$\text{(or, slightly simpler …)} = 28\times SymMonth+7\times\left\lfloor\frac{SymMonth}{3}\right\rfloor-28$$

The above expression allows 28 days for each prior elapsed month, plus 7 days for each prior elapsed mid-quarter month. For example, for June (month 6):

4+**5**+4 $$DaysBeforeMonth(\ 6\ ) = 28\times 6+7\times\left\lfloor\frac{6}{3}\right\rfloor-28 = 168+7\times\lfloor 2\rfloor-28 = 168+14-28 = 154 \text{ days.}$$

The Symmetry010 expression below differs by allowing 30 days for each prior elapsed month, plus one day for each prior elapsed mid-quarter month:

30+**31**+30 $$DaysBeforeMonth(\ SymMonth\ ) = 30\times(SymMonth-1)+\left\lfloor\frac{SymMonth}{3}\right\rfloor$$

30+**31**+30 $$\text{(or, slightly simpler…)} = 30\times SymMonth+\left\lfloor\frac{SymMonth}{3}\right\rfloor-30$$

For example, for June:

30+**31**+30 $$DaysBeforeMonth(\ 6\ ) = 30\times 6+\left\lfloor\frac{6}{3}\right\rfloor-30 = 180+\lfloor 2\rfloor-30 = 182-30 = 152 \text{ days.}$$

The *SymDayOfYear*( ) function

To calculate the ordinal day number of a given day *SymDay* within a given month *SymMonth*, simply add *SymDay* to the value returned by the *DaysBeforeMonth*( ) function, defined above:

$$SymDayOfYear(\ SymMonth, SymDay\ ) = DaysBeforeMonth(\ SymMonth\ ) + SymDay$$

There are no special exceptions, because the calendar is perpetual with its leap week at the end of leap years.

For example, for June 17th:

$$SymDayOfYear(\ 6, 17\ ) = DaysBeforeMonth(\ 6\ ) + 17$$

4+**5**+4 $SymDayOfYear(\ 6, 17\ ) = 154 + 17 = 171$ (we calculated the 154 in the previous section)

30+**31**+30 $SymDayOfYear(\ 6, 17\ ) = 152 + 17 = 169$ (we calculated the 152 in the previous section)

The *SymDayOfYear*( ) arithmetic is the same for both Symmetry454 and Symmetry010. The 2-day difference arises within the *DaysBeforeMonth*( ) function, as explained in the previous section.

Converting a Sym454 or Sym010 date to a fixed day number: *SymToFixed*( )

The *SymToFixed*( ) function converts any Sym454 or Sym010 calendar date to the corresponding fixed day number, by adding the ordinal day number within the year to the fixed day number of the New Year Day, less 1:

SymToFixed(*SymYear*, *SymMonth*, *SymDay*) =
SymNewYearDay(*SymYear*) + *SymDayOfYear*(*SymMonth*, *SymDay*) – 1

For example, for April 5, 2009:

SymToFixed(2009, 4, 5) = *SymNewYearDay*(2009) + *SymDayOfYear*(4, 5) – 1

With *SymEpoch* = 1 the *SymNewYearDay*(2009) function returns 733405.

SymDayOfYear(4, 5) returns 96, so we have 733405 + 96 – 1 = 733500.

In this example, the same result is obtained using the 52/293 or 69/389 cycle, and either Sym454 or Sym010.

Finding the year that contains a given fixed day number: *FixedToSymYear*( )

The *FixedToSymYear*( ) function is the inverse of the *SymNewYearDay*( ) function. It returns the *SymYear* that contains a given fixed day number and also returns *StartOfYear*, which is the fixed day number of its New Year Day, both of which are required early in the *FixedToSym*( ) function to be discussed next. The calculation first estimates the *SymYear* number, which will always be within ±1 year of the target *SymYear*, and then checks if it is correct, incrementing or decrementing it if necessary.

To estimate the required year number, simply calculate the difference between the fixed day number and the calendar epoch, divide that by the cycle mean year, and then take the *ceiling*( ) of that result:

$$SymYear = \left| \frac{FixedDate - SymEpoch}{CycleMeanYear} \right|$$

This expression needs the *ceiling*( ) function because the year at the calendar epoch was year 1, not year 0.

The 52/293 northward equinox cycle uses $CycleMeanYear \equiv 365+{}^{71}/_{293} \equiv {}^{107016}/_{293} \approx 365.24232082$ days.

The 69/389 north solstice cycle uses $CycleMeanYear \equiv 365+{}^{94}/_{389} \equiv {}^{142079}/_{389} \approx 365.241645$ days.

The estimated year number is generally correct for dates that are not near the beginning or end of the year. We check the accuracy of the estimated *SymYear*, incrementing or decrementing it if necessary, by comparing the fixed day number of its New Year Day with the given *FixedDate*. We get *StartOfYear* for the estimated *SymYear* using the *SymNewYearDay*( ) function:

StartOfYear = *SymNewYearDay*(*SymYear*)

If *StartOfYear* equals the given *FixedDate* then the given date is the New Year Day of the *SymYear* and in that uncommon case no further adjustment is needed. We don't, however, bother checking for that condition, because if it is TRUE then processing will simply skip through the following IF statements:

```
IF StartOfYear < FixedDate THEN
    ' SymYear starts before FixedDate and is either correct or needs to be incremented
    IF FixedDate – StartOfYear >= 364 THEN
        StartOfNextYear = SymNewYearDay( SymYear + 1 )
```

```
            IF FixedDate >= StartOfNextYear THEN
                ' FixedDate is on or after the start of next year, so next year is the correct year.
                ' Increment the estimated year number and return its New Year Day
                SymYear = SymYear + 1
                StartOfYear = StartOfNextYear
            END IF ' otherwise FixedDate is in the leap week of SymYear
        END IF  ' otherwise FixedDate is within SymYear
    ELSEIF StartOfYear > FixedDate THEN
        ' estimated SymYear too far into the future, go back a year and recalculate the New Year Day
        SymYear = SymYear – 1
        StartOfYear = SymNewYearDay( SymYear )
    END IF ' otherwise StartOfYear = FixedDate so there is nothing else to do
```

The initially estimated *SymYear* is always within ±1 year of the correct year, so there is no need for logic to check outside that range.

FixedToSymYear( ) processing is now complete. *SymYear* contains the correct year number, and *StartOfYear* contains the fixed day number of its New Year Day.

For a given *FixedDate* of 733649 with *SymEpoch* = 1 and the 52/293 leap cycle, we estimate the *SymYear*.

$$SymYear = \left\lceil \frac{733649-1}{365+71/293} \right\rceil$$

$$SymYear = \left\lceil 2008+\frac{8842}{13377} \right\rceil = 2009$$

Although this shows exact fractional arithmetic, decimal fractions work just as well here, and the *CycleMeanYear* can be pre-calculated and declared as a double-precision floating-point constant.

StartOfYear = *SymNewYearDay*(2009) = 733405 as was shown in *SymToFixed*( ) above

The given *FixedDate* is 733649–733405=244 days after the New Year Day of the estimated year, so the 2009 estimate is correct, as it typically is for dates that are not near the beginning or end of the year, and we have the fixed day number of its New Year Day.

Now let's try a *FixedDate* of 733406:

$$SymYear = \left\lceil \frac{733406-1}{365+71/293} \right\rceil$$

$$SymYear = \left\lceil 2007+\frac{106553}{107016} \right\rceil = 2008$$

StartOfYear = *SymNewYearDay*(2008) = 733041 (364 days earlier than the previous example)

The given *FixedDate* is 733406–733041=365 days after the New Year Day of the estimated year, so the year 2008 estimate is correct only if 2008 was a leap year (it wasn't). We get the *StartOfNextYear*:

StartOfNextYear = *SymNewYearDay*(2008 + 1) = *SymNewYearDay*(2009) = 733405 (like previous example)

The formal logic says that if *FixedDate* is on or after *StartOfNextYear* then next year is correct, but that was obvious anyway because we can immediately see that *FixedDate* is simply the day after the New Year Day of year 2009. This is a typical example of a case where *FixedDate* is near the beginning of a calendar year, so the initial year estimate lands in the prior year.

Now let's try a *FixedDate* of 733774:

$$SymYear = \left\lceil \frac{733774 - 1}{365 + 71/293} \right\rceil$$

$$SymYear = \left\lceil 2009 + \frac{115}{35672} \right\rceil = 2010$$

$$StartOfYear = SymNewYearDay(2010) = 733776$$

The given *FixedDate* is 733774 is 2 days earlier, so we need the prior year and its New Year Day. This is a typical example of a case where *FixedDate* is near the end of a calendar year, so the initial year estimate lands in the next year. In this case the given *FixedDate* was in the leap week of year 2009.

Converting a fixed day number to a Sym454 or Sym010 date: *FixedToSym*( )

The *FixedToSym*( ) function converts any fixed day number to the corresponding Symmetry454 calendar date (year, day of year, week of year, month of year, quarter of year, day of month, weekday).

This function requires some way of returning multiple values, either by changing global variables, returning a structure, or storing its results as properties in a calendar object.

It first uses the *FixedToSymYear*( ) function, described above, to obtain the *SymYear* and *StartOfYear*. In the *Kalendis* implementation, *SymYear* is returned as the function result, and *StartOfYear* is passed as a *by reference* parameter to the *FixedToSymYear*( ) function, which modifies it directly:

$$SymYear = FixedToSymYear(FixedDate, StartOfYear)$$

The Symmetry454 ordinal day number within the year (1 to 364 for non-leap, or to 371 for leap years) is simply the given *FixedDate* minus the *StartOfYear*, plus one day:

$$DayOfYear = FixedDate - StartOfYear + 1$$

The following expression computes the Symmetry454 week number (1 to 52 for non-leap, to 53 for leap years):

$$WeekOfYear = \left\lceil \frac{DayOfYear}{7} \right\rceil$$

The following expression computes the Symmetry454 quarter number within the calendar year. The fraction causes the expression to yield the correct *Quarter* number for all weeks including the 53rd week at the end of leap year:

$$Quarter = \left\lceil \frac{4}{53} WeekOfYear \right\rceil$$

Compute the Symmetry454 day within the Quarter as the ordinal day number minus the total number of days in prior elapsed quarters, using either of the following expressions. There are 91 days per quarter, except when

there is a leap week added to the end of the quarter, but the arithmetic yields the correct *DayOfQuarter* even when the given *FixedDate* is a day within the leap week:

$$DayOfQuarter = DayOfYear - 91 \times (Quarter - 1) = DayOfYear - 91 \times Quarter + 91$$

Compute the Symmetry454 week number within the quarter using the following expression:

$$WeekOfQuarter = \left\lfloor \frac{DayOfQuarter}{7} \right\rfloor$$

Compute the calendar month number within the quarter as follows:

4+**5**+4 $$MonthOfQuarter = \left\lfloor \frac{2}{9} WeekOfQuarter \right\rfloor$$

30+**31**+30 $$MonthOfQuarter = \left\lfloor \frac{2}{61} DayOfQuarter \right\rfloor = \left\lfloor \frac{DayOfQuarter}{30.5} \right\rfloor$$

The above expressions yield *MonthOfQuarter* = 4 during the leap week. **If using the normally recommended leap week append-to-December mode then take a maximum of 3 for *MonthOfQuarter*.**

Compute the calendar month using either of the following expressions, which will yield 13 if *MonthOfQuarter* was allowed to remain at 4 (as it would if the leap week is stand-alone as a 13th "mini-month"):

$$SymMonth = 3 \times (Quarter - 1) + MonthOfQuarter$$
$$= 3 \times Quarter + MonthOfQuarter - 3$$

Optionally, if required for the application, compute the number of days or weeks in *SymYear* as follows, based on the rule that regular years have 364 days (52 weeks) and leap years have 371 days (53 weeks):

IF *isSymLeapYear*(*SymYear*) THEN *DaysInYear* = 371 ELSE *DaysInYear* = 364

IF *isSymLeapYear*(*SymYear*) THEN *WeeksInYear* = 53 ELSE *WeeksInYear* = 52

or, more simply: $WeeksInYear = \frac{DaysInYear}{7}$ (because *DaysInYear* is always divisible by 7)

Optionally, if required for the application, for example to display the entire calendar month, compute the number of days in *SymMonth* as follows:

The expression $\left\lfloor \frac{SymMonth \text{ MOD} 3}{2} \right\rfloor$ or alternatively $\left\lfloor \frac{MonthOfQuarter \text{ MOD} 3}{2} \right\rfloor$ evaluates to 0 for a short month or 1 for a long month. We can safely use any programming language's MOD operator for that expression because *SymMonth* or alternatively *MonthOfQuarter* is guaranteed to be a small positive integer.

If the leap week mode has Irvember as a stand-alone 13th "mini-month" (an experimental option in *Kalendis*) and if *SymMonth* = *Irvember* (numerically 13) then set *DaysInMonth* = 7, otherwise we allow 28 days for all months, but add 7 to total 35 days for long months:

4+**5**+4 $$DaysInMonth = 28 + 7 \times \left\lfloor \frac{SymMonth \text{ MOD} 3}{2} \right\rfloor$$

For Symmetry010 we instead allow 30 days for all months, but adds one day for a mid-quarter month:

30+**31**+30

$$DaysInMonth = 30 + \left\lfloor \frac{SymMonth \bmod 3}{2} \right\rfloor$$

If the leap week mode has the leap week appended to December, as recommended, then add 7 days to the length of the month if it is a December in a leap year (where *December* = 12):

IF *SymMonth* = *December* THEN IF *isSymLeapYear*(*SymYear*) THEN *DaysInMonth* = *DaysInMonth* + 7

Also optionally, one may use *DaysInMonth* to compute the number of weeks in the month as follows, but this is only valid for the Symmetry454 calendar, which always has a whole number of weeks in every month:

4+**5**+4

$$WeeksInMonth = 4 + \left\lfloor \frac{SymMonth \bmod 3}{2} \right\rfloor = \frac{DaysInMonth}{7}$$

The Symmetry454 day number in the month is simply the ordinal day number of the Symmetry454 year minus the number of days elapsed before that month, using the *DaysBeforeMonth*( ) function that was defined above:

$$SymDay = DayOfYear - DaysBeforeMonth(SymMonth)$$

Optionally, if required for the application, compute the week number in *SymMonth* that contains *SymDay*. This value is valid for the Symmetry454 calendar, which has months comprised only of whole weeks, but it is invalid for the Symmetry010 calendar because within quarters it breaks weeks between months:

4+**5**+4

$$WeekOfMonth = \left| \frac{SymDay}{7} \right|$$

Processing of the *FixedToSym*( ) function is now complete, with the main results being *SymYear*, *SymMonth*, and *SymDay*, but also an assortment of other parameters are available if required for the application (those that are shown in **boldface** are valid for Symmetry454 but not Symmetry010): *Weekday*, *StartOfYear*, *WeeksInYear*, *WeekOfYear*, *DaysInYear*, *DayOfYear*, *Quarter*, *MonthOfQuarter*, *WeekOfQuarter*, *DayOfQuarter*, ***WeekOfMonth***, ***WeeksInMonth***, *DaysInMonth*. As a demonstration, *Kalendis* displays many of these parameters in the Symmetry454 Date Status Table, discussed later.

Determining the weekday for any Symmetry454 or Symmetry010 date

The universal way to determine the weekday for any date on any calendar that conserves the 7-day week is to use the *FixedToWeekdayNum*( ) function to compute the weekday (Sunday=0 through Saturday=6) as follows:

$$weekday = FixedToWeekdayNum(FixedDate)$$

If more convenient, one may substitute the *DayOfYear*, taking into account the calendar year starting weekday:

$$weekday = (DayOfYear + StartWeekday - 1) \bmod 7$$

For the recommended case of the calendar year starting on Monday, the line above simplifies to:

$$weekday = (DayOfYear + Monday - 1) \bmod 7 = DayOfYear \bmod 7$$

The Symmetry 454 calendar (but not the Symmetry010 calendar) starts every month on the same weekday, so the weekday is always the same for every date of every month. Therefore one can simply use the MOD operator to compute the weekday from the day number within the month as follows:

4+**5**+4 $$weekday = (SymDay + StartWeekday - 1) \text{ MOD } 7$$

For the recommended case of every month starting on Monday, the line above simplifies to:

4+**5**+4 $$weekday = (SymDay + Monday - 1) \text{ MOD } 7 = SymDay \text{ MOD } 7$$

Determining the date of Easter

Because of its longer calendar mean year, the established Gregorian Easter *computus* is not suitable for long-term use with the shorter mean year of the Symmetry454 calendar.

The World Council of Churches (WCC) recommends a common Easter date for all churches:

- Easter should fall on the Sunday following the first vernal full moon
- calculate the moments of the vernal equinox and full moon using the most accurate scientific methods
- reference the astronomical moments to the meridian of Jerusalem = Jerusalem local apparent time.

For implementation details, see Nachum Dershowitz and Edward M. Reingold: *Calendrical Calculations, 3rd edition*, chapter 20 *Astronomical Lunisolar Calendars*, page 325, section 20.1 *Astronomical Easter*. Use their *astronomical-easter*( ) function to return the fixed day number for the astronomical Easter, then use the *FixedToSym*( ) function described earlier to obtain the corresponding Symmetry454 date.

Such a ***lunisolar*** Easter is a moveable date, and causes all Christian ecclesiastical special days counted prior to Easter (such as Ash Wednesday, Passion Sunday, Palm Sunday, Holy Thursday, Good Friday, Holy Saturday) and events counted after Easter (such as Easter Monday, Rogation Sunday, Ascension Day) to also shift within the calendar year, in series with the shifting date of Easter. Such shifts would undermine the intended perpetual design of the Symmetry454 calendar.

In the Symmetry454 calendar with 52/293 leap rule the astronomical Easter can only land on a few possible Sunday dates: March 21st, March 28th, April 7th, April 14th, April 21st, or rarely on April 28th, of which Sunday April 7th is the most frequent. (The Symmetry010 calendar has the same astronomical Easter distribution except that it has March 23rd and March 30th as the last two Sundays in March.)

Therefore, Sunday April 7th is proposed as a perpetually fixed Easter date, which if adopted would also perpetually fix all Christian ecclesiastical special days counted before or after Easter. The fixed day number for this perpetually fixed Easter date in any desired year can be obtained very simply:

$$FixedEaster(SymYear) = SymToFixed(SymYear, April, 7)$$

If the symmetry calendar year always starts on Monday then April 7th is guaranteed to always be Sunday, and is the 98th ordinal day number in every year.

For a papal statement regarding the use of a fixed date for Easter on perpetual calendars see: "Appendix: A Declaration of the Second Ecumenical Council of the Vatican on Revision of the Calendar" at the end of the archived document "Constitution on the sacred liturgy *Sacrosanctum Concilium* solemnly promulgated by His Holiness Pope Paul VI on December 4, 1963":

> "The Second Ecumenical Sacred Council of the Vatican, recognizing the importance of the wishes expressed by many concerning the assignment of the feast of Easter to a fixed Sunday and concerning an unchanging calendar, having carefully considered the effects which could result from the introduction of a new calendar, declares as follows:
>
> 1. The Sacred Council would not object if the feast of Easter were assigned to a particular Sunday of the Gregorian Calendar, provided that those whom it may concern, especially the brethren who are not in communion with the Apostolic See, give their assent.
> 2. The sacred Council likewise declares that it does not oppose efforts designed to introduce a perpetual calendar into civil society.
>
> But among the various systems which are being suggested to stabilize a perpetual calendar and to introduce it into civil life, the Church has no objection only in the case of those systems which retain and safeguard a seven-day week with Sunday, without the introduction of any days outside the week, so that the succession of weeks may be left intact, unless there is question of the most serious reasons. Concerning these the Apostolic See shall judge."

As a demonstration, for the Symmetry454 and Symmetry010 calendars *Kalendis* reports astronomical Easter dates as "Uniform Easter" and reports the related ecclesiastical special days with the prefix "Uniform", for example "Uniform Good Friday". *Kalendis* also reports symmetry Sunday April 7th as "Fixed Easter" and reports the related ecclesiastical special days with the prefix "Fixed", for example "Fixed Easter Monday".

Kalendis allows the user to experimentally start the calendar year on a weekday other than Monday, which affects the fixed date for Easter because it must fall on a Sunday:

$$FixedEaster(\ SymYear\ ) = SymToFixed(\ SymYear, April, 10 - (\ 2 + StartingWeekday\ )\ \text{MOD}\ 7\ )$$

where *StartingWeekday* is set to one of: *Sunday* = 0, *Monday* = 1, … *Saturday* = 6.

The following table shows the possible values for the day-within-April expression:

Year Starts On	Weekday#	Fixed Easter Day in April	Ordinal Day in Year
Sunday	0	8th	99
Monday	1	7th	98
Tuesday	2	6th	97
Wednesday	3	5th	96
Thursday	4	4th	95
Friday	5	10th	101
Saturday	6	9th	100

The selected day in April in each case is based on the median Easter date from the distribution of astronomical Easter dates with that weekday starting the calendar year.

Computer programs that are intended to recognize or find astronomical Easter-related days on the Symmetry454 calendar can approach it two ways:

1) To find the date within the calendar year, determine the fixed day number for the astronomical Easter using the CCUE *astronomical-easter*( ) function, then add or subtract the offset in days to the desired Easter-related day, and finally use the *FixedToSym*( ) function to convert that back to the Symmetry454 date. For example, for Good Friday subtract 2 days from the Easter date, for Easter Monday add 1 day to the Easter date.

2) Starting with a Symmetry454 date, convert it to a fixed day number using the *SymToFixed*( ) function, then subtract the *astronomical-easter*( ) fixed day number. This yields a negative difference in days for all Easter-related days prior to Easter, zero if the date is Easter, or a positive difference in days for all Easter-related days after Easter. To recognize an Easter-related day just compare this difference to an appropriate list, which varies between Catholics, Orthodox, Protestants, *etc*..

Computer programs that are intended to recognize or find perpetually fixed Easter-related days on the Symmetry454 calendar can use the same approach, except for substituting the *FixedEaster*( ) function for the *astronomical-easter*( ) function, or they can directly work with the perpetually fixed dates of the Easter-related

days. For example, if Easter is always April 7th then Good Friday is always April 5th and Easter Monday is always April 8th.

Determining the dates of holidays and special events

The dates of non-Easter-related holidays and special events are perpetually fixed on the Symmetry454 calendar.

Computer programs that are intended to recognize or find holidays or special on the Symmetry454 calendar can work with the perpetually fixed dates of those events, so there is no need for rules specifying the n^{th} *k*-day in the month, where *n* is the week number in the month and *k* is the weekday, nor is there a need for calendrical calculations to support such rules. For example, USA Columbus Day / Canada Thanksgiving Day on the 2nd Monday in October is always the 8th of October:

ColumbusDay(*SymYear*) = *SymToFixed*(*SymYear*, *October*, 8)

A program that starts with a fixed day number can use the *FixedToSym*( ) function and then simply check the results for the desired holidays and special events. For example if *SymMonth* = 10 and *SymDay* = 8 then that day is USA Columbus Day / Canada Thanksgiving Day.

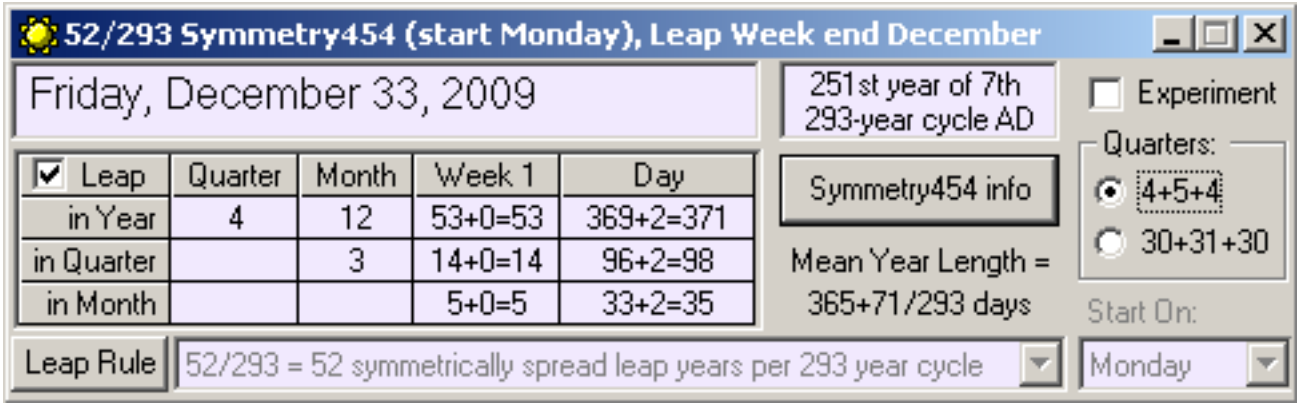
Dates that are traditionally on a specific day of the month are perpetually on the same weekday, for example Boxing Day on the 26th of December is always a Friday:

BoxingDay(*SymYear*) = *SymToFixed*(*SymYear*, *December*, 26)

A program that starts with a fixed day number can use the *FixedToSym*( ) function and then if *SymMonth* = 12 and *SymDay* = 26 then that day is Boxing Day.

The Symmetry454 / Symmetry010 date status table that is displayed by *Kalendis*

The freeware Windows program, *Kalendis*, a calendar calculator / converter, displays the details of the Sym454 or Sym010 date status as a table, for example, showing the leap week 2 days before the end of year 2009:



This table of information is useful for a variety of business statistics and calendar calculations. The internal coherency of the Symmetry454 and Symmetry010 calendars makes it easy to calculate all of the date status values, using very simple arithmetic expressions. Although most of the arithmetic has already been discussed above, here it is arranged as a table of formulae for direct comparison with the above screen shot:

Quarter	Month	Week #	Day	
$\left\lceil \frac{4}{53} WeekOfYear \right\rceil$	3 x *Quarter* + *MonthOfQuarter* – 3	$\left\lceil \frac{DayOfYear}{7} \right\rceil$	*FixedDate* – *StartOfYear* + 1	of Year
	See above for Sym454 or Sym010 *MonthOfQuarter* expression.	$\left\lceil \frac{DayOfQuarter}{7} \right\rceil$	*DayOfYear* + 91 – 91 x *Quarter*	of Quarter
	Week of Month only shown for Sym454 →	$\left\lceil \frac{DayOfMonth}{7} \right\rceil$	*DayOfYear* – *DaysBeforeMonth*	of Month

The column heading "Week 1" in the screen shot indicates that the displayed week is the first week of a repeated 4-week cycle. This is useful as a guide for regular payments that are every 2nd week or every 4th week, and so on. It is shown as "Week #" in the table above, and can be 1, 2, 3, or 4. The expression for the week # in the 4-week cycle is:

$$w = floor\left(\frac{(FixedDate - SymEpoch) \bmod 28)}{7} \right) + 1$$

For the n^{th} year of the m^{th} cycle:

$$n = amod(\ SymYear, YearsPerCycle\ ) \quad \text{and} \quad m = ceiling\left(\frac{SymYear}{YearsPerCycle} \right)$$

LeapYear = *isSymLeapYear*(*SymYear*) …depending on which leap rule is selected

Weeks…	Days…	
IF *LeapYear* THEN 53 ELSE 52	IF *LeapYear* THEN 371 ELSE 364	per Year
IF *Quarter* = 4 AND *LeapYear* THEN 14 ELSE 13	IF *Quarter* = 4 AND *LeapYear* THEN 98 ELSE 91	per Quarter
Sym454 only: $\frac{DaysInMonth}{7}$	See the *DaysInMonth*( ) formula above, which differs for Symmetry454 *vs* Symmetry010	per Month

Algorithms to check if an entered Symmetry454 or Symmetry010 date is valid

The algorithms of CCUE and the Symmetry454 and Symmetry010 functions presented herein don't "crash" if given invalid dates, such as specifying week = 53 when it isn't a leap year, or a month date of 29 to 35 (or more) when the month has only 28 days. Typically the fixed day number that would be calculated in such cases is logically correct if that date existed. Thus specifying the 29th of January as a Symmetry454 date will generate the fixed day number corresponding to the next day, February 1st, instead of triggering a run-time exception. Likewise specifying December 33rd for a non-leap year will generate the fixed day number for January 5th of the next year.

Nevertheless, programmers ought to prevent invalid date entries from being accepted or unintentionally converted to valid dates.

It is simplest to use the method suggested in CCUE: convert the calendar date to a fixed day number and then convert it back to a calendar date. If the returned date is unchanged (same as the original date) then the entered calendar date is valid. However if what comes back is a different date then either give the user an "invalid date" error message or do a detailed check as follows:

In the recommended leap week append-to-December mode, the Symmetry454 month number can range from 1 to 12. If the leap week is stand-alone after December, then the month number will be 13 during Irvember.

If a month short form is used then see the heading "Which date short forms are preferred for the Symmetry454 calendar?" in the document entitled *Frequently Asked Questions (FAQs) about the Symmetry454 and Symmetry010 Calendars*.

The entered day of the month can be validated against the calculated length allowed for that month, see the *DaysInMonth* formula, above.

There is no need for manual entry of the **weekday** for Symmetry454 or Symmetry010 dates, as it is always simple to directly derive it from the day of the month (Sym454 only), ordinal day number of the quarter or year, or the fixed day number (see the definition of the *FixedToWeekdayNum*( ) function and the heading "**Determining the Weekday for any Given Symmetry454 or Symmetry010**", above).

Example data for verification of calendar arithmetic functions

The following are examples of converted dates that programmers can use to verify calendar arithmetic:

Gregorian Date	CCUE *rata die*	Fixed Day 1 = Jan 1, 2001 AD	Julian Day (add 0.5)	Week Day	52/293 Symmetry454 Date	52/293 Symmetry010 Date	69/389 Symmetry454 Date	69/389 Symmetry010 Date
Apr 26, –121	–44,444	-774,929	1,676,980	Sat	Apr 27, –121	Apr 27, –121	Apr 27, –121	Apr 27, –121
Sep 27, –91	–33,333	-763,818	1,688,091	Mon	Sep 22, –91	Sep 24, –91	Sep 22, –91	Sep 24, –91
Sep 7, 122	44,444	-686,041	1,765,868	Mon	Sep 8, 122	Sep 10, 122	Sep 8, 122	Sep 10, 122
Jul 4, 1776	648,491	-81,994	2,369,915	Thu	Jul 4, 1776	Jul 4, 1776	Jul 4, 1776	Jul 4, 1776
Jul 1, 1867	681,724	-48,761	2,403,148	Mon	Jul 1, 1867	Jul 1, 1867	Jul 1, 1867	Jul 1, 1867
Oct 24, 1947	711,058	-19,427	2,432,482	Fri	Oct 26, 1947	Oct 26, 1947	Oct 26, 1947	Oct 26, 1947
Aug 10, 1995	728,515	-1,970	2,449,939	Thu	Aug 11, 1995	Aug 9, 1995	Aug 11, 1995	Aug 9, 1995
Feb 29, 2000	730,179	-306	2,451,603	Tue	Feb 30, 2000	Feb 28, 2000	Feb 30, 2000	Feb 28, 2000
May 2, 2004	731,703	1,218	2,453,127	Sun	May 7, 2004	May 5, 2004	May 7, 2004	May 5, 2004
Dec 31, 2004	731,946	1,461	2,453,370	Fri	Dec 33, 2004 or Irv 5, 2004	Dec 35, 2004 or Irv 5, 2004	Jan 5, 2005	Jan 5, 2005
Feb 20, 2020	737,475	6,990	2,458,899	Thu	Feb 25, 2020	Feb 23, 2020	Feb 25, 2020	Feb 23, 2020
Feb 2, 2222	811,236	80,751	2,532,660	Sat	Feb 6, 2222	Feb 4, 2222	Feb 6, 2222	Feb 4, 2222
Mar 1, 3333	1,217,048	486,563	2,938,472	Sun	Feb 35, 3333	Mar 2, 3333	Feb 35, 3333	Mar 2, 3333

Additional verification data for any date and any Symmetry454 leap rule can be generated using the *Kalendis* freeware program that is downloadable from the Symmetry454 web site at <http://individual.utoronto.ca/kalendis/kalendis.htm>.