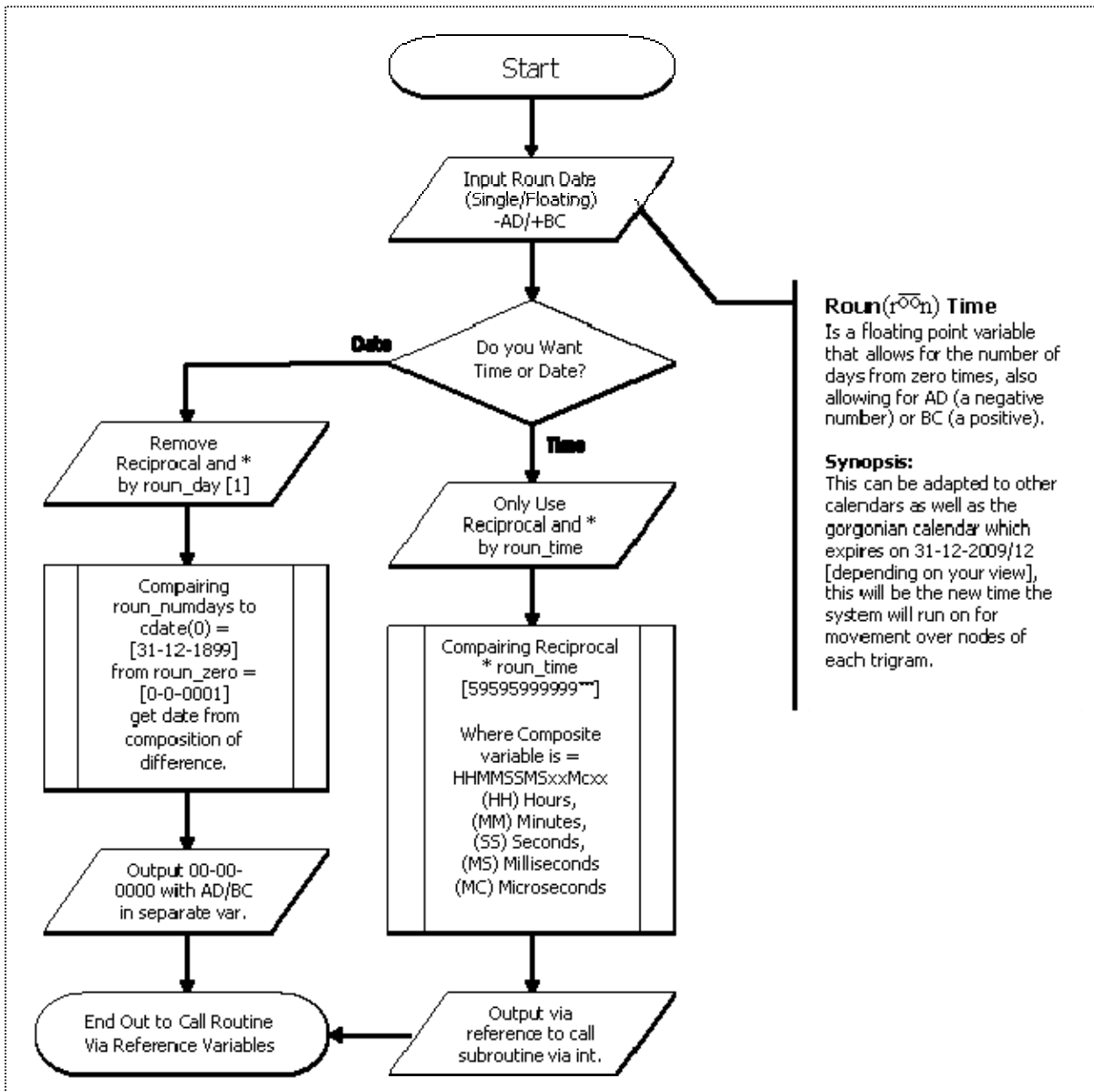


Roun Time

A Floating Date Explained

By Simon Roberts

<http://time.chronolabs.org.au/>



Roun Time – A floating point time date explained.

Union Movements

It was necessary to build a better date control that allows for more dynamic bases of trigram plotting as each one has movement of time per line, per segment, and the deeper in layers you go the more movement you have over more time.

As there will be natural language you will be able to plot from a group of trigrams that represent different segments of a text, for example if ant finds a word that it does not understand then it will prompt you for information for example if you input for and object like movement - for example a car. For example you input '30 times a day for around 100kms' into movement in the plot window under object. Ant not first knowing what this means will prompt you to break '%n times a %p for around %n%d'; removing common keywords from the array. This would come out as:-

'%n times a %p' = '[repetition]'
'%n%d' = '[distance]'

It will be necessary to only use a triangulated calendar for the initial presentation of the calendar set for roun time, that also present a hypothesis that what is an ouninoun, a micro small part of the roun time fractional component. Once we have a list of calendar's and dates of people it will be possible to divination what the date is at the moment of calendar. This precision variable will flow one motion a day, with the time being the fractional decimal place.

This will mean you can + - x / ^ log() Mod() or do whatever you want to the date easier as it is fractional, with whatever component the Roun time will be a integral component of the motion of measurement within this application and as it will be component driven in ActiveX it will be possible to put it in your computer software.

Of course the diagram above does not take into account that there is 365.25 days in a year, this is that it takes every year 365.25 days to revolve around the sun. Now it is good that existing date theorem take this into account, but the year for different planets and solar bodies need to be measured as well, this can be done with Roun time. See the variable that you times the reciprocal by to get the date components is not dynamic the amount of hours, minutes and seconds can also change on other planets, the idea is to have a date system that can be also used for astrological application.

So the working for something like the current date in Roun time to the number of years would simply in maths work both in the compiler as well as for general mathematicians this is to get the number of years in the date you divide the amount of days plus with the reciprocal which is the time by 365.25 where you would then round up and report this as the number of years within the roun time that is here.

This means that there is an opportunity to make error, but you will find you will have to make the following function as a day doesn't truly turn over the date correctly unless you take the .25 into account not to do this you have to look at the constant and at ability for it to only click over.

Of course the number of days in a revolution around a star or astral body is then not stored in a modern date as it has to support accurate pin-point calculation of other planetary or quantum bodies in the heavens. Therefore meaning that with the

Roun Time – A floating point time date explained.

Union Movements

current system of dates the year in the seed or floating point this time is not stored, more worked out on the spot with the number of roun_days in a year.

The Roun Date output will be designed to read from right to left like the Japanese as the date is seen mainly in the right hand corner of documents and machines. As I have been a big fan of star trek this could be seen as a method of being able to calculate a 'star-date' that is universally compatible with other planetary bodies in the heavens and for the calculations on our computers and blackboards.

Some Suggested Method of Displaying Roun time:

Method 1: YYYY-MMM-DD.TTTTTT = 2006-Dec-08.090909

Method 2: YYYY.MMM.DD.TTTTTT = 2006.Dec.08.090909

Method 3: YYYY.DDD/Y.TTTTTT = 2006.234.090909

Method 4: YYYY.DDD/L.TTTTTT = 2006.894.090909

I am attempting to not change much with roun time so it is easy to read and easy to understand, there is many opportunities for expansion on this text, but I am starting by providing an easy template for your mind to form image from.

At this stage the number of months in a year is 12, I am thinking about staying with this and the number of days will change as well; names of them could be initially presented in Latin origins, this is the most ideal scenario, which offers a new position, however, there is the disadvantage to this is that there has to be a new system introduced to the populus that some will find difficult to follow as they are set in their ways.

For the moment I will explore the maths of roun time as well as first the constants in roun time example here on our planetoid – Earth. These constants are for this planet we are on, time is unique in space in that like water, can flow at different rates and positions. This means that a normal calendar is not compatible with space travel as such, as a year is different on any planet your on – this is varied by how long it take to orbit the centre of it's solar system

Working

In this time we will be using the roun time of 24-June-2006 22:15:32.876 which makes the roun time represented by $R(t)$ in its entirety in this article. The value of this date would be $R(t) = 732866.132024733327829747123082$ which is equal to 24-June-2006 22:15:32.876.

This example shows the mathematical process of getting the fields seen in method 1 of the suggested way of displaying roun time. This will allow you to see the simple working with the floating point variable to get the time of the day or placement of this information.

Through the next couple of steps and examples, I will explore some simple methods with you of modulating the data in the roun time to receive large on the following page is a simple explanation of the functions and properties of roun that will allow for easy splining of time from the floating point time seed.

Roun Time – A floating point time date explained.

Union Movements

Most calendars are intended to follow some celestial cycle, such as the phases of the moon or the seasons. These cycles usually don't consist of a nice, integral number of days or months, so sometimes complicated rules for varying the length of years and months must be used to keep them in alignment with the target cycle.

The *epoch* of a calendar is its adopted starting point: the start of year 1 (or day 1) of the calendar. This is not necessarily the same as the day when the calendar was first used. For example, the currently used epoch for the Julian and Common calendars was invented about 500 years after the time indicated by that epoch.

An *era* is the period of time after an *epoch*, and so is connected to a particular calendar. To fully specify a particular day, one must specify not just the date corresponding to that day in the chosen calendar, but also which calendar one uses, for readers who might not use the same calendar system. This is done by specifying the era. For example, the period starting with year 1 of the Common calendar is the Common Era. Year 1432 of the Common calendar is referred to as year 1432 of the Common Era, or 1432 C.E.

Constants:

Roun_HoursInDay = $R_{(rh)}$ = '24'
Roun_MinutesInHour = $R_{(rm)}$ = '60'
Roun_SecondsInMinute = $R_{(rs)}$ = '60'
Roun_MillisecondsInSeconds = $R_{(rms)}$ = '1000'
Roun_MicrosecondsInMillisecond = $R_{(rmc)}$ = '1000'
Roun_DaysInYear = $R_{(rd)}$ = '365.25'
Roun_ZeroPoint_TimeWeight = $R_{(ztw)}$ = '0.0'
Roun_ZeroPoint_DayWeight = $R_{(zp)}$ = '1'

Factoring constants – $f_{(zad)}$:

Roun_ZAD_DaysFromEpoch = $R_{(zdfc)}$ = '732691.5'
Roun_ZAD_BCYear = $R_{(zady)}$ = '2006'
Roun_ZAD_BCMonth = $R_{(zadm)}$ = '1'
Roun_ZAD_BCDay = $R_{(zadd)}$ = '1'
Roun_ZAD_BCHour = $R_{(zadh)}$ = '0'
Roun_ZAD_BCMinute = $R_{(zadi)}$ = '0'
Roun_ZAD_BCSeconds = $R_{(zads)}$ = '0'
Roun_ZAD_BCDayName = $R_{(zaddn)}$ = 'Sunday'
Roun_ZAD_FollowingLeap = $R_{(zadfl)}$ = '2008'
Roun_Planet = $R_{(p)}$ = 'earth'
Roun_Planet_DiameterKM = $R_{(dkm)}$ = '127565.3'
Roun_timeseed_mask = $R_{(m)}$ = 'd.hhmmssmsxmcx'

Roun Time – A floating point time date explained.

Union Movements

The best way of seeing a floating point time is how it is store sitting there floating translucently flamboyant in the device using the time. A floating point was a new number that came out before the turn of the century, it was a mathematical magic with the way the floating point stores thousandths of million of decimal/reciprocal places in a number in less bits than a larger number, as well as being globally supported in a lot of UNIX databases.

Currently most if not all computer systems, use a form of UTC Time, Universal Time. However not all time you want to track is on the same spinning roundly thing like a solar system containing different bodies that have each there own $r_{(zad)}$ factorials , or a factory process that is a more linear system of time that has each its own sequences.

This means that a hypothesis could be draw that a better more portable system of date keeping should be kept, where the fundamental maths and system of timing can be derived from a floating point whole and time from the reciprocal of that precision number allow for absolute measurement and moment of movement.

For Example there are two primary different sequences that could be used here on the planet earth there is two different systems of the way the calendar can move for the leap year we have here on earth with the measurement of 365.25 days in each year. Let's explore sequence one for example with the two different system of calendars that can be applied on earth in roun time one is deficient the other is not.

Leap year System - Earth: (None Deficient)

$$F_1(\text{seq}) = \{ 31, 30, 31, 30, 30, 30, 31, 30, 31, 30, 31, 30 \} = 365 \text{ days}$$

$$F_2(\text{seq}) = \{ 31, 30, 31, 30, 31, 30, 31, 30, 31, 30, 31, 30 \} = 366 \text{ days}$$

$$F_3(\text{seq}) = \{ 31, 30, 31, 30, 30, 30, 31, 30, 31, 30, 31, 30 \} = 365 \text{ days}$$

$$F_4(\text{seq}) = \{ 31, 30, 31, 30, 30, 30, 31, 30, 31, 30, 31, 30 \} = 365 \text{ days}$$

Leap year System - Earth: (Deficient)

$$F_1(\text{seq}) = \{ 31, 30, 31, 30, 30, 30, 31, 30, 31, 30, 31, 30 \} = 365 \text{ days}$$

$$F_2(\text{seq}) = \{ 31, 30, 31, 30, 31, 30, 31, 30, 31, 30, 31, 30 \} = 366 \text{ days}$$

$$F_3(\text{seq}) = \{ 31, 30, 31, 30, 31, 30, 31, 30, 30, 30, 31, 30 \} = 365 \text{ days}$$

$$F_4(\text{seq}) = \{ 30, 30, 31, 30, 31, 30, 31, 30, 31, 30, 31, 30 \} = 365 \text{ days}$$

As the leap year system that is deficient would provide a closer system for farming and agriculture as the month and seasonal shifts will be similar but; the none deficient system as a function would be a less difficult system to navigate from a sociological perspective but it does offer closer match to seasonal shifts that are common place on a planet supporting life like earth.

The great thing about roun time is its portability, you can take it in whatever direction you need to, there are around 8 major calendars here on earth and from our test we found roun time was even compatible with the Mayan calendar which means that we can have a more versatile and more mathematically computable format of retrieving the date of the period.

A Floating Point Precision Number Explained:

A floating-point representation requires, first of all, a choice of *base* or radix for the significant, and a choice of the number of digits in the significant. In this article, the base will be denoted by b , and the number of digits, or precision, by p . The significant is a number consisting of p digits in radix b , so each digit lies between 0 and $b-1$. A base of 2 (that is, binary representation) is nearly always used in computers, though some computers use $b=16$. A base of 10 (that is, decimal representation) is used in the familiar scientific notation.

As an example, the revolution period of Jupiter's moon Io could be represented in scientific notation as 1.528535047×10^5 seconds. The string of digits "1528535047" is the significant, and the exponent is 5.

Now this could be represented as any of

$$\begin{aligned} &1528.535047 \times 10^2 \\ &1528535047. \times 10^{-4} \\ &.000001528535047 \times 10^{11} \end{aligned}$$

The main benefit of scientific notation is that it makes representations like the last one unnecessary, by allowing the decimal point to be put in a convenient place. True floating-point notation uses a precise specification that the point is always just to the right of the leftmost digit of the significant, so the correct representation is

$$1.528535047 \times 10^5$$

This, plus the requirement that the leftmost digit of the significant be nonzero, is called **normalization**. By doing this, one no longer needs to say where the point is; it is deduced from the exponent. In decimal floating-point notation with precision of 10, the revolution period of Io is simply

For any fixed precision p , the floating-point numbers can represent only a subset of the real numbers (This is so even if the possible exponents are any integer. In actual computer representations, there is a finite range for the exponent, and the total set of floating-point numbers is actually finite.) This incompleteness is the same as what people are used to when they express measurements to a certain number of decimal places, as in "The atomic weight of Hydrogen is 1.008". Correct analysis of floating-point arithmetic requires a more precise awareness of what this means. There is a common misconception that Floating-point numbers are imprecise. They are only approximations to the real numbers.

It is the floating-point **arithmetic operations**, not the numbers themselves, that are imprecise. Every floating-point number is in fact exact, and exactly represents a real number. For example, the floating-point representation of π , with binary precision 24.

Roun Time – A floating point time date explained. Union Movements

Function: RounCalendar(\$unixtime, \$gmt,[\$poffset, \$pweight, \$deficiency, \$timeset])

On the following page is the routine for calculating the true calendar system in roun time, the roun time calendar is by far the most accurate calendar to date, the following php code will calculate it. To calculate other calendar system, you can also use the function of *RounCalendar*.

We are still to name the 12 months in roun time, but they will be eventually assigned a name and hopefully be given there rightful place amongst the clocks and timepieces of the world.

```
// Union Movement for Time - doc: http://www.chronolabs.org.au/bin/roun-time-article.pdf
//Function Rountime Calendar

function RounCalendar($unix_time, $gmt, $poffset = '2008-05-11 10:05 AM', $pweight = '-20.2222222223', $deficiency='deficient', $timeset=
array("hours" => 24, "minutes" => 60, "seconds" => 60))
{
    // Code Segment 1 - Calculate Floating Point
    $time = $unix_time;
    if ($gmt>0){
        $gmt=-$gmt;
    } else {
        $gmt=abs($gmt);
    }
    $ptime = strtotime($poffset)+(60*60*$gmt);
    $weight = $pweight+(1*$gmt);
    $roun_xa = ($time)/(24*60*60);
    $roun_ya = $ptime/(24*60*60);
    $roun = (($roun_xa - $roun_ya) - $weight)+(microtime/999999);
    // Code Segment 2 - Set month day arrays
    $nonedeficient = array("seq1" => array(31,30,31,30,30,30,31,30,31,30,31,30),
        "seq2" => array(31,30,31,30,31,30,31,30,31,30,31,30),
        "seq3" => array(31,30,31,30,30,30,31,30,31,30,31,30),
        "seq4" => array(31,30,31,30,30,30,31,30,31,30,31,30));
    $deficient = array("seq1" => array(31,30,31,30,30,30,31,30,31,30,31,30),
        "seq2" => array(31,30,31,30,31,30,31,30,31,30,31,30),
        "seq3" => array(31,30,31,30,31,30,31,30,30,30,31,30),
        "seq4" => array(30,30,31,30,31,30,31,30,31,30,31,30));
    $monthusage = isset($deficiency) ? $deficiency : $deficient;
    // Code Segment 3 - Calculate month number, day number, day count etc
    foreach($monthusage as $key => $item){
        $i++;
        foreach($item as $numdays){
            $ttl_num=$ttl_num+$numdays;
            $ttl_num_months++;
        }
        // As well as Function MayanTihkalCalendar
        $revelutionsperyear = $ttl_num / $i;
        $numyears = floor((ceil($roun) / $revelutionsperyear));
        $avg_num_month = $ttl_num_months/$i;
        $jtl = abs(abs($roun) - ceil($revelutionsperyear*($numyears+1)));
        while($month=0){
            $day=0;
            $u=0;
            foreach($monthusage as $key => $item){
                $t=0;
                foreach($item as $numdays){
                    $t++;
                    $tt=0;
                    for($sh=1;$sh<=$numdays;$sh++){
                        $ii=$ii+1;
                        $tt++;
                        if ($ii==floor($jtl)){
                            if ($roun<0){
                                $daynum = $tt;
                                $month = $t;
                            } else {
                                $daynum = $numdays - ($tt-1);
                                $month = $avg_num_month - ($t-1);
                            }
                            $sequence = $key;
                            $nodaycount=true;
                        }
                    }
                    if ($nodaycount==false)
                        $day++;
                }
                $u++;
            }
        }
        $timer = substr($roun, strpos($roun, '.')+1, strlen($roun)-strpos($roun, '.')-1);
        $roun_out= $numyears.'/'.$month.'/'.$daynum.' '.$daynum.' '. floor(intval(substr($timer,0,2))/100*$timeset['hours']).':'.
        floor(intval(substr($timer,2,2))/100*$timeset['minutes']).':'.
        floor(intval(substr($timer,4,2))/100*$timeset['seconds']).substr($timer,6, strlen($timer)-6);
        $roun_obj = array('year'=>$numyears,'month'=>$month, 'day'=>$daynum, 'jtl'=>$jtl,
            'day_count'=>$day, 'hours'=>floor(intval(substr($timer,0,2))/100*$timeset['hours']), 'minute'=>
            floor(intval(substr($timer,2,2))/100*$timeset['minutes']), 'seconds'=>
            floor(intval(substr($timer,4,2))/100*$timeset['seconds']), 'microtime'=>substr($timer,6, strlen($timer)-6), 'strout'=>$roun_out);
        return $roun_obj;
    }
}
```

Systems and Processes – Time Segmenting

Traditionally time movements in mathematics are seen as a homogeneous movement in the old school algebraic deductions; however with understand in means of Qoun and other types of times in fields like quantum physics within this finite system of time. This means in Ounion movement is more now being isopolymorphic and allows for a free flowing system and comparative tables. This time dependant linear system within roun time but this does not always apply with other system and process which can always have an analytical test or series of tests that can prove the nature of that special segments or isopolymorphic movement that can be describe in our current day algebraic topography.

Here on the planet we have factors of orbital rate, planet size, spatial factorials that are defined in quantum physical evolution of these formulae is applied; but these are not always defined in movement having an orbital calendar system for days of the month and seasons of the year.

These other system in time would be the substitute for the ZAD environmental constants that could have fare differences in the environment needing to be track within its comparable time sequences and segments.

But a time defined in a quantum singularity maybe in theory a Qoun time segment as it would have none radial properties in most instances, Quantum singularities have been produced in test environment often with damage to life and limb.

Other such segments exists, you would be able to use the segmentation method to build the basis of a time zone system based on the factors in the $R_{(zad)}$ Constants Grouping. At this stage I will not explore the more complex motion of time, such as in space itself where there are other factors in the physics as well as the bountiful supply of possible applicable formulas that can be used in factor Qoun time $Q_{(zad)}$ factors.

A Qoun time would be similar to the system of time keeping that was seen in some well known 21st century treknology known as the star date. This was a none calendar system that could be used to navigate nautical space, that layer of area in all accomplishment that we have yet to reach as a people here in the 22nd century as an explorative deep-space missions.

To complete the segmenting of Roun-time to a universally compatible system if predetermined month names are not used then a combination of the dates of the existing calendars, there epochs and figurer heads in historical and philosophical standpoints as well as royalty and ancient as well as post modern history must be factored in.

To truly divinise a result you must look at all the deithy representation in the time dependant linear system a chart of time must be plotted with any similarities in deithy representation in the seasons, festivity and spiritual ramification must be factored in.

Roun Time – A floating point time date explained.

Union Movements

From this the list of gods and deithy as well as figure heads can be cross pollinated and then used to form the basis of names of the month. This as you can see from data collection is a big task and quiet laborious even for the most dedicated mathematics analyst.

Step I – Time (TTTTTT)

First remove the whole number from the roun time and now let $R_{(r)}$ = the reciprocal of the floating point. Now we have to work out the value of the floating point in the planetary bodies by subtracting 1 from $R_{(rh)}$, $R_{(rm)}$, $R_{(rs)}$, $R_{(rms)}$ & $R_{(rmc)}$. So the time in a string output with the decimal output would look like:

$$R_{(w)} = (((R_{(rh)}-1)/100) + ((R_{(rm)} -1)/10000) + ((R_{(rs)} -1)/1000000) + ((R_{(rms)} -1)/1000000000) + ((R_{(rmc)} -1)/ 1000000000000))$$

$$t_{(i)} = \frac{R_{(r)}}{R_{(w)}}$$

This will give you $t_{(i)}$ which looking something like this 0.221532876918 where you have removed the days from the roun time and divided like seen in the process at the beginning of this article.

Lets look at some of the working: Now where $R_{(r)} = 0.132024733327829747123082$ and $R_{(w)} = 0.595959999999$ then the working to get Hours minutes and seconds is seen above in Step I now in this working here with $R_{(r)}$ & $R_{(w)}$ are set then $t_{(i)} = 0.221532876918$ then $t_{(i)} = 0.HHMMSSMSxxMCxx$. that means the time in this example is in 24hour clock – 22:15:32.876 otherwise known as quarter past ten and thirty two seconds.

This is how you deduct time or the motion of a single day with roun time, with this method you can convert times between astral bodies like planets and have a similar star-date that is compatible as well as comparable to another time in another solar system. This floating point is much like the existing Timestamps done by the bios clock, however this timestamp does not need conversion or any functions to add and subtract or divide or multiple the instance of time, as it is a floating point not a timeseed, more so a floating - timeseed.

if you wanted to for example find the difference between two dates, normally you would have to use datediff or a similar function within, the language you are coding in, however with roun time all you have to do is subtract from one and another and power out the remainder of the two variable being subtracted from one another.

When you calculate the time you can have massive reciprocals that are absolutely micro-fine measurements of time, that are possibly very hard, if not impossible to track with a domestic home computer. It would take dedicated processors to do such a massive amount of maths to.

Roun Time – A floating point time date explained.

Union Movements

Warning: Although this system is designed to have milliseconds and microseconds into its computation systems building a timer system to work with such force that the count is accurate would have to be achieved in A+ or similar low level compiler.

Step 2 – Date of year

The month system is going to be broken up into 30 day blocks, there is a tolerance of 5 days outside a leap year and 6 days during a leap year the months will have the following sequences that consist of either 30 or 31 only days in a month. In a leap year the forth motion of the orbit of earth in this example is what adds the day onto 'May' this is when $0.25 * 4 = 1$ day when $R_{(rd)} = 365.25$ days in year.

The calendar should move in a motion of either one day to three days should be found in the calendars motion of days per month. In this example we will show you how to deduct the day as well as the year from the roun time $R_{(t)}$.

The calendar starts on $R_{(zdfc)}$ which means by the default ZAD constants you will be able to determine the day, whether it is a leap year, if the number is negative then the death of a religious figure like AD/BC in the Gregorian calendar the all powerful deity known as Jesus. PO or Point of Origin or the Epoch as it is called in UTC, Georgian and Julius Calendars, will exist in other species and sentient life forms, this is a question that is not unique to here and now on earth and need to be built into the measure.

$$F_1(\text{seq}) = \{ 31, 30, 31, 30, 30, 30, 31, 30, 31, 30, 31, 30 \} = 365 \text{ days}$$

$$F_2(\text{seq}) = \{ 31, 30, 31, 30, 31, 30, 31, 30, 31, 30, 31, 30 \} = 366 \text{ days}$$

Based in $R_{(t)}$ divide by $R_{(rd)}$ to get the year that the date is based in so rounded up then if the year is represented by $T_{(y)}$ then the method below will describe how to get the year, similarly day is seen as $T_{(d)}$, Month $T_{(m)}$:

$$T_{(y)} = R_{(t)} / R_{(rd)}$$

$R_{(zadff)}$ is the next leap year that follows the ZAD indicator for what timeseed represent the beginning of the calendar. Based on a function to test for leap year then use month sequence correlating to the correct number of days in a year. Comparing $R_{(t)}$ to $R_{(zad)}$; calculate from the ZAD Indicators that exist for the conditions of $R_{(zad)}$, you will be able to find month, day, day name from this function the test function will be represented by $F^1(x)$.

Solve: What is the measurement in this formulae represented by (...):

$$T_{(...)} = F^1_y(R_{(t)_y})$$

Solve: What is the measurement in this formulae represented by (...):

$$T_{(...)} = F^1_m(R_{(t)_y})$$

Roun Time – A floating point time date explained.

Union Movements

Solve: *What is the measurement in this formulae represented by (...):*

$$T(\dots) = F^l_d(R(t)_y)$$

Roun time is portable into the major calendars listed below, these are a synopsis of each calendar and the scales of years that are used in it. These will be support by the roun time control developed by Cronolabs Australia. Roun-time at this stage does not have names for the month itself officially, these will be divinisid from data collected in the zero-point application to be first released late 2006.

Common Calendar

We refer to the calendar system that is in most widespread international use around the world today as the Common Calendar. The Common Calendar consists of two parts: it uses the rules of the Julian Calendar for dates up to 4 October 1582, and the rules of the Gregorian Calendar for dates starting with 15 October 1582. In the Common Calendar, 15 October 1582 was the day immediately following 4 October 1582. The era of this calendar is referred to as the Common Era (C.E.).

Julian Calendar

The Julian Calendar is a solar calendar that was introduced in the Roman Empire, in the year now referred to (in the Common Calendar) as -44, during the reign of Julius Caesar, for whom it and its fifth month are named. The calendar was slightly reformed by emperor August, for whom the sixth month is named. After this reform, which was complete by the year now referred to as year 8 C.E., this calendar was used continuously in many regions of Europe until 1582 C.E., though there was variation (over both time and region) in such things as the used epoch, the used day count within each month, and which day was the first day of the year. The following description reflects the modern choices for these things.

Number	Month Name	Length
1	January	31
2	February	28 (29 in a leap year)
3	March	31
4	April	30
5	May	31
6	June	30
7	July	31
8	August	31
9	September	30
10	October	31
11	November	30
12	December	31

In this calendar, each year consists of 12 months with between 28 and 31 days per month, with new days starting at midnight. Each new day starts at midnight. The English names of these months and their lengths are listed in the following table.

In this calendar, years have either 365 or 366 days. The longer years, called leap years, occur every fourth year, whenever the year count is evenly divisible by 4. For example, the years 1000 and 1004 were leap years. The extra day is inserted at the end of the month of February, as the 29th day of February. January 1 is now the beginning of the new year, though this was not always the case.

Many different epochs have been used in conjunction with this calendar, for example, the assumed year of the foundation of the city of Rome (-753 C.E., referred to as year 1 A.U.C. -- ab urbe condita), or the beginning of the reign of emperor Diocletian (284 C.E.), or the approximate birth year of the Messiah of the Christian faith (1 C.E. = 1 A.D. -- anno Domini)

For dating historical events, use of the final rules of the Julian calendar (with the Common Era) is extended into the past. This anachronistic calendar is referred to as the Julian Proleptic Calendar, or, if confusion is unlikely, as the Julian Calendar.

Lunar Calendar

This is a lunar calendar, represented by the (fractional) count of New Moons since the first New Moon after JD 0 (Julian Day Numbers). The count is based on the mean motion of the Moon, disregarding short-term perturbations. In this calendar, an integer date corresponds (approximately) to New Moon, and a date ending in .5 corresponds to a Full Moon.

Roun Time – A floating point time date explained.

Union Movements

Gregorian Calendar

The Gregorian Calendar was introduced in the Catholic parts of Europe in 1582 C.E. by Pope Gregory XIII (then the religious leader of the Roman Catholic faith) as an improvement upon the Julian Calendar to keep the average length of the calendar year better in line with the seasons.

The rules, months, and days of the Gregorian calendar are the same as those of the Julian Calendar, except for the leap year rules. In the Gregorian calendar, a year is a leap year if the year number is evenly divisible by 4, but not if the year number is evenly divisible by 100, and this last exception must not be applied if the year number is evenly divisible by 400. For example, 1600 and 2000 are leap years, but 1700, 1800, and 1900 are not.

Islamic Calendar

The Islamic calendar is a strict lunar calendar. The beginning of a new month is tied to the first sighting of the lunar crescent in the evening after a New Moon. Because the beginning of the month is determined by observation, it cannot be accurately predicted. However, for secular use a tabular calendar is available that is determined by fixed rules. This tabular calendar is described below. The Islamic tabular calendar has 12 months per year, that each have 29 or 30 days, starting at sunset. The month names and lengths in days are listed in the following table.

Number	Month Name	Length
1	Muharram	30
2	Safar	29
3	Rabi`a I	30
4	Rabi`a II	29
5	Jumada I	30
6	Jumada II	29
7	Rajab	30
8	Sha`ban	29
9	Ramadan	30
10	Shawwal	29
11	Dhu al-Q`adah	30
12	Dhu al-Hijjah	29 (30 in a leap year)

There are 11 leap years in a fixed cycle of 30 years. In a leap year, the extra day is added at the end of the month of Dhu al-Hijjah. The epoch of the calendar is sunset of 15 July 622 C.E. and "year of the Era of the Hegira" may be abbreviated to A.H. (= Anno Hegirae). The epoch coincides with the migration of the Prophet Mohammed from Mecca to Medina. The Roun GetCalendar() Function takes an A.H. date to refer to the date that is current at noontime. The first noon after the epoch was the noon of 16 July 622 C.E., so the Roun GetCalendar() Function equates 1 Muharram 1 A.H. with 16 July 622 C.E.

Egyptian Calendar

Ancient Egyptians used a calendar that had 365 days in a year, without exceptions. The year was divided into 12 months of 30 days each, plus 5 extra days (referred to by the ancient Greeks as the epagomenai) after the last month. Because of its great regularity, this calendar was used by ancient Greek and European astronomers until only a few centuries ago.

The names and lengths of the months of the Egyptian calendar are listed in the following table.

The Roun GetCalendar() Function regards the epagomenai as a 13th month. The era used for this calendar by the Roun GetCalendar() Function is the Era of Nabonassar, used by Ptolemy, with epoch 26 February -746 C.E. Other eras that have been used elsewhere are the Era of Philippos (which marks the death of Alexander the Great) starting in year 425 of Nabonassar, the Era of emperor Hadrian of Rome, starting in year 864 of Nabonassar, and the Era of emperor Antoninus of Rome, starting in year 885 of Nabonassar.

Number	Name	Length
1	Thoth	30
2	Phaophi	30
3	Athyr	30
4	Choiak	30
5	Tybi	30
6	Mecheir	30
7	Phamenoth	30
8	Pharmuthi	30
9	Pachon	30
10	Payni	30
11	Epiphi	30
12	Mesore	30
	epagomenai	5

Roun Time – A floating point time date explained.

Union Movements

Latin Calendar

The ancient Romans started the Julian calendar which eventually evolved into the Common calendar. However, the method of the Romans for designating dates in their calendar was quite different from ours. We refer to the Common calendar with the Roman way of designating dates (in the Latin language) as the "Latin Calendar". The Romans did not know Arabic numerals. They wrote numbers using letters, according to the following table.

Letter	Numeric Value
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

If a "smaller" letter (i.e., earlier in the table) follows a "bigger" one (i.e., later in the table), then the values add up, but if a smaller letter precedes a bigger one, then its value must be subtracted from the total. For example, DCX stands for $500 + 100 + 10 = 610$, but CDX stands for $500 - 100 + 10 = 410$.

The Latin names for the months are listed in the following table. They are similar to the English month names, which are derived from them. In the Latin language, the way to write a word -- and especially the last part of a word -- depends on the context. The table lists three forms that are useful in the calendar.

Three days in each month had names: the Kalends (hence calendar), the Nones, and the Idus (as in "Beware the Ides of March"). The Kalends was the first name of a month. The Idus was the 13th day in most months, but the 15th day in March, May, July, and October. The Nones was 8 days before the Idus, so it was the 5th or 7th day of the month. These days

Number	Latin Month Names			English Month Name
1	Ianuarus	Ianuariis	Ianuaris	January
2	Februarius	Februariis	Februarius	February
3	Martius	Martiis	Martias	March
4	Aprilis	Aprilibus	Apriles	April
5	Maius	Maiis	Maias	May
6	Iunius	Iuniis	Iunias	June
7	Iulius	Iuliis	Iulias	July
8	Augustus	Augustis	Augustas	August
9	September	Septembribus	Septembres	September
10	October	Octobribus	Octobres	October
11	November	Novembribus	Novembres	November
12	December	Decembribus	Decembres	December

were referred to using month names from the second column of the table; for example Kalendae Ianuariis, Nonae Februariis, Idibus Martiis. The day preceding one of these days was referred to using month names from the third column of the table, after the word Pridie; for example, Pridie Kalendas Apriles, Pridie Nonae Maias, Pridie Idus Iunias.

The Romans indicated other days of the month by counting backwards from the next later Kalends, Nones, or Idus. This means that days in the second half of every month (after the Idus) would be referred to as "so many days before the Kalends of the next month". In addition, the Romans counted inclusive. In figuring out the difference between two numbers, they'd count both the first and the last numbers. For example, to get from today to tomorrow, the Romans would count two days rather than just one. So, the 30th day of June, which is the day before the Kalends (first day) of July, would be referred to as Pridie Kalendas Iulias, and the day before that (the 29th of June) as Ante Diem III Kalendas Iulias. The "ante diem" means something like "the earlier day".

The Romans used to count years from the (mythical) year of the founding of the city of Rome in year -751 of the Common Era. They referred to a year count in the era as Ab Urbe Condita ("since the founding of the City"), abbreviated to A.U.C. However, our Latin calendar uses the same era as the Common calendar. The year number is introduced by the word "Anno" (year). As an example of a complete date, the 15th of December of 1965 is referred to as "Ante Diem XVIII Kalendas Ianuarias Anno MCMLXVI", which translates loosely as "The 18th inclusive day before the Kalends of January of the year 1966".

The Romans did not know of the number zero or of negative numbers. Such year numbers are printed in the Latin calendar using the usual Arabic numerals. In addition, numbers greater than or equal to 4000 are also printed using Arabic numerals.

Roun Time – A floating point time date explained.

Union Movements

Hebrew Calendar

The Hebrew calendar is a lunisolar calendar. Its current rules were pronounced in the 4th century C.E. by Patriarch Hillel II. New days start at sunset, new months start at a New Moon, and new years start in the northern hemisphere spring.

A Hebrew calendar year has 12 or 13 months, that each have 29 or 30 days. The month names and lengths in days are listed in the following table. Biblical tradition lists Nisan as the first month but has the new year start on the first day of Tishri. The Roun GetCalendar() Function counts months from the start of the year in Tishri. The month numbers, names, and lengths (in days) are listed in the following table

There are 7 leap years in a fixed cycle of 19 years. The Roun GetCalendar() Function treats Adar and Adar II as a single month, which has 30 days in a non-leap year and 59 days in a leap year. This way, Nisan and later months always corresponds to the same month number, regardless of whether the year is a leap year or not.

The epoch of the Hebrew calendar is sunset of 6 October -3760 C.E., which was taken to be the date of the creation of the world. The Era of the Hebrew calendar is referred to as A.M. (= Anno Mundi). The first noon after the epoch was the noon of 7 October -3760 C.E., so the Roun GetCalendar() Function equates 1 Tishri 1 A.M. with 7 October -3760 C.E.

Number	Month Name	Length
1	Tishri	30
2	Heshvan	29 or 30
3	Kislev	29 or 30
4	Tevet	29
5	Shevat	30
6	Adar	30
	Adar II	29
(only present in leap years)		
7	Nisan	30
8	Iyyar	29
9	Sivan	30
10	Tammuz	29
11	Av	30
12	Elul	29

Leap days may be inserted at the ends of the months of Heshvan and Kislev, and a leap month Adar II may be inserted at the end of the month of Adar (I). This means that any given year may contain six different numbers of days, as listed in the following table

Days	Designation
353	deficient ordinary year
354	regular ordinary year
355	complete ordinary year
383	deficient leap year
384	regular leap year
385	complete leap year

Julian Day Number

Julian Day Numbers were introduced by astronomers in the 19th century C.E. as a continuous day numbering scheme without years or leap days. The epoch (the start of day 0) is 1 January -4712 C.E. at 12:00:00 TT (by recommendation of the International Astronomical Union). For precise astronomical calculation, fractional Julian day numbers are used. The Roun GetCalendar() Function returns fractional Julian day numbers, assuming that the fractional part of the day specification in the source calendar is measured since the midnight before the noon corresponding to that date. This means that an integer date in the source calendar corresponds to a JD (= Julian Date) ending in ".5". To get the integer count, round to the nearest integer; round JDs ending in ".5" up to the next greater integer.

For the official definition of the Julian Date by the International Astronomical Union, see at <http://maia.usno.navy.mil/iauc19/iaures.html#B1>

Mayan Tikal Calendar

The peoples of Central America used a great number of calendar systems, but they all followed the same overall pattern. The calendar described here is the Mayan "Tikal" calendar.

The Tikal calendar used a cycle of 20 days with a name for each day in the cycle, and a cycle of 13 days with a cardinal number for each day in the cycle (starting with 1). These two cycles were counted concurrently, so that after day "6 Ik" followed day "7 Akbal" and then "8 Kan". A particular combination recurred after 260 days, which period was called the "tzol kin" in the Yucatecan language. The names of the days in the 20-day cycle were as follows: Imix, Ik, Akbal, Kan, Chicchan, Cimi, Manik, Lamat, Muluc, Oc, Chuen, Eb, Ben, Ix, Men, Cib, Caban, Etz'nab, Cauac, Ahau.

There was also a year count, called haab, with a year of 365 days divided into 18 months of 20 days each and a 19th month with 5 days only. The months had names, and the days had numbers, and

Roun Time – A floating point time date explained.

Union Movements

these were counted as we are used to today, so after day "1 Pop" followed day "2 Pop" and so on. However, the numbers started at 0 instead of 1. The Mayan names of the months were: Pop, Uo, Zip, Zotz, Tzec, Xul, Yaxkin, Mol, Ch'en, Yax, Zac, Ceh, Mac, Kankin, Muan, Pax, Kayab, Cumku, Uayeb. There were no leap years in the central American calendars, so the year count ran out of step with the seasons by about one day every four years.

A particular date was usually identified by its position in both the tzol kin and the haab, for instance as "6 lk 2 Pop", and for the next day "7 Akbal 3 Pop". After 52 years (of 365 days) the same tzol-kin/year-count designation would return. This period is often referred to as a "calendar round" or a "century".

In many cases, Mayan monuments display dates in only the calendar-round manner, which means that these dates return every 52 years (of 365 days). This means that we can pinpoint those dates in the modern calendar only up to a multiple of 52 years.

The Spanish conquistadores who conquered Central America in the 16th century ordered the destruction of much of the Mayan written records, and the precise correlation between the Mayan calendars and modern calendars is therefore not exactly known.

The Roun GetCalendar() Function can return, in text form, the Tikal designation (tzol kin -- haab) for any date in the modern calendars, based on the most widely accepted correlation between the Mayan and modern calendars.

The Mayan Long Count

The Mayan Long Count is a calendar consisting of 5 cycles that indicate the number of days since the last beginning of the full cycle. The name, definition, and length of each cycle is indicated in the following table.

Name	Definition	Length
kin	= 1 day	1 day
uinal	= 20 kin	20 days
tun	= 18 uinal	360 days
katun	= 20 tun	7,200 days = about 20 years
baktun	= 20 katun	144,000 days = about 394 years
(full cycle)	= 20 baktun	2,880,000 days = about 7885 years

A particular date in the Long Count is written as a set of five numbers, one for each subcycle, separated by periods (.). For example, the date 1.2.3.4.5 means 1 baktun, 2 katun, 3 tun, 4 uinal, 5 kin after the beginning of the full cycle. The beginning of the last cycle, at long count 0.0.0.0, is thought to correspond to 6 September -3113 C.E.

The Long Count 13.0.0.0.0 corresponds to 21 December 2012 C.E., and the current full cycle will be complete on 13 October 4772 C.E. The Roun GetCalendar() Function can return, in text form, the Long Count corresponding to any date.

Annotation: Time Scales

Various time scales are in use. International Atomic Time (TAI) is a uniform time scale with a unit of one SI second. It was implemented around 1960. Each TAI minute is 60 SI seconds, each TAI hour equals 60 TAI minutes, and each TAI day equals 24 TAI hours. Coordinated Universal Time (UTC) is the basis of most administrative times on Earth. Its units are the SI second, the minute, the hour, and the day, and it differs from TAI by an integral number of seconds. By infrequent and irregular insertion of leap seconds, the UTC day start is tied to the Earth's rotation. The length of specific UTC minutes, hours, and days may therefore differ from TAI minutes, hours, and days. Universal Time (UT, UT1) is a non-uniform time scale tied to the Earth's rotation. It has no leap seconds, so the length of its unit (the UT second) depends on the Earth's rotation. UT differs from UTC by less than 0.9 seconds through the judicious insertion of leap seconds in UTC. Terrestrial Dynamical Time (TDT) -- or Terrestrial Time (TT) for short -- was implemented in 1984 as the dynamical time scale for geocentric phenomena. Barycentric Dynamical Time (TDB) was implemented in 1984 as the dynamical time scale for solar-system barycentric phenomena. The difference between TDT and TDB is always less than 0.002 seconds. The difference between TDT and TAI is fixed at about 32 seconds. Before 1984, Ephemeris Time (ET) was used as a uniform time scale for ephemerides, instead of TDT and TDB. TAI time is bases of measure in UTC (Universal Time) and will be used as a precursor for Roun time.