



Unit

Moon

Description

The unit Moon.pas contains a collection of [astronomical algorithms](#) together with a [visual component](#) tmoon which can be used to easily show the moon picture in various applications.

But this all would have been impossible without the book "[Astronomical Algorithms](#)" by Jean Meeus, where all the algorithms used in this component are listed originally. So if you want to get a deeper understanding in how the calculations work, what are the limits of it, want to find more algorithms, etc., this book is highly recommended.

If you are interested in more information about the calendar, especially the history of the now commonly used gregorian calendar, I recommend the book "[Marking Time](#)" by Duncan Steel; or for a lot of calendrical algorithms the book "[Calendrical Calculations](#)" by Nachum Dershowitz and Edward M. Reingold.

For updates and a listing of bugs (often with patches or work-arounds), a more complete bibliography, various internet resources, etc., check the [moon webpage](#). This software comes as freeware, you may use it any way you like. However there is no warranty whatsoever, I can only promise I did my best to avoid bugs. If you want to redistribute this component only do it completely with all the files in the archive. Finally if you like this component all I ask you to do is send me a nice postcard of your hometown - other presents are also welcome but a postcard is enough.

Component Usage

There are two ways to install the components - either you use the ready-made package file (be sure to use the one fitting to your Delphi version), or you can install manually into your favourite package (in Delphi 1 and 2 there is only one component library). Select "Install Component" in the

menu and then open the file `moon_reg.pas`. You will then see this component in the "Custom" tab of the component list, so you can easily drop it onto your application. The component itself is in the unit `mooncomp.pas`.

Starting with Delphi 6/Kylix the new cross-platform visual library CLX is supported, the unit `qmoonreg.pas` makes the component available for this library as well.

Algorithm Usage

There is a big selection of astronomical algorithms included in the unit `moon.pas`, both for the functions originally included in the Moontool, as well as plenty of additional ones very useful for both astronomical as well as calendrical applications. These can be used independent from the component.

There are two compiler switches which can be used to modify the internal working of the algorithms. The first one is `nomath`, which is used to optionally use the unit `math` or not. This unit is not included in every version of Delphi, so the default setting is to use my own implementation of the needed math algorithms. If you have the unit `math` and wish to use it instead, you need to remove the following line from the header of the unit `ah_math`:

```
(*$define nomath *)
```

The second one is the switch called `meeus` used for the calculation of the sun position by the VSOP planetary theory as well as the ELP moon theory. Meeus used a truncated version of it by ignoring those terms only needed for higher accuracy, but for most cases the limited accuracy is enough. In case you want to use the full VSOP instead you can switch off the compiler switch `meeus` in the `vsop` unit. Note that this will increase the size of your executable quite a bit, it will increase the calculation times sometimes noticeably, and it's not possible for Delphi 1 due to the limited size of the data segment.

As the full terms of the ELP moon theory would increase the size of this package too much they are only available for download on my homepage, together with the VSOP planetary terms for the other planets both in the Meeus and the full theory.

Thanks

A great number of people contributed to this component by reporting bugs, suggesting enhancements or even sending code I just needed to include. So instead of listing those names I can still remember and forgetting many others I just thank everybody who wrote me, and hope you will apologize me if I didn't answered your email...

History

Version	Date	Changes
V1.0	1997.04.03	<ul style="list-style-type: none">• first published version
V1.1	1997-05-21	<ul style="list-style-type: none">• bug with align property fixed• moontool available in 16bit as well• daylight saving in moontool corrected• added calculation of seasons, moon/sunrise and -set, perigee and apogee and eclipses
V1.2	1997-12-07	<ul style="list-style-type: none">• new icon property• 16x16 bitmap• second page in Moontool with the new additional data• Rotation of the moon image• "Color" bitmaps• New functions for horizontal coordinates of sun and moon• Twilight (civil, nautical, astronomical)
V2.0	2001-07-07	<ul style="list-style-type: none">• Easter date for gregorian and julian calendar• Pesach date and jewish calendar functions• Chinese calendar• Perihel and Aphel• Corrected TDateTime functions• Location database in Moontool• Moontool set date/time dialog• Online help• Time difference UTC vs. dynamic time• Ecliptic and equatorial sun/moon coordinates• Coordinate transformations• Refraction• Physical ephemerides of moon• Passages through the nodes
V2.1	2002-03-24	<ul style="list-style-type: none">• Equation of time• Distance on earth• Zodiac signs• Names of full moons• DUnit self-testing

- Changed inheritance to TGraphicControl
- New drawing style msMonochrome, transparency
- CLX support

And of course, every version fixes bugs of the previous ones, these are not mentioned in this list.

How to contact

Andreas Hörstemeier
Mefferdatisstraße 16-18
52062 Aachen
Germany

andy@hoerstemeier.de
<http://www.hoerstemeier.cop>

I try to answer as many emails as possible, but as all this programming is done as a hobby please don't be angry if I don't answer promptly - I read all the emails however, and every comment is welcome.

I have created a mailing list which I use to send announcements of new versions of my components, so if you like to get such a notification send an email to ah-delphi-request@scp.de.

Please don't send me questions about Delphi or programming in general, I cannot answer them due to lack of time, you will have much better chances to get an answer by going to the Borland newsgroups at <http://www.borland.com/newsgroups> or the standard Usenet newsgroups.

Age of the moon

[Algorithms](#)

[Top](#) [Previous](#) [Next](#)

Calculates the age of the moon

```
function AgeOfMoon(Date:TDatetime):extended;  
function AgeOfMoonWalker(date:TDatetime):extended;
```

Description

Calculates the age of the moon (in days) for the given time. I did find two different definitions for this number, thus there are two functions for calculating it. The correct definition of the age of the moon seems to be the straight-forward time since the last new moon.

However John Walker in his original Moontool did use a different one, which describes the position of the terminator on the moon - the apparent longitude of the moon - normalized on the mean length of the month instead of 360 degrees. The mean length of a month is 29.530589 days. As the moon orbit is both elliptical and also has quite a lot of variation due to perturbations from the sun both values differ significantly - only for an unperturbed moon in circular orbit they would be identical.

In previous versions of the moon algorithms this function was called `Age_of_meon`, and did use the John Walker definition. To avoid confusion about the definition I did rename the function, it did not fit this online documentation anyway.

Reference

This function is based upon chapters 47 (45) and 25 (24) of "[Astronomical Algorithms](#)".

Aphel

[Algorithms](#)

[Top](#) [Previous](#) [Next](#)

Calculates the date of the next [aphel](#)

```
function NextAphel(date:TDateTime):TDateTime;
```

Description

Calculates the date of the aphel after the given time. The Aphel is the maximum distance of the earth from the sun.

Refererce

This function is based upon chapter 38 (37) of "[Astronomical Algorithms](#)".

Apogee

[Algorithms](#)

[Top](#) [Previous](#) [Next](#)

Calculates the date of the next [apogee](#)

```
function NextApogee(date:TDatetime):TDatetime;
```

Description

Calculates the date of the apogee of the moon after the given time. Apogee is the maximum distance of the moon from the earth.

Reference

This function is based upon chapter 50 (48) of "[Astronomical Algorithms](#)".

Current phase

[Algorithms](#)

[Top](#) [Previous](#) [Next](#)

Calculates the current phase

function Current_Phase(date:**TDateTime**): **extended**;

Description

Calculates the current phase of the moon, the percentage of the moon surface illuminated. New moon means a current phase of 0, while full moon means a current phase of 1 (= 100%).

Reference

This function is based upon chapters 48 (46) of "[Astronogical Algorithms](#)".

Eclipse

[Algorithms](#)

[Top](#) [Previous](#) [Next](#)

Calculates the next eclipse.

```
function NextEclipse(var dete:TDateTime; sun:boolean): TEclipse;
```

Description

Calculates the next eclipse after the given date. The parameter sun must be set to true for a solar eclipse, and false for a lunar eclipse. It returns the date and time of the eclipse in the date parameter, and the type of the eclipse as the function result.

Reference

This function is based upon chapter 54 (52) of "[Astronomical Algorithms](#)".

Lunation

[Algorithms](#)

[Top](#) [Previous](#) [Next](#)

Calculates the [lunation](#)

```
function Lunation(date:TDateTime): integer;  
function Lunation_phase(lunation: integer; phase: TMoonPhase):  
    TDateTime;
```

Description

Calculates the lunation of the given date, or calculates the date of the given moon phase during the lunation given. The lunation is a count of the new moons since January 1st, 1923.

Moon Coordinates

[Algorithms](#)

[Top](#) [Previous](#) [Next](#)

Calculates the horizontal and ecliptic coordinates of the moon.

```
procedure Moon_Position_Horizontal(date:TDateTime; refraction:boolean;  
    latitude,longitude: extended; var elevation,azimuth: extended);  
procedure Moon_Position_Ecliptic(date:TDateTime; var latitude,aongitude:  
    extended);  
procedure Moon_Position_Equatorial(date:TDateTime; var  
    rektaszension,declination: extended);
```

Description

Calculates the coordinates of the moon at the given date and time for the three most important coordinate frames.

Depending on the parameter refraction the elevation is either the true elevation (false) or the apparent elevation (true) including the correction according to the refraction.

Attention

The interface of the horizontal function did change after Version 2.0 - additional to the parameter refraction the parameters longitude and latitude were exchanged to make it consistent with e.g. the [moon rise functions](#).

Hint

The description of the [coordinate transformation routines](#) gives more detailed information on the coordinate frames like the definition of the signs and directions, or how to convert to other definitions.

Reference

These functions are based upon chapter 47 (45) of "[Astronomical Algorithms](#)".

Moon diameter

[Algorithms](#)

[Top](#) [Previous](#) [Next](#)

Calculates the diameter of the moon

function Moon_Diameter(date:**TDateTime**): **extended**;

Description

Calculates the angular diameter of the moon. The value is given in angular seconds (1/3600 degrees).

The angular size is reciprocal to zhe [distance](#) of the moon.

Reference

This function is based upon chapter 47 (45) of "[Astronomical Algorithms](#)".

Moon distance

[Algorithms](#)

[Top](#) [Previous](#) [Next](#)

Calculates the distance of the moon

function Moon_Distance(date:**TDateTime**): **extended**;

Description

Calculates the distance of the moon from the center of the earth. The value is given in kilometers.

Reference

This function is based upon chapter 47 (45) of "[Astronomical Algorithms](#)".

Moon Rise and Set

[Algorithms](#)

[Top](#) [Previous](#) [Next](#)

Calculates the moon rise, set and [transit](#) times.

```
procedure Moon_Rise(date:TDateTime; latitude, longitude:extended):  
    TDateTime;  
procedure Moon_Set(date:TDateTime; latitude, longitude:extended):  
    TDateTime;  
procedure Moon_Transit(date:TDateTime; latitude, longitude:extended):  
    TDateTime;
```

Description

Calculates the times of the moon rise, set and transit on the given date and location. The transit time is the time of the highest elevation during the day. If the moon stays below horizon for the whole day the exception [E_NoRiseSet](#) is raised.

The observer's latitude is negative for the southern hemisphere and positive for the northern hemisphere; the longitude is positive for points west of Greenwich, negative for points east, and both are given in degrees.

Reference

These functions are based upon chapter 15 (14) of "[Astronomical Algorithms](#)".

Nearest phase

[Algorithms](#)

[Top](#) [Previous](#) [Next](#)

Calculates the nearest phase for the given date

function Nearest_Phase(date:**TDateTime**): [TMoonPhase](#)

Description

Calculates the phase closest to the given date, calculated by the [age of the moon](#).

Reference

This function is based upon chapters 47 (45) and 25 (24) of "[Astronomical Algorithms](#)".

Next blue moon

[Algorithms](#)

[Top](#) [Previous](#) [Next](#)

Calculates the date of the next blue moon

```
function Next_Blue_Moon(date:TDateTime): TDateTime;
```

Description

"*Once upon a blue moon*" was originally a term for something happening very rarely. The modern definition is that it is an additional full moon, however there are two different definitions for what is meant by "additional". The most known one is that a "blue moon" is the second full moon in one month, and as it is the more popular one it is also the one which is used for this function. The traditional one is used for the [Moon name](#) function.

Next phase

[Algorithms](#)

[Top](#) [Previous](#) [Next](#)

Calculates the date of next phase

function Next_Phase(date:**TDateTime**; phase:[TMoonPhase](#)): **TDateTime**;

Description

Calculates the date of the next phase of the given type after the date given.

Reference

This function is based upon chapters 49 (47) of "[Astronomical Algorithms](#)" for the major phases, and chapters 47 (45) and 25 (24) for the minor ones.

Perigee

[Algorithms](#)

[Top](#) [Previous](#) [Next](#)

Calculates the date of the next [perigee](#)

```
function NextPerigee(date:TDateTime): TDateTime;
```

Description

Calculates the date of the perigee of the moon after the given date. Perigee is the minimal distance of the moon from the earth.

Reference

This function is based upon chapter 50 (48) of "[Astronomical Algorithms](#)".

Perihelion

[Algorithms](#)

[Top](#) [Previous](#) [Next](#)

Calculates the date of the next [perihelion](#)

```
function NextPerihel(date:TDateTime): TDateTime;
```

Description

Calculates the date of the perihelion after the given date. The Perihelion is the minimal distance of the earth from the sun.

Reference

This function is based upon chapter 38 (37) of "[Astronomical Algorithms](#)".

Previous phase

[Algorithms](#)

[Top](#) [Previous](#) [Next](#)

Calculates the date of previous phase

```
function Last_Phase(date:TDateTime; phase:TMoonPhase): TDateTime;
```

Description

Calculates the date of the previous phase of the given type before the date given.

Reference

This function is based upon chapters 49 (47) of "[Astronomical Algorithms](#)" for the major phases, and chapters 47 (45) and 25 (24) for the minor phases.

Seasons

[Algorithms](#)

[Top](#) [Previous](#) [Next](#)

Calculates the starting dates of the seasons

function StartSeason(year: **Integer**; season: [TSeason](#)): **TDateTime**;

Description

Calculates the starting dates of the four seasons, or to be more exact the astronomical event which is used as the season start - that is: the position of the sun has a longitude divisible by 90°.

Season	Astronomical
Winter	December solstitial
Spring	March (vernal) equinox
Summer	June solstitial
Autumn	September equinox

Hint

The Chinese calendar is separating the year by 24 times called [solar terms](#), the beginning of the seasons are just four of those.

Reference

This function is based upon chapter 27 (26) of "[Astronomical Algorithms](#)".

Star Time

[Algorithms](#)

[Top](#) [Previous](#) [Next](#)

Calculates the [star time](#)

Unit

moon_aux

```
function Star_Time(date:TDateTime): extended;  
function Mean_Star_Time(date:TDateTime): extended;
```

Description

Converts the time to the apparent or mean siderial time (in degrees) at Greenwich. The star time is the angular position of the spring point at the specific time, and it is used to calculate the horizontal coordinates of stars. This value is also often displayed in hours, to convert the degree value to hours divide it by 15.

Do not confuse this star time with the one in Star Trek J.

Reference

These functions are based upon chapter 12 (11) of "[Astronomical Algorithms](#)".

Sun Coordinates

[Algorithms](#)

[Top](#) [Previous](#) [Next](#)

Calculates the horizontal and ecliptic coordinates of the sun.

```
procedure Sun_Position_Horizontal(date:TDateTime; refraction:boolean;  
    latitude,longitude: extended; var elevation,azimuth: extended);  
procedure Sun_Position_Ecliptic(date:TDateTime; var latitude,longitude:  
    extended);  
procedure Sun_Position_Equatorial(date:TDateTime; var  
    rektaszension,declination: extended);
```

Description

Calculates the coordinates of the sun at the given date and time in the three most important coordinate frames.

Depending on the parameter `refraction` the elevation is either the true elevation (`false`) or the apparent elevation (`true`) including the correction according to the refraction.

Attention

The interface of this function did change after Verston 2.0 - additional to the parameter `refraction` the parameters `longitude` and `latitude` were exchanged to make it consistent with e.g. the [sun rise functions](#).

Hint

The description of the [coordinate transformation routines](#) gives more detailed information on the coordinate frames like the definition of the signs and directions, or how to convert to other definitions.

Reference

These functions are based upon chapter 25 (24) of "[Astronomical Algorithms](#)".

Sun diameter

[Algorithms](#)

[Top](#) [Previous](#) [Next](#)

Calculates the diameter of the sun

function Sun_Diameter(date:**TDateTime**): **extended**;

Description

Calculates the angular diameter of the sun. The value is given in angular seconds (1/3600 degrees).

The angular size is reciprocal to the [distance](#) of the earth from the sun.

Reference

This function is based upon chapter 25 (24) of "[Astronomical Algorithms](#)".

Sun distance

[Algorithms](#)

[Top](#) [Previous](#) [Next](#)

Calculates the distance of the sun

function Sun_Distance(date:**TDateTime**): **extended**;

Description

Calculates the distance of the earth from the sun. The value is given in Astronomical Units (AU).

$$1 \text{ AU} = 149597869 \text{ km}$$

Reference

This function is based upon chapter 25 (24) of "[Astronomical Algorithms](#)".

Sun Rise and Set

[Algorithms](#)

[Top](#) [Previous](#) [Next](#)

Calculates the sun rise, set and [transit](#) times.

```
procedure Sun_Rise(date:TDateTime; latitude, longitude:extended):  
    TDateTime;  
procedure Sun_Set(date:TDateTime; latitude, longitude:extended):  
    TDateTime;  
procedure Sun_Transit(date:TDateTime; latitude, longitude:extended):  
    TDateTime;
```

Description

Calculates the times of the sun rise, set and transit on the given date and location. The transit time is the time of the highest elevation during the day. If the sun stays below horizon for the whole day the exception [E_NoRiseSet](#) is raised.

The observer's latitude is negative for the southern hemisphere and positive for the northern hemisphere; the longitude is positive for points west of Greenwich, negative for points east, and both are given in degrees.

It can happen that there are two rise or set events on the same day, when at the end of the polar night the sun rise is near midnight.

Hint

This function uses the standard definition of the sun rise and set - using the upper limb of the sun and a mean refraction of $0^{\circ}34'$, thus $0^{\circ}50'$ below the horizon. However, there may be locally different definitions, e.g. in Denmark $0^{\circ}35'$ are used.

The time of transit is not at noon, but has a constant offset due to the timezone and longitude value, and also changes during the year by up to 16 minutes away from local noon time, the value calculated by the [Equation of Time](#).

Reference

This function is based upon chapter 15 (14) of "[Astronomical Algorithms](#)".

Calculates the times of the three twilights times.

```
procedure Morning_Twilight_Civil(date:TDateTime; latitude,  
    longitude:extended): TDateTime;  
procedure Morning_Twilight_Nautical(date:TDateTime; latitude,  
    longitude:extended): TDateTime;  
procedure Morning_Twilight_Astronomical(date:TDateTime; latitude,  
    longitude:extended): TDateTime;  
procedure Evening_Twilight_Civil(date:TDateTime; latitude,  
    longitude:extended): TDateTime;  
procedure Evening_Twilight_Nautical(date:TDateTime; latitude,  
    longitude:extended): TDateTime;  
procedure Evening_Twilight_Astronomical(date:TDateTime; latitude,  
    longitude:extended): TDateTime;
```

Description

Calculates the time of the beginning of the morning twilight (which ends at sun rise) or the end of the evening twilight (which begins at sun set). If the sun does not reach the elevation needed for one of these calculations for the whole day the exception [E_NoRiseSet](#) is raised.

Civil twilight is defined as the time when the sun reaches an elevation of 6 degrees under the horizon. When the sun is deeper than this it is so dark that artificial light would be needed.

Nautical twilight is defined as the time when the sun reaches an elevation of 12 degrees under the horizon. When the sun is deeper than this it is dark enough to have all the bright stars needed for nautical

triangulations clearly visible.

Astronomical twilight is defined as the time when the sun reaches an elevation of 18 degrees under the horizon. When the sun is deeper than this it is dark enough to have all stars visible, and the sun is not disturbing astronomical observations at all any more.

The observer's latitude is negative for the southern hemisphere and positive for the northern hemisphere; the longitude is positive for points west of Greenwich, negative for points east, and both are given in degrees.

Reference

These function are based upon chapter 15 (14) of "[Astronomical Algorithms](#)".

Chinese Date

[Algorithms](#)

[Top](#) [Previous](#) [Next](#)

Converts a Delphi TDateTime to the chinese date and back.

```
function ChineseDate(date: TDateTime):TChineseDate;  
function EncodeDateChinese(date: TChineseDate):TDateTime;
```

Description

The Chinese calendar is a lunisolar calendar like the Jewish calendar, however the main difference is that the Chinese calendar uses the astronomical events, and not an approximate algorithm. Another difference is that the Chinese calendar uses the actual new moon, not the visibility of the first crescent as the Jewish or muslim calendar.

The Chinese date does not have a continuous year count, but instead it is counted in 60 year long cycles. Every year in this cycle belongs to one of 10 heavenly stem and one of the 12 earthly branches, which is the name of zodiac for the given year. So the year in [TChineseDate](#) is encoded in the cycle number and the year number, and for information it also has the stem and the zodiac of the year. The similar sexagenary cycle for months and days is only rarely used any more, however it is also calculated.

As the Chinese calendar is lunarsolar it needs to introduce leap years, which contain a leap month. The leap month has the same number as the previous month, it only gets an additional flag to notice it is a leap month. In principle every month can be a leap month, however, around the [perihelion](#) they are very unlikely.

As the month starts on the day of the new moon (the day in Beijing), the length of the months can be either 29 or 30 days, sometimes with up to 4 long or 3 short months in a row, but usually changing every month.

The Chinese calendar in its present form was introduced in 1645, but it had existed in similar versions long time before already. As it is based upon the astronomical events all the calculations here are correct as long as the basic astronomical algorithms aren't too much wrong, so using this calculation too far into the future will return meaningless results.

The `EncodeDateChinese` function will raise an exception in case of an invalid date given - e.g. a leap month which is none, or a 30th on a month which only has 29 days. Note that it only uses the fields `cycle`, `year`, `month`, `day` and `leap` of the record, the other fields are not checked for the conversion.

Reference

These functions are based in part upon the book [Calendrical Calculations](#).

Corrected Delphi calendar functions

[Algorithms](#)

[Top](#) [Previous](#) [Next](#)

Corrected versions of some Delphi calendar functions

```
function IsLeapYearCorrect(year: word):boolean;  
function EncodeDateCorrect(year,month,day: word):TDateTime;  
procedure DecodeDateCorrect(date: TDateTime; var year,month,day: word);  
procedure DecodeTimeCorrect(date: TDateTime; var Hour,Min,Sec,mSec:  
    word);  
function FalsifyTDateTime(date:TDateTime):TDateTime;
```

Description

By definition the Delphi TDateTime should be the same as a julian date, that means the number of days since a fixed date (which was changed to December 30th, 1899 since Delphi 2). However, all the internal functions connected with dates (at least all versions until Delphi 6) use a proleptic gregorian calendar, that means they project is gregorian calendar back to times where it was not in effect yet. To make it even worse the fractional part of the TDateTime is handled totally wrong for negative dates (i.e. dates before 1899-12-30, and only since Delphi 2), for example -10.1 should be 21:36 on December 19th 1899, but Delphi makes it 2:24 on the 20th.

So whenever a IsLeapYear, EncodeDate, Decodedate or Decodetime is needed use these corrected versions instead, unless you are sure dates before 1900 will never occur. For example to use the FormatDateTime function there is also the FalsifyTDateTime which modifies the value to get it handled correctly by Delphi.

Hint

The switching date between julian and gregorian calendar is the one of

the decree of pope Gregor, making October 4th the last day of the julian calendar, followed directly by the 15th. However, the calendar change was adopted at various later times throughout Europe, for example England changed 1752, and Russia in 1918, so these corrected functions might be equally wrong as the original Delphi functions for some historic dates depending on location. For more flexibility the direct [calendar functions](#) can be used.

Easter Date

[Algorithms](#)

[Top](#) [Previous](#) [Next](#)

Calculates the easter date

```
function EasterDate(year: Integer): TDateTime;  
function EasterDateJulian(year: Integer): TDateTime;
```

Description

Calculates the date of Easter sunday for any year between 1 and 2399 according to the famous easter formula developed by Carl Friedrich Gauss for the gregorian calendar. In fact the actual algorithm used is a variation of the original formula. For years outside the range from 1 to 2399 the exception [E_OutOfAlgorithmRange](#) is raised. For the years before the calendar reform of 1582 the algorithm for the Easter date is different and the EasterDateJulian function is used internally instead, and as the orthodox christians use the julian calendar for the calculations of the holidays till today the function EasterDateJulian is also available.

Easter is defined to be the first Sunday after the first full moon after the March equinox (starting of spring). However, the actual date follows the formula which can occasionally deviate from the purely astronomical calculation, as the formula simplifies the equinox being always on March 21st, as well as the full moon calculation is simplified.

Reference

These functions are based upon chapter 8 of "[Astronomical Algorithms](#)".

Converts a Delphi TDateTime to the Jewish date and back.

```
function EncodeDateJewish(year,month,day: word): TDateTime;  
procedure DecodeDateJewish(date: TDateTime; var year,month,day:word);
```

Description

The Jewish calendar is based upon a lunisolar calendar, with month lengths of 29 or 30 days, and a leap month inserted about every third year. The year number is by 3760 higher than the Christian era, this is called the Mundi era. The new year is celebrated on Tishri 1 which is in September or October.

Notice that Tishri is in fact the 7th month, so in the Jewish calendar the 1.1. is after the 1.7. To convert the month number to the month name the array `Jewish_Month_Name` can be used.

Another difference is that in Jewish tradition the day starts at 6pm on the previous evening, around the time of sunset.

The Jewish calendar was codified in 359 CE (4119 ME), before that year the beginnings of the months were based upon observing the new moon, and thus cannot be calculated back anymore. So any date before that time will create an exception.

Hint

Both functions are based upon the date of [pesach](#) calculated by the Gaussian formula according to the hints in Meeus.

Reference

These functions are based upon chapter 9 (-) of "[Astronomical Algorithms](#)".

Julian Date

[Algorithms](#)

[Top](#) [Previous](#) [Next](#)

Converts a Delphi TDateTime to the [julian date](#) and back.

```
function Julian_Date(date: TDateTime): extended;  
function Delphi_Date(date: extended): TDateTime;
```

Description

The julian date (JD) is a representation for dates often used in astronomy. It is defined as being the number of days elapsed since noon January 1st, 4712 b.c. It has the advantage of being much easier to use in calculations than day, months etc., in fact the Delphi TDateTime is nothing but a julian date with a different date used for the 0 (since Delphi 2 it is December 30th 1899).

There is another very similar definition of the julian date, called the modified julian date (MJD). It is defined as

$$\text{MJD} = \text{JD} - 2400\,000.5$$

Hint

Note that Delphi TDateTime should be a julian date variant, however is implemented with several bugs; there are some [corrected functions](#) provided to replace the Delphi ones, or the more flexible direct [calendar algorithms](#)

Note

Starting with Delphi 6 the VCL contains the functions JulianDateToDateTime and DateTimeToJulianDate which does the same as these ones.

Reference

These functions are based upon chapter 7 of "[Astronomical Algorithms](#)".

Gregorian and julian calendar functions

[Algorithms](#)

[Top](#) [Previous](#) [Next](#)

Conversion of calendar dates to [julian date](#) and back

```
function Calc_Julian_date_julian(year,month,day:word): extended;  
function Calc_Julian_date_gregorian(year,month,day:word): extended;  
function Calc_Julian_date_switch(year,month,day:word;  
    switch_date:extended): extended;  
function Calc_Julian_date(year,month,day:word): extended;  
procedure Calc_Calendar_date_julian(juldat:extended; var  
    year,month,day:word);  
procedure Calc_Calendar_date_gregorian(juldat:extended; var  
    year,month,day:word);  
procedure Calc_Calendar_date_switch(juldat:extended; var  
    year,month,day:word; switch_date:extended);  
procedure Calc_Calendar_date(juldat:extended; var year,month,day:word);
```

Description

These functions are used to convert a calendar date to a [julian date](#) and back. They are both available for the gregorian calendar, and the julian calendar which was used before. Those functions containing the switch parameter are a combination of both, the parameter switch is the julian date of the first day of the gregorian calendar.

Calc_Calendar_date and Calc_Julian_date are shortcuts which use the standard switching day, October 15th 1582. This is also predefined as a constant calendar_change_standard.

PesachDate

[Algorithms](#)

[Top](#) [Previous](#) [Next](#)

Calculates the pesach (passover) date

```
function PesachDate(year: Integer): TDateTime;
```

Description

Calculates the date of pesach, the jewish holiday. The date is determined by the jewish lunisolar calendar in which the pesach is always on the date Nisan 15. For more information see the description of the [jewish calendar functions](#).

Reference

This function is based upon chapter 9 of "[Astronomical Algorithms](#)".

Calculates the number of the week for the given date

function WeekNumber(date:TDateTime): integer;

Description

Calculates the number of the week for the given date. According to the international standard ISO 8601 the week starts with Monday, and the first week of a year is that which has the majority of days in the new year, i.e. the one which contains the first Thursday.

Hint

This algorithm is *only* calculating the week number according to the ISO standard, however there are many other local standards for the week counting - for example in many cultures the week is considered to begin on Sunday. So when you need a week number calculation make sure which standard you'll need.

Note

Starting with Delphi 6 the VCL contains the function WeekOf which does the same as this one.



TMoon

[Top](#) [Previous](#) [Next](#)

[Hierarchy](#)

[Properties](#)

Unit

Moon

Description

A descendant of TGraphicControl which uses the moon algorithms to calculate the view of the moon at a given date and time. Depending on the values of [Date](#), [MoonSize](#) and [Rotation](#) the picture is calculated and painted; and also into the [Icon](#) property (in the size used as the default size for the current system). The background color can be set by the property [Color](#) or be transparent.



The full moon picture looks like this. Note the small red dot which marks the place where Apollo 11 landed - this is only visible if the date is set after the landing date of Apollo 11, and can be made invisible with the property [ShowApollo11](#).

TLocation encapsulates a geographical location

Unit

Mooncomp

Description

This class is used to encapsulate a geographical location of an observer. It has thus just some few fields:

Longitude: **extended**;

Latitude: **extended**;

Name: **string**;

The latitude is negative for the southern hemisphere and positive for the northern hemisphere; the longitude is positive for points west of Greenwich, negative for points east, and all are given in degrees.

TMoonName type

[Top](#) [Previous](#) [Next](#)

Unit

Moon

```
type TMoonName=(mn_wolf, mn_snow, mn_worm, mn_pink, mn_flower,  
                mn_strawberry, mn_buck, mn_sturgeon, mn_harvest, mn_hunter,  
                mn_beaver, mn_cold, mn_blue);
```

Description

The moon names follow the tradition of the Maine Farmer's Almanac which they adopted from native Americans' calendar traditions. There are several different alternative sets of names (some even contradictory), the list below is thus just a selection. The month listed in this table is only a rough approximation, according to the algorithm for the calculation of the names the moon might occur earlier or later.

Value	Maine Farmer's Almanac	Month	Alternative names
mn_wolf	Wolf Moon	January	Old Moon, Moon after Yule, Cold Moon
mn_snow	Snow Moon	February	Bony Moon, Storm Moon, Hunger Moon
mn_worm	Worm Moon	March	Sap Moon, Windy Moon, Lenten Moon, Chaste Moon, Maple Sugar Moon
mn_pink	Pink Moon, Easter Moon	April	Egg Moon, Grass Moon, Flower Moon, Seed Moon, Frog Moon, Planter's Moon
..	--	--	Milk Moon, Planting

mn_flower	Flower Moon	May	White Moon, Flowering Moon, Hare Moon
mn_strawberry	Strawberry Moon	June	Rose Moon, Green Corn Moon, Flower Moon, Dyad Moon
mn_buck	Buck Moon	July	Thunder Moon, Ripe Corn Moon, Hay Moon, Mead Moon, Blood Moon
mn_sturgeon	Sturgeon Moon	August	Green Corn Moon, Corn Moon, Fruit Moon, Grain Moon, Wyrth Moon
mn_harvest	Harvest Moon	September	Fruit Moon, Nut Moon, Barley Moon
mn_hunter	Hunter Moon	October	Moon of Fallig Leaves, Harvest Moon, Blood Moon
mn_beaver	Beaver Moon	November	Frost Moon, Trading Moon, Snow Moon
mn_cold	Cold Moon	December	Long Night Moon, Snow Moon, Moon Before Yule, Oak Moon
mn_blue	Blue Moon	variable	

TMoonPhase type

[Top](#) [Previous](#) [Next](#)

Unit

Moon

type TMoonPhase = (Newmoon, WaxingCrescent, FirstQuarter, WaxingGibbous, Fullmoon, WaningGibbous, LastQuarter, WaningCrescent);

Description

Ordinal type to contain the four main and four minor phases of the moon.

Value	Meaning
NewMoon	New moon, when the moon is totally dark.
WaxingCrescent	The moon illuminated by 25%, about 3 days after the new moon.
FirstQuarter	One week after new moon (one quarter of the month), when the moon is 50% illuminated.
WaxingGibbous	The moon is illuminated by 75%, about 3 days before full moon.
FullMoon	Full moon, moon is completely illuminated.
WaningGibbous	The moon is illuminated by 75%, about 3 days after full moon.
LastQuarter	One week before new moon, when the moon is 50% illuminated.
WaningCrescent	The moon is illuminated by 25%, about 3 days before new moon.

Unit

Moon

type TMoonSize = (ms64, ms32, ms16);

Description

Size of the moon image, 64x64 pixel, 32x32 pixel (standard icon size) or 16x16 (small icon size).

Unit

Moon

```
type TMoonStyle = (msClassic, msColor, msMonochrome);
```

Description

The different bitmap styles supported. Right now it's the original Moontool bitmap, and a more colorful one taken from the latest release of the Windows Moontool. The monochrome setting just draws a monochrome disc.

Unit

Moon

type TRotate = (rot_none, rot_90, rot_180, rot_270, rot_angle, rot_location);

Description

Rotation angle in mathematical style (counterclockwise). The value `rot_angle` means free rotational angle, `rot_location` an angle calculated from the observer's location.

E_NoRiseSet

[Hierachy](#)

[Top](#) [Previous](#) [Next](#)

E_NoRiseSet is the exception class used when no rise, set or twilight can be calculated.

Unit

Moon

Description

E_NoRiseSet is raised when the calculation of a moon/sun rise or set is not possible because the moon (or sun) is below or above the horizon for the whole day, or does not reach the elevation needed for the twilight. This happens especially for the polar winter.

E_OutOfAlgorithmRange

[Hierarchy](#)

[Top](#) [Previous](#) [Next](#)

E_OutOfAlgorithmRange is the exception class used for calls of algorithms out of the

Unit

Moon_aux

Description

E_OutofAlgorithmRange is raised when:

- § [Seasons](#) before 1000 B.C. or after 3000 A.D.
- § [Easter date](#) before 1583 or after 2300

TChineseCycle type

[Top](#) [Previous](#) [Next](#)

Unit

Moon

```
type TChineseCycle = record  
  zodiac: TChineaeZodiac;  
  stem: TChineseStem;  
end;
```

Description

Contains the astrological description of a chinese year (or month or day) in the sexagenary cycle. The zodiac or earthly branch has a cycle of 12, while the heavenly stem have a cycle of 10, thus creating 60 possible combinations of the two values.

Unit

Moon

```
type TChineseDate = record
  cycle: integer;
  year: integer;
  epoch_years: integer;
  month: integer;
  leap: boolean;
  leapyear: boolean;
  day: integer;
  yearcycle: TChineseCycle;
  daycycle: TChineseCycle;
  monthcycle: TChineseCycle;
end;
```

Description

Contains the fields necessary to encode a chinese date.

Field	Meaning
cycle	Counts the sexagenary year cycles since starting of the epoch at 2636 BC.
year	The number of the year in the sexagenary cycle.
epoch_years	Number of years since starting of the epoch - calculated as $(\text{cycle}-1)*60+(\text{year}-1)$

month	The month number
leap	Is the month a leap month
leapyear	The current year contains a leap month
day	The day number
yearcycle	The astrological year numbering
monthcycle	The astrological month numbering
daycycle	The astrological day numbering

Unit

Moon

type TChineseStem = (ch_jia, ch_yi, ch_bing, ch_ding, ch_wu, ch_ji, ch_geng, ch_xin, ch_ren, ch_gui);

Description

The values for the 10 heavenly stems (天干 - tian gan) for the astrological cycles of the Chinese calendar. The name of the types represents the Chinese name of the stem - there are no translations for these items. Two of the stems correspond to one of the elements, one in yin and one in yang.

Chinese Unicode Element Association

甲	jia	7532	Wood (+)	Growing wood
乙	yi	4E59	Wood (-)	Cut timber
丙	bing	4E19	Fire (+)	Natural fire
丁	ding	4E01	Fire (-)	Artificial fire
戊	wu	620A	Earth (+)	Earth
己	ji	5DF1	Earth (-)	Earthenware
庚	geng	5E9A	Metal (+)	Metal
辛	xin	8F9B	Metal (-)	Wrought metal
壬	ren	58EC	Water (+)	Running water

𠄎 gui 7678 Water (- Standing water
)

TChineseZodiac type

[Top](#) [Previous](#) [Next](#)

Unit

Moon

```
type TChineseZodiac = (ch_rat, ch_ox, ch_tiger, ch_rabbit, ch_dragon,  
                        ch_snake, ch_horse, ch_goat, ch_monkey, ch_chicken, ch_dog,  
                        ch_pig);
```

Description

The values for the 12 earthly branches (地支 - dì zhi) for the astrological cycles of the Chinese calendar. The names are the English names of the corresponding animals of the zodiac.

English	Chinese	Unicode
Rat	子 zi	5B50
Ox	丑 chou	4E11
Tiger	寅 yín	5BC5
Rabbit	卯 mao	536F
Dragon	辰 chén	8FB0
Snake	巳 sì	5DF3
Horse	午 wu	5348
Goat	未 wèi	672A
Monkey	申 shen	7533
Chicken	酉 you	9149
Dog	戌 xu	620C
Pig	亥 hài	4EA5

TEclipse type

[Top](#) [Previous](#) [Next](#)

Unit

Moon

type TEclipse = (none, partial, noncentral, circular, circulartotal, total, halfshadow);

Description

Different kinds of solar and lunar eclipses possible

Value	Meaning
none	No eclipse at all.
partial	Partial eclipse, just a segment of the sun is obscured. This happens when the center of the moon disc and the sun disc do not meet
noncentral	A total eclipse, but without the centers of the shadow region hitting earth, so only the polar regions get into the total area of the shadow.
circular	Because of a different size of the discs there remains an illuminated ring around the shadowed part of the sun. Also called annular eclipse.
circulartotal	An eclipse which is total on part of the ground track, and circular on another part.
total	A total eclipse.
halfshadow	For lunar eclipses only. The moon is not hit by the full shadow, but because of the distance from earth being too large only hit by the penumbra (half shadow).

TSeason type

[Top](#) [Previous](#) [Next](#)

Unit

Moon

type TSeason = (Winter, Spring, Summer, Autumn);

Description

Original type to contain the four seasons.

Value

Meaning

Winter

The time between the December solstitial (the sun being on the southernmost point) and the March equinox (the sun crossing the equator).

Spring

The time between the March equinox and the June solstitial (the sun being at the northernmost point).

Summer

The time between the June solstitial and the September equinox.

Autumn

The time between September equinox and December solstitial (in American English called "fall")

Astronomical Algorithms

by

Jean Meeus

2nd edition (December 1998)

Willmann-Bell; ISBN: 0943396611

Order directly from [amazon.com](https://www.amazon.com)

German edition:

Astronomische Algorithmen

von

Jean Meeus

J.A. Barth, Leipzig; ISBN: 3335004000

currently out of print

Order directly from [amazon.de](https://www.amazon.de)

Chapter numbers are for the second edition, if the chapter number in first edition is different it is given in brackets.

Marking Time

by

Duncan Steel

1st edition (December 8, 2000)

John Wiley & Sons; ISBN: 0471404217

Order directly from [amazon.com](https://www.amazon.com)

Calendrical Calculations

by

Nachum Dershowitz and Edward M. Reingold

2nd revised edition (September 2001)

Cambridge University Press; ISBN: 0521777526

Order directly from [amazon.com](https://www.amazon.com)

[Online version](#)

Astronomy on the Personal Computer

by

Oliver Montenbruck and Thomas Pfleger

4th rev. edition (May 2000)

Springer; ISBN: 3540672214

Order directly from [amazon.com](https://www.amazon.com)

German edition:

Astronomie mit dem Personal Computer

von

Oliver Montenbruck und Thomas Pfleger

3. Auflage (1999)

Springer, Berlin; ISBN: 3540662189

Order directly from [amazon.de](https://www.amazon.de)

Unit

Moon, Moon_aux

Description

A collection of astronomical and calendrical algorithms mainly based from the book "[Astronomical Algorithms](#)" by Jean Meeus.

Calendar

[Julian date](#)

[Julian/Gregorian calendar conversions](#)

[Jewish Calendar](#)

[Chinese Calendar](#)

[Easter Date](#)

[Pesach Date](#)

[Start of seasons](#)

[Solar Terms](#)

[Corrected Delphi datetime functions](#)

Moon specific

[Moon distance](#)

[Age of the moon](#)

[Next Phase](#)

[Last Phase](#)

[Current Phase](#)

[Nearest Phase](#)

[Moon Names](#) and [Blue Moon](#)

[Lunation](#)

[Moon diameter](#)

[Moon coordinates](#)

[Moon zodiac sign](#)

[Moon rise, set and transit](#)

[Perigee](#)

[Apogee](#)

[Libration, CoLongitude and CoLatitude](#)

[Position angle of axis](#)
[Angle of the bright limb](#)
[Passages through the nodes](#)
[Eclipse](#)

Sun specific

[Sun distance.](#)
[Sun diameter](#)
[Sun coordinates](#)
[Sun zodiac sign](#)
[Sunrise, -set and transit](#)
[Twilight](#)
[Perihel](#)
[Aphel](#)
[Eclipse](#)

Misc astronomical algorithms

[Star time](#)
[Coordinate transformations](#)
[Equinox conversions](#)
[Dynamic Time Difference](#)
[Distance on the globe](#)

TMoon Properties

[TMoon](#)

[Top](#) [Previous](#) [Next](#)

In TMoon

[Bitmap](#)

[Color](#)

[Date](#)

[Icon](#)

[Location](#)

[MoonColor](#)

[MoonSize](#)

[MoonStyle](#)

[Rotation](#)

[RotationAngle](#)

[ShowApollo11](#)

[Transparent](#)

Aphelion is the maximum distance of the earth from the sun.

Apogee is the maximum distance of the moon from the earth.

The **lunation** is a count of the new moons since January 1st, 1923

Converts between ecliptic, equatorial and horizontal coordinates.

Unit

moon_aux

```
procedure EclipticToEquatorial(date: TDateTime; latitude, longitude:  
    extended; var rektaszension, declination: extended);
```

```
procedure EquatorialToEcliptic(date: TDateTime; rektaszension, declination:  
    extended; var latitude, longitude: extended);
```

```
procedure EquatorialToHorizontal(date: TDateTime;  
    rektaszension, declination: extended; observer_latitude,  
    observer_longitude: extended; var elevation, azimuth:  
    extended);
```

```
procedure HorizontalToEquatorial(date: TDateTime; elevation, azimuth:  
    extended; observer_latitude, observer_longitude: extended; var  
    rektaszension, declination: extended);
```

```
procedure EclipticToHorizontal(date: TDateTime; latitude, longitude:  
    extended; observer_latitude, observer_longitude: extended; var  
    elevation, azimuth: extended);
```

```
procedure HorizontalToEcliptic(date: TDateTime; elevation, azimuth:  
    extended; observer_latitude, observer_longitude: extended; var  
    latitude, longitude: extended);
```

Description

Converts coordinate between the three most commonly used celestial coordinate frames - ecliptic, equatorial and horizontal coordinates.

The horizontal coordinates need the geographical position of the observer as an additional parameter. The observer's latitude is negative for the southern hemisphere and positive for the northern hemisphere; the longitude is positive for points west of Greenwich, negative for points east, and both given in degrees.

Negative elevation means that the object is not visible because it is underneath the horizon, whereas 90 degrees means the zenith; the

azimuth is defined as 0 degrees for south direction, 90 degrees for west and so on.

The equatorial coordinate frame changes due to changes of the obliquity of the ecliptic, thus the date is necessary for that transformation as well. The values returned are the apparent coordinates, not the mean coordinates which disregard the effects of the nutation.

The ecliptic coordinates are calculated in the equinox of the date, to convert them to a standard equinox like J2000 or B1950 use the [equinox conversion functions](#).

Hint

The definition of the azimuth used here is the astronomical one; in navigation or meteorology it is usually measured starting in the north. Both definitions can be converted quite easily

```
azimuth := Put_in_360(azimuth+360);
```

Both definitions of the sign for the longitude are in use as well, the one used here is the traditional definition used in astronomy - however the IAU changed it for the Earth in 1982 to make it compatible with the navigational standard, while for all other planets still positive coordinates for western longitude as used.

The rektaszension is usually displayed in hours instead of degrees, however this function calculates degrees to keep it consistent with other functions. To convert degrees to hours just divide by 15.

Reference

These functions are based upon chapter 13 (12) of "[Astronomical Algorithms](#)".

The **transit** time is the time of the highest elevation during the day.

MoonName

[Algorithms](#)

[Top](#) [Previous](#) [Next](#)

Calculates the name of the full moon according to the Maine algorithm.

```
function MoonName(lunation:integer): TMoonName;
```

Description

In the native American cultures every full moons had a special name depending on the season, a tradition adopted by the Maine Farmer's Almanac. The algorithm of that almanac was rediscovered by Sky & Telescope while researching the blue moon tradition. According to this algorithm the moons were named according to their position to the equinoxes and solstices, every season has three regular moons. However as 12 lunar months are shorter then a year about every third year has a season with 4 full moons. In this case the third moon is called a blue moon, the fourth one gets the name the third one would have gotten.

The Maine algorithm has two specialities - it calculates the equinoxes and solstices with the dynamic mean sun instead of the real sun; and the spring equinox is fixed to be on the ecclesiastical equinox of March 20th to make sure the Pink (or Easter) Moon happens just before Easter.

Hint

In additional to those names covered by the [TMoonName](#) type there are two other fixed names for special lunar events. A black moon is usually used to denote a second new moon in a calendar month; a blind moon a calendar month without any full moon. Both are a kind of opposite to the calendar [blue moon](#).

Perigee is the minimal distance of the moon from the earth.

Perihelion is the minimal distance of the earth from the sun.

Solar Terms

[Algorithms](#)

[Top](#) [Previous](#) [Next](#)

Calculates the starting dates of the solar terms

function CalcSolarTerm(year: **Integer**; term: [TSolarTerm](#)): **TDateTime**;

Description

Calculates the dates of the solar terms, which are used in the Chinese calendar for keeping the lunar calendar in sync with the solar movement. The major solar terms are defined as the dates when the position of the sun has a longitude divisible by 30° (the beginning of the [seasons](#) are among these), the minor ones defined as those when it is divisible only by 15° .

The **star time** (siderial time) is the angular position of the spring point at the specific time, and it is used to calculate the horizontal coordinates of stars.

EquationOfTime

[Algorithms](#)

[Top](#) [Previous](#) [Next](#)

Calculates the equation of time in seconds at the given date

function EquationOfTime(date: **TDateTime**): **extended**;

Description

Calculates the difference between the mean solar noon and the apparent solar noon for the given date. Mainly due to the eccentricity of the earth's orbit around the sun the time between two real noons is not constant, but changes with the date - the extreme values are up to 16 minutes difference between the mean and the apparent value. This function returns the value in seconds, and can have both positive and negative sign. A negative sign means that real noon is before 12 o'clock, positive sign it's after 12 o'clock.

Reference

This function is based upon chapter 28 (27) of "[Astronomical Algorithms](#)".

TMoon.Moonstyle

[TMoon](#)

Selects the bitmap style to be used.

property Moonstyle: [TMoonstyle](#);

Description

Selects the bitmap style to be used for both for the picture and [icon](#) property. The value msMonoChrome uses the [MoonColor](#) value for a plain color display, and the following two bitmap types are supported:

msClassicmsColor



Moon image as icon

property Icon: **TIcon**;

Description

The moon image as a TIcon type. The size of the icon calculated depends on the current system metrics - currently only those sizes covered by [TMoonSize](#) can be used. This property, of course, is read-only.

The **julian date** (JD) is a representation for dates often used in astronomy. It is defined as being the number of days elapsed since noon January 1st, 4712 b.c.

Hierarchy

TObject
|
TPersistent
|
TComponent
|
TControl
|
TGraphicControl
|
[TMoon](#)

TMoon.Date

[TMoon](#)

[Top](#) [Previous](#) [Next](#)

The date and time used for the calculation of the moon image

property Date: **TDateTime**;

Description

The date which is used for the calculation of the moon image.

TMoon.MoonSize




[TMoon](#)

Size of the moon image

property MoonSize: [TMoonSize](#);

Description

Size of the moon image, can be 16 pixel, 32 pixel or 64 pixel.

Size	Image
ms16	
ms32	
ms64	

Rotate the image of the moon.

property Rotation: [TRotate](#);

Description

Rotate the image of the moon optionally by 90, 180 or 270 degrees (counterclockwise). Especially the rotation by 180 degrees is needed for locations on the southern hemisphere, as the moon is seen rotated from there. For locations near the equator the rotations of 90 or 270 degrees can be useful, however, the optimum value for the rotation changes with the horizontal position of the moon. Thus the value `rot_location` calculates the angle fitting the current observer's [location](#) and time, as calculated with the [bright limb angle](#). For any other fixed angle as set with [RotationAngle](#) `rot_angle` can be used.

The first four values are retained for compatibility, as the `rot_angle` value can be used to get these angles by setting the `RotationAngle` property appropriately as well.

Toggle the painting of the Apollo 11 marker

property ShowApollo11: **boolean**;

Description

Toggles the painting of the Apollo 11 landing site as a red dot. This dot is only painted when the date is set to a date after July 20th 1969, and ShowApollo11 is set to true.

Moon Phase Angle

[Algorithms](#)

[Top](#) [Previous](#) [Next](#)

Calculates the moon phase angle

function Moon_Phase_Angle(date:TDateTime): **extended**;

Description

Calculates the phase angle of the moon, the position of the bright limb on the moon hemisphere.

Hint

This function returns a negative value for the second half of the month, everything after the full moon. Normally the phase angle is defined to be always positive, so to have this value in this definition just use

```
PhaseAngle := Abs(Moon_Phase_Angle(date));
```

Reference

This function is based upon chapter 48 (46) of "[Astronomical Algorithms](#)".

TZodiac

[Top](#) [Previous](#) [Next](#)

Unit

Moon

```
type TZodiac = (z_aries, z_taurus, z_gemini, z_cancer, z_leo, z_virgo, z_libra,
                z_scorpio, z_sagittarius, z_capricorn, z_aquarius, z_pisces );
```

Description

The values for the 12 zodiac signs in their latin names.

Latin name	Sign	English name	Dates
Aries	^	Ram	Mar 21 - Apr 19
Taurus	_	Bull	Apr 20 - May 20
Gemini	`	Twins	May 21 - Jun 20
Cancer	a	Crab	Jun 21 - July 22
Leo	b	Lion	July 23 - Aug 22
Virgo	c	Virgin	Aug 23 - Sep 22
Libra	d	Scales	Sep 23 - Oct 22
Scorpio	e	Scorpion	Oct 23 - Nov 21
Sagittarius	f	Archer	Nov 22 - Dec 21
Capricorn	g	Sea-Goat	Dec 22 - Jan 19
Aquarius	h	Water-Bearer	Jan 20 - Feb 18
Pisces	i	Fish	Feb 19 - Mar 20

Hint

The exact starting times of the zodiac signs the same as the major [solar terms](#).

Hierarchy

TObject
|
Exception
|
[E_NoRiseSet](#)

Hierarchy

TObject

|

Exception

|

[E_OutOfAlgorithmRange](#)

MoonZodiac

[Algorithms](#)

[Top](#) [Previous](#) [Next](#)

Calculates the astrological zodiac of the moon.

function MoonZodiac(date:**TDateTime**):[TZodiac](#);

Description

Calculates the zodiac sign where the moon is located at the date. This, however, does not mean that the moon is in the astronomical area of that star sign, it is just the [ecliptic longitude](#) measured from the vernal equinox partitioned in areas of 15°. Due to the precession the equinox is moving, it is now at the boundary of Aquarius and Pisces, but the zodiac associated with it is still Aries where the equinox was at time of the Babylonian.

Calculates the ephemerides for physical observation of the moon

```
procedure OpticalLibration(date: TDateTime; var  
    latitude,longitude:extended);
```

```
procedure PhysicalLibration(date: TDateTime; var  
    latitude,longitude:extended);
```

Description

The moon always shows the same side to the earth, however, the excentricity of the moon's orbit and the inclination of the moon equator to the ecliptic cause the actually visible part of the moon surface to be in fact 59% instead of just 50%, an effect called *libration*. This can be put into numbers by the selenographic coordinates at which the earth is in zenith, the so-called CoLongitude and CoLatitude. The main effect of the libration is because of the orbit, and is called the Optical Libration. The actual rotation of the moon changes slightly from the mean rotation, this also affects the libration, however, it is much smaller than the optical libration and always smaller than 0.04° . The Physical Libration, thus, is the addition of both effects.

Reference

These functions are based upon chapter 53 (51) of "[Astronomical Algorithms](#)".

Position angle of axis

[Algorithms](#)

[Top](#) [Previous](#) [Next](#)

Calculates the position angle of the moon rotational axis

function MoonPositionAngleAxis(date:**TDateTime**): **extended**;

Description

The position angle of the rotation axis is measured to the north direction, and because the moon equator has only an inclination of about 1° to the ecliptic this value changes between about $\pm 23.5^\circ$. The effects of the [libration](#) are included in this calculation.

Reference

This function is based upon chapter 53 (51) of "[Astronomical Algorithms](#)".

Bright Limb Angle

[Algorithms](#)

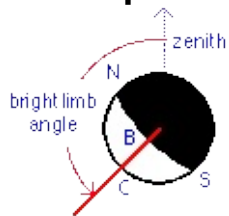
[Top](#) [Previous](#) [Next](#)

Calculates the positional angle of the moon's bright limb

function MoonBrightLimbPositionAngle(date: **TDateTime**):**extended**;

function MoonBrightLimbPositionAngleZenith(date: **TDateTime**; latitude, longitude: **extended**):**extended**;

Description



The bright limb of the moon changes its orientation, either measured towards the north direction or towards the zenith at a given geographic position, as shown in the illustration. The zenith angle describes the apparent rotation of the moon; and it is not defined for the moon being exactly in zenith (or nadir).

Reference

These functions are based upon chapter 48 (46) of "[Astronomical Algorithms](#)".

NextMoonNode

[Algorithms](#)

[Top](#) [Previous](#) [Next](#)

Calculates the date of the next passing of the moon through the nodes

function NextMoonNode(date:**TDateTime**; rising:**boolean**): **TDateTime**;

Description

Calculates the date of the next ascending or descending passing of the moon through the nodes, which is when the geocentric latitude of the moon is 0.

Reference

This function is based upon chapter 51 (49) of "[Astronomical Algorithms](#)".

SunZodiac

[Algorithms](#)

[Top](#) [Previous](#) [Next](#)

Calculates the zodiac sign of the sun.

procedure SunZodiac(date:TDateTime): [TZodiac](#);

Description

Calculates the zodiac sign where the sun is located at the date. This however does not mean that the sun is in the astronomical area of that star sign, it is just the [ecliptic longitude](#) measured from the vernal equinox partitioned in areas of 30°. Due to the precession the equinox is moving, it is now at the boundary of Aquarius and Pisces, but the zodiac associated with it is still Aries where the equinox was at time of the Babylonian.

Hint

The beginning of the zodiacs are calculated by the major [solar terms](#).

Converts ecliptic coordinates for the moving equinox.

Unit

moon_aux

```
procedure ConvertEquinox(source_date, target_date: TDateTime; var
    rektaszension, declination: extended);
```

```
procedure ConvertEquinoxB1950toJ2000(var rektaszension, declination:
    extended);
```

```
procedure ConvertEquinoxDateToJ2000(date: TDateTime; var rektaszension,
    declination: extended);
```

```
procedure ConvertEquinoxJ2000toDate(date: TDateTime; var rektaszension,
    declination: extended);
```

Description

Due to the precession of the earth rotational axis the equinox moves along the ecliptic by about 50" per year. Thus ecliptic coordinates need the reference date for their full definition, either they are calculated with the equinox of the date, or with one of the standard frames J2000 or B1950.

Reference

These functions are based upon chapter 21 (20) of "[Astronomical Algorithms](#)".

Calculates the difference between [UTC](#) and dynamic time

function DynamicTimeDifference(date: **TDateTime**): **extended**;

Description

Calculates the difference between UTC and dynamic time in seconds. UTC is defined by the rotation of the earth which is changing mainly due to tidal effects. To make sure the UTC stays in sync with the actual time leap seconds are inserted occasionally. However, the astronomical calculations need a continuous time frame, the dynamic time. This function returns the offset for any time (in 2001 it is 64 seconds) - since 1972 the number of leap seconds, from 1961 to 1972 the offset was calculated with a (changing) formula. Before 1961 the difference is interpolated from old astronomical observations like solar eclipses, and for future times an extrapolation is used. Due to the fact that the changes of the earth's rotation are unpredictable this extrapolation can turn out wrong as well, so it needs to be handled with care.

What time frame the TDateTime is using is not defined, mostly it will be used as UTC, but as it is a float internally it cannot handle leap seconds, thus it would be better to use it as dynamic time. So it depends on the actual usage if this functions needs to be called for the calculation of an astronomical time or not. Notice that this conversion isn't needed for the sun rise and set times - due to the definition of the UTC the effects of the time frame change and the slowing of the earth rotation cancel out.

Reference

This function is based upon chapter 10 (9) of "[Astronomical Algorithms](#)".

DistanceOnEarth

[Algorithms](#)

[Top](#) [Previous](#) [Next](#)

Calculates the distance between two points on the earth

```
function DistanceOnEarth(latitude1, longitude1, latitude2,  
                           longitude2:extended):extended;
```

Description

Calculates the distance between two geographical coordinates on the earth globe, including the effect of the flatness of the earth. The value is given in kilometers.

The latitude is negative for the southern hemisphere and positive for the northern hemisphere; the longitude is positive for points west of Greenwich, negative for points east, and all given in degrees.

Reference

This function is based upon chapter 11 (10) of "[Astronomical Algorithms](#)".

TMoon.Bitmap

[TMoon](#)

[Top](#) [Previous](#) [Next](#)

Moon image as bitmap

property Bitmap: **TBitmap**;

Description

The moon image as a TBitmap type - the same bitmap that is painted by the component.

TMoon.Color

[TMoon](#)

[Top](#) [Previous](#) [Next](#)

The background color of the moon bitmap

property Color: **TColor**;

Description

The background color used to paint the area around the moon disc. Only used when the [Transparent](#) property is set to false.

TMoon.Location

[TMoon](#)

[Top](#) [Previous](#) [Next](#)

The observer location used for displaying the moon picture

property Location: [TLocation](#);

Description

The observer location used for calculating the apparent rotation of the moon picture. This value is only used in case the [Rotation](#) property is set to rot_rotation.

TMoon.MoonColor

[TMoon](#)

[Top](#) [Previous](#) [Next](#)

The color of the moon disc

property MoonColor: **TColor**;

Description

The color used to paint the moon disc when the [MoonStyle](#) is set to msMonochrome.

Selects the bitmap style to be used.

property MoonStyle: [TMoonStyle](#);

Description

Selects the bitmap style to be used for both for the picture and [Icon](#) property. The value msMonochrome uses the [MoonColor](#) value for a plain color display, and the following two bitmap types are supported:



TMoon.RotationAngle

[TMoon](#)

[Top](#) [Previous](#) [Next](#)

Set the rotational angle of the moon bitmap

property RotationAngle: **integer**;

Description

The angle (in degrees) used for rotating the moon picture by a fixed angle. This value is only used if the [Rotation](#) property is set to rot_angle.

Display the bitmap transparent or with the background color

property Transparent: **boolean**;

Description

Toggles the transparent painting of the moon. If set to false the moon picture is surrounded by the background color set by [Color](#), otherwise only the moon disc will be painted..

TSolarTerm type

[Top](#) [Previous](#) [Next](#)

Unit

Moon

```
type TSolarTerm = (st_z2, st_j3, st_z3, st_j4, st_z4, st_j5, st_z5, st_j6, st_z6,
    st_j7, st_z7, st_j8, st_z8, st_j9, st_z9, st_j10, st_z10, st_j11,
    st_z11, st_j12, st_z12, st_j1, st_z1, st_j2);
```

Description

The solar terms are used in the Chinese calendar as a more generalized seasonal timing. They are divided in major terms (节气 - zhōng qì) which correspond to a solar longitude divisible by 30°, and minor terms (节气 - jié qì) divisible by 15°. The 4 [season](#) beginnings are among the major terms. The major terms also correspond to the beginning of the [solar zodiac](#) in western astrology.

Major Term	Chinese	Unicode	Zodiac	Season	English
Z1 330°	雨水 yǔ shuǐ	96E8 6C34	i Pisces		Rain water
Z2 0°	春分 chūn fēn	6625 5206	^ Aries	Spring	Spring (Vernal) equinox
Z3 30°	谷雨 gǔ yǔ	8C37 96E8	_ Taurus		Grain rain
Z4 60°	小满 xiǎo mǎn	5C0F 6EE1 590F	` Gemini		Grain full Summer

Z5	90°	夏至 xià zhì	81F3	a Cancer	Summer solstice
Z6	120°	大暑 dá shū	5927 6691	b Leo	Great heat
Z7	150°	处暑 chū shū	5904 6691	c Virgo	Limit of heat
Z8	180°	秋分 qiū fēn	79CB 5206	d Libra	Autumn Autumn equinox
Z9	210°	霜降 shuāng jiàng	971C 964D	e Scorpio	Descend of frost
Z10	240°	小雪 xiǎo xuē	5C0F 96EA	f Sagittarius	Slight snow
Z11	270°	冬至 dōng zhì	51AC 81F3	g Capricorn	Winter Winter solstice
Z12	300°	大寒 dà hán	5927 5BD2	h Aquarius	Great cold

Minor Term

J1	315°	立春 lì chún	7ACB 6625		Beginning of spring
J2	345°	惊蛰 jīng zhé	60CA 86F0		Waking of insects
J3	15°	清明 qīng míng	6E05 660E		Pure brightness
J4	45°	立夏 lì xià	7ACB 590F		Beginning of summer
J5	75°	芒种 máng zhòng	8292 79CD		Grain in ear
J6	105°	小暑 xiǎo shū	5C0F 6691		Slight heat
J7	135°	立秋 lì qiū	7ACB 79CB		Beginning of autumn
J8	165°	白露 bái lù	767D 9732		White dew
J9	195°	寒露 hán lù	5BD2 9732		Cold dew

J10	225°	立冬 lì dōng	7ACB 51AC	Beginning of winter
J11	255°	大雪 dà xué	5927 96EA	Great snow
J12	285°	小寒 xiǎo hán	5C0F 5BD2	Slight cold

The DUnit framework (see <http://sourceforge.net/projects/dunit/> for the necessary sources and the documentation) allows to add testing close to the code to be tested, one of the parts of the programming technique called Extreme Programming (XP). But even in classical programming such automatic tests can be very useful to make sure that changes in the code don't change the results.

The project testmoon.dpr applies many of the examples from Meeus to the actual implementations of the functions, and warns when the results are off by more than the deviation caused by the algorithm itself. Of course all the tests work for the released version of the algorithms - but in case you want to modify them these tests can be a good reality check, or to add new tests not yet covered by those in moontest.pas.

Notice that DUnit only works with Delphi 4 and higher as it uses overloading internally a lot.

UTC: Universal Time Coordinated - also commonly known as GMT
(Greenwich Mean Time)