Moon component

Unit

Moon

Description

The unit Moon.pas contains a collection of astronomical algorithms together with a visual component 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 in 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 and so one, 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) check the moon webpage. This software comes as freeware, you may use 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 for 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`.

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 additional ones very useful for both astronomical as well as calendarical application. 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 used for the calculation of the sun position by the VSOP planetary theory. Meeus use a modified version of it by ignoring those terms only needed for higher accuracy. 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 a bit, and it's not possible for Delphi 1.

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...

TMoon Properties

TMoon

In TMoon

- Date
- Icon
- MoonSize
- MoonStyle
- Rotation
- ShowApollo11

Hierarchy

TObject
    |
TPersistent
    |
TComponent
    |
TControl
    |
TGraphicControl
    |
TImage
    |
TMoon

TMoonPhase type

See also

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.

TMoon.Date

TMoon

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.

How to contact

Andreas Hörstemeier
Mefferdatisstraße 16-18
52062 Aachen
Germany

andy@hoerstemeier.de
http://www.hoerstemeier.com

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 any 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.

Component history

Version	Date	Changes
V1.0	1997.04.03	first published version
V1.1	1997-05-21	bug with align property fixed moontool available in 16bit as well daylight saving in moontool corrected
V1.2	1997-12-07	added calculation of seasons, moon/sunrise and -set, perigee and apogee and eclipses new icon property 16x16 bitmap second page in Moontool with the new additional data
V2.0	2001-07-07	Rotation of the moon image "Color" bitmaps New functions for horizontal coordinates of sun and moon Twilight (civil, nautical, astronomical) 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

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

TSeason type

See also

Unit

Moon

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

Description

Ordinal 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)

TEclipse type

See also

Unit

Moon

```
type TEplise = (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 don't 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).

TRotate type

Unit

Moon

```
type TRotate = (rot_none,rot_90,rot_180,rot_270);
```

Description

Rotation angle in mathematical style (counterclockwise).

TMoonSize

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).

TMoon

Hierarchy Properties

Unit

Moon

Description

A descendant of the TImage control 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 put into the Picture property of the TImage; and also into the Icon property (in the size used as the default size for the current system).

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 invisble with the property ShowApollo11.

Astronomical Algorithms

Unit

Moon

Description

A collection of astronomical algorithms based upon the book "Astronomical Algorithms" by Jean Meeus.

Calendar
Julian Date
Julian/Gregorian calendar conversions
Jewish Calendar
Chinese Calendar
Easter Date
PesachDate
Start of seasons
Star time
Corrected Delphi functions

Moon specific
Moon distance
Age of the moon
Next Phase
Last Phase
Current Phase
Nearest Phase
Next Blue Moon
Lunation
Moon diameter
Moon horizontal coordinates
Perigee
Apogee
Moon rise, set and transit
Eclipse

Sun specific
Sun distance
Sun diameter
Sun horizontal coordinates
Sunrise, -set and transit
Twilight
Perihel
Aphel
Eclipse

Seasons

Algorithms See also

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 starting - 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

Reference

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

Easter Date

Algorithms See also

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 occasionaly 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

This function is based upon chapter 8 of "Astronomical Algorithms".

Astronomical Algorithms
by
Jean Meeus

2nd edition (December 1998)
Willmann-Bell; ISBN: 0943396611

Order directly at amazon.com

german edition:
Astronomische Algorithmen
von
Jean Meeus

J.A. Barth, Leipzig; ISBN: 3335004000
currently out of print
Oder directly at amazon.de

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

TMoon.Rotation

TMoon

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 optimal value for the rotation changes with the horizontal position of the moon.

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	

TMoon.Icon

TMoon

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 size covered by TMoonSize can be used. This property is of course read-only.

Julian Date

Algorithms See also

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 day elapsed since noon January 1st, 4712 b.c. It has the advantage of being much easier to use in calculations then 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's December 30th 1899).

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

```
MJD = 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 one.

Reference

These functions are based upon chapter 7 of "Astronomical Algorithms".

Moon distance

Algorithms See also

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".

Sun distance

Algorithms See also

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 AU = 149597869 km
```

Reference

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

Moon diameter

Algorithms See also

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 the distance of the moon.

Reference

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

Perigee

Algorithms See also

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

Reference

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

Sun diameter

Algorithms See also

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".

Apogee

Algorithms See also

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".

Lunation

Algorithms

Calculates the lunation.

```
function Lunation(date:TDateTime):integer;
```

Description

Calculates the lunation of the given date. The lunation is a count of the new moons since January 1st, 1923.

Age of the moon

Algorithms See also

Calculates the age of the moon

```
function AgeOfTheMoon(date:TDateTime):extended;
```

Description

Calculates the age of the moon (in days) for the given time. The age of the moon 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.

Reference

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

Current phase

Algorithms See also

Calculates the current phase

```
function CurrentPhase(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 "Astronomical Algorithms".

Next phase

Algorithms See also

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.

Previous phase

Algorithms See also

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 ones.

Star Time

Algorithms

Calculates the star time.

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

Description

Converts the time to the star 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.

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

Reference

This function is based upon chapter 12 (11) of "Astronomical Algorithms".

Moon Coordinates

Algorithms See also

Calculates the horizontal coordinates of the moon.

```
procedure Moon_Position_Horizontal(date:TDateTime; longitude,latitude:
             extended; var elevation,azimuth: extended);
```

Description

Calculates the horizontal coordinates of the moon at a given time and place. Negative elevation means the moon 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 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.

Reference

This function is based upon chapter 13 (12) and 47 (45) of "Astronomical Algorithms".

Sun Coordinates

Algorithms See also

Calculates the horizontal coordinates of the sun.

```
procedure Sun_Position_Horizontal(date:TDateTime; longitude,latitude:
             extended; var elevation,azimuth: extended);
```

Description

Calculates the horizontal coordinates of the sun at a given time and place. Negative elevation means the sun 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 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.

Reference

This function is based upon chapter 13 (12) and 25 (24) of "Astronomical Algorithms".

Moon Rise and Set

Algorithms See also

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, and and transit on the given date and location. The transit time is the one 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 given in degrees.

Reference

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

Sun Rise and Set

Algorithms See also

Calculates the sub 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, and and transit on the given date and location. The transit time is the one 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 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.

Reference

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

Eclipse

Algorithms See also

Calculates the next eclipse.

```
function NextEclipse(var date: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".

E_OutOfAlgorithmRange

Hierachy

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

Unit

Moon

Description

`E_OutOfAlgorithmRange` is raised when:

- Seasons before 1000 B.C. or after 3000 A.D.
- Easter Date before 1583 or after 2300

Hierarchy

TObject
    |
Exception
    |
E_OutOfAlgorithmRange

E_NoRiseSet

Hierachy See also

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 things like the polar winter.

Hierarchy

TObject
    |
Exception
    |
E_NoRiseSet

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

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

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

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

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

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.

TMoon.ShowApollo11

TMoon

Toggle the painting 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`.

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

TMoon.Moonstyle

TMoon

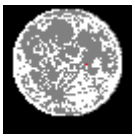
Selects the bitmap style to be used.

```
property Moonstyle: TMoonstyle;
```

Description

Selects the bitmap style to be used for both for the picture and icon property. Currently the following two are supported:

msClassic msColor



Corrected Delphi calendar functions

Algorithms See also

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 version till Delphi 4) 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 flexible the direct calendar functions can be used.

Aphel

Algorithms See also

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.

Reference

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

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

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

Perihelion

Algorithms See also

Calculates the date of the next perihelion

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

Description

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

Reference

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

Gregorian and julian calendar functions

Algorithms See also

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`.

Marking Time
by
Duncan Steel

1 edition (December 8, 2000)
John Wiley & Sons; ISBN: 0471404217

Order directly at amazon.com

PesachDate

Algorithms See also

Calculates the pesach (passover) date

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

Description

Calculates the date of pesach, the jewish holiday. The date is determinded 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".

Jewish Date

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 then 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 start 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".

Chinese Date

Algorithms See also

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 a 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 continous 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 anymore, 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's 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.

Calendrical Calculations
by
Nachum Dershowitz and Edward M. Reingold

2nd revised edition (September 2001)
Cambridge University Press; ISBN: 0521777526

Order directly at amazon.com
Online version

WeekNumber

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 accoring 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.

TChineseDate type

See also

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 `(cylce-1)*60+(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

TChineseCycle type

See also

Unit

Moon

```
type TChineseCycle = record
  zodiac: TChineseZodiac;
  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.

TChineseStem type

See also

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 for the astrological cycles of the chinese calendar. The name of the types represent the chinese name of the stem - there are no translation for these stems.

TChineseZodiac type

See also

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 for the astrological cycles of the chinese calendar. The names are the english names of the correspoing animals of the zodiac.

Nearest phase

Algorithms See also

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 See also

Calculates the date of the next blue moon

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

Description

A "blue moon" 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. However the original definition seems to be a different one - if a season contains four instead of three full moon, then the third one is a "blue moon".

Reference

This function is based upon chapters 49 (47) of "Astronomical Algorithms".

Twilight

Algorithms See also

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 then this it's be so dark artifical light would be needed.
Nautical twilight is defined as the time the sun reaches an elevation of 12 degrees under the horizon. When the sun is deeper then this it's dark enough to have all the bright starts needed for nautical triangulations clearly visible.
Astronomical twilight is defined as the time the sun reaches an elevation ob 18 degrees under the horizon. When the sun is deeper then this it's dark enough to have all stars visible, and the sun is not disturbing astronomical observations at all anymore.

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.

Reference

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

TMoonStyle
Unit

Moon

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

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.