TKronos component

Properties Methods Events

Unit

Kronos

The TKronos component provides easy access to calendaric data based on the Gregorian calendar system. Information is divided into four categories:

Year
Month
Week
Day

all of which give you key-data for a spesific time unit within a year. Additionally TKronos lets you subdivide a day into daytypes to keep track of any events connected to that day. The Daytype class is e very flexible structure that lets you construct any type of event you need to handle. TKronos comes with several predefined daytypes, that is the most common Christian churchdays and international notification days. Easterdays and churchdays related to Easter are progamatically calculated.

You may easily adjust the TKronos component to your needs, that is specify native names and other attributes for the standard daytypes as well as adding any new daytype you want. Adjustments might imply that you derive a new component from TKronos, but you can also handle country or other spesific chronologies by calling methods at runtime - or by loading prewritten definitons from disk.

Obtaining information is simple. By setting one or more of the time unit properties Year, Month, Week, Monthday, Weekday or Daynumber, which together form the Current Date (the date in focus), you can read back information from the corresponding Ext properties (extended information properties) YearExt, MonthExt, WeekExt, and DateExt/Daytypes. There are also numerous methods you can call to retrieve information and perform navigation.

Eventhandlers are implemented for each change of a time unit (OnChangeYear, OnChangeMonth, OnChangeWeek, etc.).

See the following topics for closer explanations of key aspects of the TKronos component:
Genereal guidelines
Using daytypes
Processing daytype classes

TKronos.Year

TKronos
Year stores the year that currently is in focus.

property Year : Word

Description
Use Year to change the current year. A change of Year will allways update the YearExt property.

If you at runtime attempt to set Year to a value that exceeds the limits of the MinYear or MaxYear properties, the exception EKronosError ‘Year out of bounds’ is raised.

Affecting other time unit properties
No other time unit properties will change, except if you move from a leapyear to a non leapyear or vice versa:

If the current year is a leapyear and the current month is February and the monthday is 29, then if you move to a non leapyear the Monthday property will be set to 28. The Daynumber, possibly also Week and Weekday properties will change accordingly.

Moving between a leapyear and a non leapyear will generally affect the Daynumber property if the current date is after February 28.

Affecting other Ext properties
The MonthExt, WeekExt, DateExt and Daytypes properties are updated.

TKronos.Month

TKronos
Month stores the monthnumber that is currently in focus.

property Month : Word

Description
Use Month to change the current month within the current year. A change of Month will allways update the MonthExt property.

If you at runtime attempt to set Month to less than 1 or greater than 12, the exception EKronosError ‘Month out of bounds’ is raised.

Affecting other time unit properties
Changing the month allways affects the Daynumber and the Week properties. Most often it also affects the Weekday property. The Monthday property will not change, except if the current Monthday does not fit the month you are moving to. For instance: If the current Monthday is 31 and you set the new Month value to 11 (November) Monthday is reduced to 30.

Affecting other Ext properties
WeekExt, DateExt and Daytypes are updated.

TKronos.Week

TKronos Example

Week stores the weeknumber that is currently in focus. Weeknumbers are calculated in accordance with the FirstWeekday property.

property Week : Word

Description

Use Week to change the current weeknumber within the current year. A change of Week will allways update the WeekExt property.

There are some tricky things about weeknumbers as a year never consists of a number of whole weeks. The last or first week, or both, are "partial" weeks, that is they contain less than 7 days. On a calendar it may look like a year has 53 weeks (in some years even 54!). However, a week less than 7 days, must be seen as the other part of a week in a bounding year. That is, a partial week number 53 in year 1 is the same physical week as week number 1 in year 2. Have this in mind when reading the further description.

If you at runtime attempt to set Week to less than 1 or greater than the top weeknumber of the current year, the exception EKronosError 'Week out of bounds' is raised. (To obtain the top weeknumber, read the YearExt.NumWeeks field.)

Affecting other time unit properties

Changing the Week allways affects the Daynumber property. It might also affect the Monthday and the Month property. The Weekday property does never change.

In some occations the Year property will be affected as well. Assume you set Week to 53 and the current Weekday is Saturday. However if (the "partial") week 53 does not contain Saturday, TKronos moves to Saturday in week number 1 of next year (the same physical week as 53 in previous year). In addition to the change of year, this means that your week-setting will be corrected.

Affecting other Ext properties

The DateExt and Daytypes properties are updated. MonthExt and YearExt is updated if change of month or year take place.

Week example

Assume the current week is the last week (53) of the year. The current weekday is Sunday which is equal to FirstWeekday. The last week is a partial week with 3 days: Sunday, Monday and Wednesday. The rest of the week belongs to week 1 of next year. You code:

Weekday := Thursday;
Then Year changes to next year. Week changes to 1.

Assume the current week i 15. Current weekday is Thursday. Rest as above. You code:
Week := 53;
Then Year changes to next year. Week changes to 1! 1 and 53 is the same physical week.

TKronos.Weekday

TKronos
Weekday stores the name of the weekday that is currently in focus.

property Weekday : TWeekday

Description
Use Weekday to change the current weekday within the current week. A change of Weekday will allways update the DateExt and Daytypes properties.

Affecting other time unit properties
Changing the Weekday allways affects the Daynumber and Monthday properties. It might also affect the Month property.

On some occations even the Week and Year properties are affected too. This happens if the weekday you are moving to belongs to the first or last week of a bounding year. See the Week property for further explanation of this mechanism.

Affecting other Ext properties
The MonthExt, YearExt and WeekExt properties are updated if change of month, year or week take place.

TKronos.Monthday

TKronos
Monthday stores the number of the monthday that is currently in focus.

property Monthday : Word

Description
Use Monthday to change the current monthday within the current month. A change of Monthday will allways update the DateExt and Daytypes properties.

Do not confuse Monthday and Daynumber - Daynumber is year based (ranges from 1 to 366), Monthday is month based (1-31).

If you at runtime attempt to set Monthday to less than 1 or greater than the maximum value for the month, the exception EKronosError ‘Monthday out of bounds’ is raised. (To obtain the maximum Monthday value, read the MonthExt.NumDays field.)

Affecting other time unit properties
Changing the Monthday allways affects the Daynumber property. It might also affect the Week or Weekday property.

Affecting other Ext properties
The WeekExt property is updated if change of week takes place.

TKronos.Daynumber

TKronos

Daynumber stores the number of the day that is currently in focus.

property Daynumber : Word

Description

Use Daynumber to change the current daynumber within the current year. A change of Daynumber will allways update the DateExt and Daytypes properties.

Do not confuse Daynumber and Monthday . Daynumber is year based (can be a number between 1 and 366), Monthday is month based (1-31).

If you at runtime attempt to set Daynumber to less than 1 or greater than the maximum value for the year, the exception EKronosError ‘Daynumber out of bounds’ is raised. (To obtain the maximum daynumber, read the YearExt.NumDays field.)

Affecting other time unit properties

Changing the daynumber might also affect the Month, Monthday, Week or Weekday properties.

Affecting other Ext properties

MonthExt and WeekExt is updated if change of month or week take place.

TKronos.Daytypes

TKronos Example See also

Runtime and read only

Daytypes stores information about the daytypes registered for the date that is currently in focus.

property Daytypes[AnIndex] : TDaytype

Description

Use Daytypes in connection with DaytypeCount to retrieve the registered daytypes.

Note

Yeartypes are not stored in the Daytypes property. To obtain the yeartypes you must use the FetchYeartype function.

Examples using the Daytypes property

This example examines the current date looking for user defined daytypes that meet a certain condition:

```
var
    i : Integer;
    MyDaytype : TDaytype;
:
:
DecodeDate(Date, Y, M, D);
for i := 1 to DaytypeCount do
begin
       MyDaytype := Daytypes[i];
       if MyDaytype.Id >= FirstUserId then
       {Test userdefined daytypes}
       begin
             if Y - MyDaytype.FirstShowUp = 100 then
              ShowMessage(‘100 years anniversary for ‘ + MyDaytype.TheName);
       end;
end;
```

This example lists the daytypes that are relevant for a certain month:

```
var
   i, j : Integer;
   // Assume your form contains the listbox L.
begin
   Month := 8 // August for example
   for i := 1 to MonthExt.Numdays do // Loop days of month
   begin
        Daynumber := i; // Make each monthday the current date
            if DaytypeCount > 0 then
            // If daytypes are registered with the date, make heading
               L..Items.Add(DateExt.Dayname + ‘ ‘ + IntToStr(Monthday) + ‘.’);
        for j := 1 to DaytypeCount do
                  L.Items.Add(‘      ‘ + Daytypes[i].TheName); // List daytypes for the date
   end;
```

Relevant topics:
Using daytypes
Processing daytype classes

TKronos.DaytypeCount

TKronos
Runtime and read only

property DaytypeCount : Word

DaytypeCount stores the number of Daytypes registered with the date that is currently in focus.

Description
Use DaytypeCount in connection with Daytypes to retrieve the registered daytypes.

See also:
Daytypes

TKronos.FirstWeekday

TKronos Example

FirstWeekday determines which weekday starts the week.

property FirstWeekday : TWeekday

Description

The default value is Sunday. Alter it to adjust to other requirements. The value of FirstWeekday influences how TKronos computes weeknumbers and how it organizes the MonthImage table.

Affecting time unit properties

Changing FirstWeekday might affect the Week property.

Affecting Ext properties

WeekExt is updated if change of week takes place.

First Weekday example

These code fragments show the conncetion between DateExt DayOfWeeknumber (DOW) and FirstWeekday.

```
Weekday := Wednesday;
FirstWeekday := Monday;

// DOW = 3

FirstWeekday := Thursday;

// Now DOW = 7
```

In the MonthExt.MontImage table the column numbers are DOW-numbers:

```
FirstWeekday := Monday;
// MonthImage[1,1] is the Monday cell

FirstWeekday := Thursday;
//   MonthImage[1,1] is the Thursday cell
```

TKronos.MinYear

TKronos See also

MinYear determines which year is the lower year boundary for the calendar.

property MinYear : Word

Description

The default and minimum value is 1. Use MinYear to limit the range of years a user can access.

If you at runtime enter a year that is greater than the value the MaxYear property the exception EKronosError ‘MinYear out of bounds’ is raised. Also if you at runtime set MinYear to a value that renders the current date illegal the exception EKronosError ‘Cannot set. The value of MinYear conflicts with the current date.’ is raised.

Relevant topics:
MaxYear

TKronos.MaxYear

TKronos See also

MaxYear determines which year is the higher year boundary for the calendar.

property MaxYear : Word

Description

The default and maximum value is 9999. Use MaxYear to limit the range of years a user can access.

If you at runtime enter a year that is less than the value the MinYear property the exception EKronosError 'MaxYear out of bounds' is raised. Also if you at runtime set MaxYear to a value that renders the current date illegal the exception EKronosError 'Cannot set. The value of MaxYear conflicts with the current date.' is raised.

Relevant topics:
MinYear

TKronos.DefaultToPresentDay

TKronos
DefaultToPresentDay defines which date TKronos makes the current date on creation.

property DefaultToPresentDay : Boolean

Description
The default value is True, that means the date of today will be the current date on start up. If False the designtime date will become the current date.

Note that if you at designtime shift from False to True the time unit properties are not forced to reflect the date of today. You are allways free to use the Object Inspector to manipulate all the time unit properties regardless of the value of DefaultToPresentDay. However the next time you open your project the Object Inspector will initialize TKronos to the date of today.

Changing the DefaultToPresentDay during runtime has no effect.

TKronos.WeekHolidays

TKronos
WeekHolidays defines standard holidays for all weeks in a year.

property WeekHolidays : TWeekHolidays

Description
The default value is [Sunday, Saturday]. Alter WeekHolidays if your calendar uses another standard. The value of this property will be reflected in the DateExt.Holiday field.

Changing the WeekHolidays is automatically follwed by an update of the DateExt property.

TKronos.AllowUserCalc

TKronos See also

AllowUserCalc permits daytype showups to be calculated from outside the daytype object.

property AllowUserCalc : Boolean

Description

The standard value is False. Set to True to enable user calculation and triggering of the OnCalcDaytype event.

Relevant topics:
OnCalcDaytype
Using daytypes

TKronos.HidePredefineds

TKronos
HidePredefineds controls whether the predefined daytypes will show up on the calendar.

property HidePredefineds : Boolean

Description
The standard value is False. Set to True if you want to keep clear of predefined daytypes. This has the effect that neither of the Churchday, Holiday and Flagday properties will influence the corresponding fields in the DateExt property.

Note
Hiding predfined daytypes does not remove them from the daytype list.

TKronos.DateExt

TKronos
Read and runtime only.

DateExt stores extended information about the date that is currently in focus. TKronos updates this property whenever the current date changes.

property DateExt : TDateExt

Description
Use DateExt to read details about the current date.

Be sure to understand the connection between the DateExt and the Daytypes property. It works like a one to many relationship where DateExt represents the master record and Daytypes the detail records. DateExt stores basic data as the dayname, the daynumber and so on, while Daytypes tells which role(s) the day plays on the calendar (Christmas Eve, Easter Eve, etc.).

If you look at the DateExt record you might think it contains several redundant fields, as the Year, MonthNumber and WeekNumber fieldes also are available through the corresponding up to date properties. Have in mind however that you might work with a DateExt record that does not represent the current date.

Note how the fields Churchday, Holiday and Flagday work. If the dayname is one of the WeekHolidays, Holiday is allways set to True. Generally both Churchday, Holiday and Flagday are set to True if any of the daytypes registered for the day have these attributes set to True.

TKronos.MonthExt

TKronos Example

Read and runtime only.

MonthExt stores extended information about the month that is currently in focus. TKronos updates this property whenever the current month changes.

property MonthExt : TMonthExt

Description

Use MonthExt to read details about the current month.

MonthExt example

Here is an example of how easily you can create a month calendar by means of the MonthExt.MonthImage table:

Assume you have a StringGrid component 7 rows and 8 columns:

```
// Fill in the daynames in the first row
for i := 1 to 7 do
Grid.Cells[i,0] := Daynames[DOWToDaynameIndex (i)];
{Get dayname for the Delphi dayname array. The i variable
is the DayOfWeeknumber. The DOWToDaynameIndex
function returns the index to use with the array}

Month := 3; // Chose a month, March for example.

//Fill in the weeknumbers
for i := 1 to MonthExt.NumWeeks do
if MonthExt.MonthImage[i,0] > 0 then
        Grid.Cells[0,i] := IntToStr(MonthExt[i,0]);
        // Weeknumbers > 0 belong to the current month

// Fill in the monthdays
for i := 1 to MonthExt.NumWeeks do
begin
        for j := 1 to 7 do
        begin
                if MonthExt.MonthImage[i,j] > 0 then
                {Numbers > 0 belong to the current month}
                begin
                        Daynumber := MonthExt.MonthImage[i,j];
                        {Make the daynumber in the MonthImage cell the
                        current date.}
                        Grid.Cells[j,i] := IntToStr(Monthday);
                          // Print the monthday number of the current date.
                end;
        end;
end;
```

TKronos.WeekExt

TKronos
Read and runtime only.

WeekExt stores extended information about the week that is currently in focus. TKronos updates this property whenever the current week changes.

property WeekExt : TWeekExt

Description
Use WeekExt to read details about the current week.

TKronos.YearExt

TKronos
Read and runtime only.

YearExt stores extended information about the year that is currently in focus. TKronos updates this property whenever the current year changes.

property YearExt : TYearExt

Description
Use YearExt to read details about the current year.

TKronos.FirstUserId

TKronos See also

Read and runtime only

property FirstUserId : Word

FirstUserId stores the identifier that is the value next to the last predefined daytype.

Description

Use FirstUserId to keep track of identifiers assigned to daytypes you create in addition to the prefefined types. The first new daytype is assigned the value FirstUserId, the next FirstUserId + 1, etc.

Relevant topics:
AddDaytype

TYearExt type

TKronos

Unit

Kronos

TYearExt defines extended attributes for a year.

Type

```
TYearExt = record
        Year : Word;
        NumDays : Word;
        NumWeeks : Word;
        LeapYear : Boolean;
        YearTypeCount : Word;
end;
```

Description

Field	**Meaning**
Year	Number of year
NumDays	Number of days of the year
NumWeeks	Number of weeks of the year
LeapYear	True if Year is a leapyear.
YearTypeCount	Number of yeartypes registered with the year

Comments

The NumWeeks field stores the top weeknumber of the year, not the number of whole weeks. Normally the top weeknumber is 53, somtimes 54. It is never 52.

TMonthExt type

TKronos

Unit

Kronos

TMonthExt defines extended attributes for a month.

Type

```
TMonthExt = record
    Year : Word;
    MonthNumber : Word;
    MonthName : String;
    FirstDay, LastDay : Word;
    NumDays : Word;
    NumWeeks : Word;
    FirstWeek, LastWeek : Word;
    MonthImage : TMonthImage
end;
```

Description

Field	**Meaning**
Year	The year that the month belongs to
MonthNumber	Number of month
MonthName	Name of month
FirstDay	The year based daynumber that starts the month.
LastDay	The year based daynumber that ends the month
NumDays	Number of days of the month
NumWeeks	Number of weeks that is comprised by the month
FirstWeek	The weeknumber that starts the month
LastWeek	The weeknumber that ends the month
MonthImage	A table that organizes the month in columns and rows.

Comments

NumWeeks counts the weeknumbers that are in touch with the month. It is not (NumDays div 7), but (Lastweek - FirstWeek + 1).

FirstDay and LastDay are year based, that is they store numbers between 1 and 366.

TWeekExt type

TKronos

Unit
Kronos

TWeekExt defines extended attributes for a week.

Type

```
TWeekExt = record
        Year : Word;
        WeekNumber : Word;
        FirstDay, LastDay : Word;
 end;
```

Description

Field	**Meaning**
Year	The year that the week belongs to
WeekNumber	Number of week
FirstDay	The year based daynumber that starts the week.
LastDay	The year based daynumber that ends the week

Comments
FirstDay and LastDay are year based, that is they store numbers between 1 and 366.

TDateExt type

TKronos

Unit
Kronos

TDateExt defines extended attributes for a date.

Type

```
  TDateExt = record
      Year : Word
      DayName : String;
      DayOfWeekNumber : Word;
      Monthday : Word;
      Daynumber : Word;
      DaytypeCount : Word;
      DaytypeId : TDaytypeID;
      MonthNumber : Word;
      WeekNumber : Word;
      Holiday : Boolean;
      Churchday : Boolean;
      Flagday : Boolean;
  end;
```

Description

Field	**Meaning**
Year	Year part of the date
DayName	The name of the day
DayOfWeekNumber	The weekday number (1 = the day that starts the week)
Monthday	The monthday number (1-31)
Daynumber	The year based daynumber (1-366)
DaytypeCount	Number of daytypes attatched to the date
DaytypeId	An array that stores the identifiers, if any, of the attached daytypes. DaytypeCount tells the number of used indexes.
MonthNumber	Month part of the date (1-12)
WeekNumber	Number of week to which the date belongs (1-54)
Holiday	True if the day is a holiday
Churchday	True if the day is a religious day
Flagday	True if the dayt is a flagday

TDaytype type

TKronos See also

Unit
Kronos

TDaytype defines the base class for a TKronos daytype.

Type

```
TDaytype = class(TPersistent)
	property TheDate : Word;
	property TheName : String[50];
	property Id : Word;
	property FirstShowUp : Word;
	property LastShowUp : Word;
	property ShowUpFrequency : Word;
	property RelDaytype : Word;
	property Offset : Integer;
	property Churchday : Boolean;
	property Holiday : Boolean;
	property Flagday : Boolean;
	property UserCalc : Boolean;
	property Tag : Integer;

	constructor Create
	(DaytypeDef : TDaytypeDef);
	procedure Update(DaytypeDef : TDaytypeDef; StartUserId : Word);
	procedure SetId(AnId : Word);
end;
```

Description

The TDaytype class contains the base structure of a TKronos daytype. You can derive new classes from TDaytype to meet special requirements.

All properties are read only.

Property	**Meaning**
TheDate	Showup date for the daytype
TheName	The name of the daytype
Id	The identifier of the daytype
FirstShowUp	First year the daytype is shown on the calendar
LastShowUp	Last year the daytype is shown on the calendar
ShowUpFrequency	The year interval between each showup
RelDaytype	Daytype id for a standard church daytype to be used as a starting point for an offset calculation of showup date.
Offset	Offset value from RelDaytype.
Holiday	True if the daytype is a holiday
Churchday	True if the daytype is a religious day
Flagday	True if the daytype is a flagday
UserCalc	True if the showupdate is to be calculated by the OnCalcDaytype event.

Tag	General purpose field. Allways 0 for predefined types.

Procedures	**When to use**
Create	To add a new daytype, you must first create it. Make your definition by setting the daytype attributes in the TDaytypeDef record. Then create the object and send it as parameter with the AddDaytype procedure.
Update	Update is used internally by TKronos to perform changes to an existing daytype definition. *Never* call this procedure yourself. To change a daytype definition call the TKronos.UpdateDaytype procedure.
SetId	Used internally by Tkronos to assign an identifier to a daytype. *Never* call it yourself.

Relevant topics:
Using daytypes
Processing daytype classes
TDaytypeDef type

TMonthImage type

TKronos

Unit

Kronos

TMonthImage defines a two dimentional array that corresponds to the columns and rows of a month calenadar.

Type

TMonthImage = array[1..6, 0..7] of Smallint

Description

The first dimention represents the rows, the second the columns, that is the weeknumbers and the weekdays. Note the following:

Index [n,0] contains
either positiv numbers for the weeks that are comprised, totally or partially, by the current month
or negative numbers for weeks that totally belong to a bounding month.

Indexs [n,1..7] contains
either positiv values for the *year based* daynumbers in the current month
or negative values for the *monthday* numbers in bounding months.

A variable of the TMonthImage type could look like this:

Idx	0	1	2	3	4	5	6	7
1	12	-28	-29	-30	91	92	93	94
2	13	95	96	97	98	99	100	101
3	14	102	103	104	105	106	107	108
4	15	109	110	111	112	113	114	115
5	16	116	117	118	119	120	121	-1
6	-17	-2	-3	-4	-5	-6	-7	-8

The weekday column 1 corresponds to the first day of the week, that is the day defined by the FirstWeekday property.

TDaytypeId type

TKronos

Unit

Kronos

TDaytypeID stores the identifiers of the daytypes that are attached to a date.

Type

TDaytypeID = array[1..255] of Word

Description

TDaytypeID is a field of the TDateExt type. When TKronos fills a variable of TDateExt the daytypes connected to the date, if any, are located and referenced in the TDaytypeId array. You seldom need to access the array directly. TKronos make use of it internally when setting the Daytypes property or when you call the function FetchDaytype.

TWeekday type

TKronos

Unit

Kronos

TWeekday defines the days of the week.

Type

TWeekday = (Sunday, Monday, Tuesday, Thursday, Friday, Saturday)

TWeekHolidays type

TKronos

Unit

Kronos

TWeekHolidays defines the standard holidays of a week.

Type

TWeekHolidays = set of TWeekday

Daytype constants

TKronos

Unit

Kronos

The Daytype constants represents predefined daytypes.The ch prefix defines churchdays, the co prefix defines common international notification days.

Description

Constant	**Value**	**Meaning**
chAdvent1	1	First Sunday of Advent
chAdvent2	2	Second Sunday of Advent
chAdvent3	3	Third Sunday of Advent
chAdvent4	4	Fourth Sunday of Advent
chChristmasEve	5	Christmas Eve
chChristmasDay	6	Christmas Day
chBoxingDay	7	Boxing Day (the day after Christmas day)
chNewYearEve	8	New Year's Eve
chNewYearDay	9	New Year's Day
chAshWednesday	10	Ash Wednesday (Lent)
chShroveTuesday	11	Shrove Tuesday (Lent)
chPalmSunday	12	Palm Sunday (Sunday before Easter Sunday)
chMaundyThursday	13	Maundy Thursday (Thurday before Easter Sunday)
chGoodFriday	14	Good Friday (Friday before Easter Sunday)
chEasterEve	15	Easter Eve
chEasterSunday	16	Easter Sunday
chEasterMonday	17	Easter Monday
chWhitEve	18	Whit Eve
chWhitSunday	19	Whit Sunday
chWhitMonday	20	Whit Monday
chAscensionDay	21	Ascension Day
coUNDay	22	United Nations Day
coWomensDay	23	International Womens Day
coMayDay	24	May Day
coLiteracyDay	25	International Literacy Day
UserDaytype	26	Start of userdefined daytypes

TKronos.SetCountrySpecifics

TKronos

Redefines standard daytypes and adds those of your own.

procedure SetCountrySpecifics; virtual

Description

Override this protected procedure when deriving a new country spesific TKronos component. In SetCountrySpecifics you can place calls to AddDaytype and SpecifyStandardDay to give the calendarium a stable, reusable profile.

Any new daytype you add wil become part of the basic daytype list, that is the daytypes permanently tied to the calendar profile. Such daytypes cannot be deleted, but can be adjusted through the SpecifyStandardDay or UpdateDaytype method - or by creating a new definition in an external file to be loaded with the LoadFromFile method.

A closer explanation of how to use SetCountrySpecifics is found in the topic Using daytypes.

TKronos.ExistsDaytype

TKronos
Checks if a daytype with the same name as ADaytypeName already exists.

function ExistsDaytype(ADaytypeName : String) : Word

Description
Use ExistsDaytype to prevent the daytype list from containing duplicate names. Duplicate names may be problematic as serach functions using the daytypename as key only returns the first found instance of the daytype.

The function returns the number of   daytypes with the same name as ADaytypeName.

TKronos.AddDaytype

TKronos See also Example

Adds a user defined daytype to the daytype list.

function AddDaytype(Daytype : TDaytype) : Word;

Description

Use AddDaytype to add a new daytype object to the daytypelist. The daytype list consists of the predefined church and common daytypes plus the types you define yourself. The function returns the identifier of the added daytype.

To add a new daytype object you must first create it with the TDaytype.Create method. To delete a daytype never destroy it directly, but call the DeleteUserDaytype method. All daytype objects are automatically disposed of as part of the destroying process of TKronos itself.

Every new daytype you add is assigned an identifier you may use to reference the daytype. The identifiers are incremented by 1 for each new add in. The FirstUserId property holds the identifier of the first daytype that is added, the next FirstUserId + 1, and so on.

Identifiers are useful when working with predefined daytypes. You might also use identifiers with daytypes added "on the fly", by loading a calendar defintion from a file for instance or by using definitions stored in a library unit. However, if your application deletes and add daytypes dynamically, the identifiers may be of less value. Say you add three daytypes which are assigned the id-numbers 26, 27 and 28. Number 27 is deleted. If you save this definition to a file and later reload it, the daytype number 28 becomes 27. When initializing a calendar profile TKronos allways creates a contigious row if id-numbers.

Note 1

If you set both the ADate field and the ARelDaytype field of the TDaytypeDef object to zero, you create a yeartype rather than a daytype.

Note 2

If you set the AUserCalc field of the TDaytypeDef object to True, you create a user calculated daytype. Values of ADate and ArelDaytype fields are then ignored.

Note 3

To prevent duplicate daytype names call the ExistsDaytype function before adding the new daytype.

Relevant topics:
DeleteUserDaytype
ClearUserDaytypes
Daytypes property
DateExt property
FetchYearType
Using daytypes
Processing daytype classes
OnCalcDaytype event

AddDaytype example

```
var
      DaytypeDef : TDaytypeDef;
:
with DaytypeDef do
begin
               AName := ’10 days left to Easter’;
               ADate = 0;
               AReldayType = chEasterSunday;
               AnOffset = -10;
               AFirstShowUp = 1;
               ALastShowUp := 9999;
               AShowUpFrequency = 1;
               AHoliday := False;
               AChurchday := False;
               AFlagDay := False;
               AUserCalc := False;
               ATag := 0;
               AddDaytype(TDaytype.Create(DaytypeDef));
end;
:
```

TKronos.UpdateDaytype

TKronos See also Example

Updates an exisiting userdefined daytype with a new definition.

procedure UpdateDaytype(AnId : Word; ADaytypeName : String; DaytypeDef : TDaytypeDef)

Description

Use UpdateDaytype to change one or more of the attributes of a userdefined daytype that is currently loaded. Pass the id or the name of the daytype you want to change in the AnId/AName parameter. If the daytype is not found the exception EKronosError 'Daytype not found' is raised.

AnId/ADaytypeName is mutually exclusive. To search for an ID set ADaytypeName to an empty string. To serach for a name, set AnId to 0. If both AnId and ADaytypeName have values, the id value is the preferred key.

A call to this procedure also updates the DateExt and Daytypes properties.

To change one of the predefined church or common days use the SpecifyStandardDay procedure.

Note 1

To prevent duplicate daytype names call the ExistsDaytype function before updating the new daytype.

Note 2

You have limited control over predefined daytypes added in TKronos descendents through the SetCountrySpecifics method. Setting new values for the fields Date, Reldaytype, Offset, FirstShowUp, LastShowUp and ShowUpFrequency will have no effect.

Relevant topics:
AddDaytype
DeleteUserDaytype
ClearUserDaytypes
GetDaytypeDef

TKronos.GetDaytypeDef

TKronos See also Example

Retrieves a user daytype definition.

function GetDaytypeDef(AnId : Word; ADaytypeName : String) : TDaytypeDef

Description

Use GetDaytypeDef to obtain the definition of a daytype. If the daytype is not found the exception EKronosError ‘Daytype not found’ is raised. This function is useful when you want to make changes to an existing user daytype definition.

AnId/ADaytypeName is mutually exclusive. To search for an ID set ADaytypeName to an empty string. To serach for a name, set AnId to 0. If both AnId and ADaytypeName have values, the id value is the preferred key.

GetDaytypeDef example

To alter the definition av an exisiting user defined daytype:

```
var
   MyDaytype : TDaytypeDef;
:
:
  MyDaytype := GetDaytypeDef(0,‘My daytype’);
  MayDaytype.AName := ‘Your daytype’;
  MayDaytype.ADate := 1030;
  UpdateDaytype(0, ‘May daytype’, MyDaytype);
:
:
```

Relevant topics:
AddDaytype
DeleteUserDaytype
ClearUserDaytypes
UpdateDaytype

TDaytypeDef type

TKronos

Unit

Kronos

TDaytypeDef contains the base definition of a daytype.

Type

```
TDayTypeDef = record
   AName : String[50];
   ADate : Word;
   ARelDaytype : Word;
   AnOffset : Integer;
   AFirstShowUp : Word;
   ALastShowUp : Word;
   AShowUpFrequency : Word;
   AChurchday : Boolean;
   AHoliday : Boolean;
   AFlagday : Boolean;
   AUserCalc : Boolean;
   ATag : Integer;
end;
```

Description

Field	**Meaning**
AName	The name of the daytype. Max 50 characters
ADate	The showup date. It must be formatted as Monthnumber * 100 + Monthday. March 15 is for example equal to 315. You may also pass 0 in this parameter. See below.
ARelDaytype	Use a daytype constant in connection with an offset value to make the showup date relative to any of the prefefined church daytypes. In that case set Date to 0. Set RelDaytype to 0 if you use a fixed date. If you set both Date and RelDaytype to 0 you create a yeartype.
AnOffset	Use in connection with RelDaytype to position the showup date relative the chosen daytype. Positive numbers move forwards, negative numbers backwards. Example: RelDaytype = chChristmasEve. Offset = -1. Resulting date wil be the day before Christmas Eve. Set to 0 if you use a fixed date.
AFirstShowUp	First year the daytype is shown on the calendar
ALastShowUp	Last year the daytype is shown on the calendar
AShowUpFrequency	The year interval between each showup
AHoliday	True if the daytype is a holiday
AChurchday	True if the daytype is a religious day
AFlagday	True if the daytype is a flagday

AUserCalc	True if the if the showup date of the daytype is calculatetd through the OnCalcDaytype event.
Tag	General purpose field. Allways 0 for predefined types.

TKronos.ClearUserDaytypes

TKronos See also

Clears all the user defined daytypes from the daytype list.

procedure ClearUserDaytypes

Description

Use ClearUserDaytypes to remove all the user defined daytypes from the list. A call to this procedure also updates the DateExt and Daytypes properties.

Note

ClearUserDaytypes does not affect the predefined daytypes which cannot be deleted.

Relevant topics:
DeleteUserDaytype
AddDaytype

TKronos.DeleteUserDaytype

TKronos See also

Deletes a user defined daytype from the daytype list.

procedure DeleteUserDaytype(AnId: Word; ADaytypeName : String)

Description

Use this procedure to delete a user defined daytype matching AnId/ADaytypeName from the daytype list.

AnId/ADaytypeName is mutually exclusive. To search for an ID set ADaytypeName to an empty string. To serach for a name, set AnId to 0. If both AnId and ADaytypeName have values, the id value is the preferred key.

A call to this procedure also updates the DateExt and Daytypes properties.

Note

Predefined daytypes cannot be deleted.

If ADaytypeName is not found the exception EKronosError 'Daytype <ADaytypeName> not found' is raised.

Relevant topics:
ClearUserDaytypes
AddDaytype

TKronos.SpecifyStandardDay

TKronos See also

Specifies the name and the staus attributes for a standard TKronos daytype.

procedure SpecifyStandardDay(AnId : Word; AName : String;
IsHoliday, IsFlagday : Boolean);

Description

Uses SpecifyStandardDay to redefine the name and the status attributes for a standard TKronos daytype. The standard daytypes come with English names and False for the Holiday and Flagday attributes. To set country spesifc attributes you must call this procedure for every relevant daytype. If you don't, TKronos will use the standard values.

Parameters

AnId : Word;
The identifier for the daytype you want to redefine, e.g. chChristmasDay.

AName : String
The new name of the daytype.

IsHoliday : Boolean
Set to True if you wish to mark the day as a holiday

IsFlagday : Boolean
Set to True if you wish to mark the day as a flagday.

Note

To check for duplicate daytype names call the ExistsDaytype function.

Relevant topics:
Daytype constants
SetCountrySpecifics

TKronos.FetchYearExt

TKronos Example

Fetches extended information about a year.

function FetchYearExt(AYear : Word) : TYearExt

Description

Use FetchYearExt to obtain extended information about a year wihout changing the current year.

Note

You should use FetchYearExt instead of temporarily making the target year the current one. The Fetch functions are faster and more to the point when you want to obtain information about time units outside the current date.

TKronos.FetchMonthExt

TKronos Example

Fetches extended information about a month.

function FetchMonthExt(AYear, AMonth : Word) : TMonthExt

Description

Use FetchMonthExt to obtain extended information about a month in a specified year without changing the current year/month.

You should use FetchMonthExt instead of temporarily making the target month the current one. The Fetch functions are faster and more to the point when you want to obtain information about time units outside the current date.

TKronos.FetchWeekExt

TKronos Example

Fetches extended information about a week.

function FetchWeekExt(AYear, AWeek : Word) : TWeekExt

Description

Use FetchWeekExt to obtain extended information about a week in a specified year without changing the current year/week.

Note

You should use FetchWeekExt instead of temporarily making the target week the current one. The Fetch functions are faster and more to the point when you want to obtain information about time units outside the current date.

TKronos.FetchDaytype

TKronos See also Example

Fetches one of the daytypes attched to a date.

function FetchDaytype(ADateExt : TDateExt; AnIndex : Word) : TDaytype

Description

Use FetchDaytype to extract the daytypes registered with the date held in ADateExt. To fill ADateExt use one of the FetchDateExt functions.

Note

You should use FetchDateExt/FetchDaytype instead of temporarily making the target date the current one. The Fetch functions are faster and more to the point when you want to obtain information about dates outside the current date.

Relevant topics:
FetchDateExt

TKronos.FetchYeartype

TKronos See also Example

Fetches one of the yeartypes attached to the year.

function FetchYeartype(AYearExt : TYearExt; AnIndex : Word) : TDaytype

Description

Use FetchYeartype to extract the yeartypes registered with the year held in AYearExt. To fill AYearExt use the FetchYearExt function. The YeartypeCount field of the TYearExt record tells you how many yeartypes there are.

Note

FetchYearExt is somewhat of a specialty as it returns a TDaytype object, of which the Churchday, Holiday and Flagday fields are irrelevant. Moreover there is no Yeartypes property you can investigate as you can with the Daytypes property. In fact, a yeartype is a daytype that is not attached to a spesific date. You register a yeartype with the AddDaytype method as you would do with a normal daytype, but set both the Date field and the RelDaytype field to zero.

Instead of linking such daytypes to each an every day in a year, they are excluded from the Daytypes property and stored for themselves. This prevents duplicating information, thereby speeding up performance.

Because there is no Yeartypes property, you also have to use FetchYeartype to obtain the yeartypes of the current year. Pass the YearExt property as the AYearExt parameter.

Relevant topics:
AddDaytype
Using daytypes
Daytypes property

FetchYeartype example

This example shows how to make a list of year events.

```
var
   L : TListbox;
   I : Integer;
   YExt : TYearExt;
    DType : TDaytype;
begin
    If IsThisYear(2000) then
        YExt := YearExt
    else
        YExt := FetchYearExt(2000);
 {Retrieve information about year 2000. If this is the current year the information is already at hand}
    for I := 1 to YExt.YeartypeCount do
    begin
        DType := FetchYeartype(YExt, I);
        L.Items.Add(DType.Name);
    end;
end;
```

Fetch example

This example demonstrates two ways to retrieve information from dates outside the current date:

```
var
    i : Word;
    DateInf : TDateExt;
    Daytype : TDaytype;
    YearInf : TYearExt;
:
:
SaveCD; {Save current date}
DisableEvents(True) ;
{Disable event triggering when performing operations on dates that are not the real current date}
try
  GotoDate(2000, 1, 1); {Put focus on target date}
  for i := 1 to YearExt.Numdays do
  begin
         Daynumber := i; {New current date}
         for j := 1 to DaytypeCount do
         begin
                 Daytype := Daytypes[j];
                 {Do some action}
         end;
  end;
finally
   RestoreCD; {Back to the real current date}
   DisableEvents(False);
end;
```

Note that every time a new date becomes the current date all the Ext properties are laoded with information. This is waste of time as long as you only need information on a spesific time unit level. This is a much more efficient way to loop days:

```
YearInfo := FetchYearExt(2000);
for i := 1 to YearInfo.Numdays do
begin
        DateInf := FetchDateExtDn(2000, i);
        for j := 1 to DateInfo.DaytypeCount do
        begin
                 Daytype := FetchDaytype(DateInfo, j);
                 {Do some action}
        end;
end;
```

This code runs faster and there is no need for saving and restoring the current date. It might of course be cases when you have good reasons for using the first method, but most of the time you probably will manage well with the Fetch functions.

TKronos.FetchDateExt

TKronos See also Example

Fetches extended information about a date.

function FetchDateExt(AYear, AMonth, AMonthday : Word) : TDateExt

Description

Use FetchDateExt to obtain extended information about a date wihout changing the current date.

Note

You should use FetchDateExt instead of temporarily making the target date the current one. The Fetch functions are faster and more to the point when you want to obtain information about dates outside the current date.

Relevant topics:
FetchDaytype
FetchDateExtDt
FetchDateExtDn

TKronos.FetchDateExtDt

TKronos See also Example

Fetches extended information about a date.

function FetchDateExtDt(ADate : TDateTime) : TDateExt

Description

The same function as FetchDateExt, except you pass a single TDateTime parameter instead of year, month and monthday.

Relevant topics:
FetchDateExt
FetchDateDn

TKronos.FetchDateExtDn

TKronos See also Example

Fetches extended information about a date.

function FetchDateExtDn(AYear, ADaynumber : Word) : TDateExt

Description

The same function as FetchDateExt, except you pass Daynumber as parameter instead of month and monthday.

Relevant topics:
FetchDateExt
FetchDateExtDt

TKronos.IsLeapyear

TKronos

Determines if a year is a leapyear.

```
function IsLeapYear(AYear : Word) : Boolean
```

Description

Use IsLeapyear to find out if a year is a leapyear without changing the current year. Returns True if leapyear.

TKronos.IsLastDayOfMonth

TKronos See also

Determines if a monthday is the last day of a month

function IsLastDayOfMonth(AYear, AMonth, AMonthday : Word) : Boolean

Description

Use IsLastDayOfMonth to find out if AMonthday in AYear is the last day of AMonth without changing the current year/month. Returns True if last day.

Relevant topic:
IsLastWeekOfYear

TKronos.IsLastWeekOfYear

TKronos See also

Determines if a weeknumber is the last weeknumber of a year

function IsLastWeekOfYear(AYear, AWeek : Word) : Boolean

Description

Use IsLastWeekOfYear to find out if AWeek in AYear is the last weeknumber without changing the current year/week. Returns True if last week.

Relevant topics:
IsLastDayOfMonth

TKronos.MonthsInInterval

TKronos
Calculates the number of months in a specified interval.

function MonthsInInterval(Year1, Month1, Year2, Month2: Word) : Integer

Description
Use MonthsInInterval to get the number of months between Month1 in Year1 and Month2 in Year2. If Year1/Month1 is greater than Year2/Month2 the function will return a negative number else 0 or a positiv number.

TKronos.WeeksInInterval

TKronos
Calculates the number of weeks in a specified interval.

function WeeksInInterval(Year1, Week1, Year2, Week2: Word) : Integer

Description

Use WeeksInInterval to get the number of weeks between Week1 in Year1 and Week2 in Year2. If Year1/Week1 is greater than Year2/Week2 the function will return a negative number else 0 or a positiv number.

TKronos.DaysInInterval

TKronos See also

Calculates the number of days in a specified interval

function DaysInInterval(Year1, Month1, Monthday1,
Year2, Month2, Monthday2 : Word; WorkdaysOnly : Boolean) : Integer

Description

Use DaysInInterval to get the number of days between Monthday1 in Month1/Year1 and Monthday2 in Month2/Year2. If date 1 is greater than date 2 the function will return a negative number else 0 or a positive number.

If you set the WorkdaysOnly parameter to True, holidays are not counted. Assume the weekday of date 1 is Friday. Saturday and Sunday are week holidays. Date 2 is the following Monday. If WorkdaysOnly the function will return 1 (Saturday and Sunday are skipped) else 3.

Note: Setting WorkdaysOnly to True may slow down performance (notably with big intervals) as each day in the interval has to be examined.

Relevant topics
DaysInIntervalDt

TKronos.DaysInIntervalDt

TKronos

Calculates the number of days in a specified interval

```
function DaysInIntervalDt(ADate1, ADate2 : TDateTime; WorkdaysOnly : Boolean) : Integer
```

Description

The same function as DaysInInterval, except you pass TDateTime parameters instead of year, month and monthday parameteres.

TKronos.DaynumberByTypeName

TKronos See also Example

Returns the year based daynumber that results from a successful search for a daytype name in a specified year.

function DaynumberByTypeName(AYear : Word; DaytypeName : String) : Word

Description

Use DaynumberByTypeName to retrieve the year based daynumber of a date that is registered with <DaytypeName>. If no match the function returns 0.

DaynymberByTypeName example

```
if DaynumberByTypeName(2000, ‘Cristmas Day’) = 0 then
        ShowMessage(‘No presents for those who misspell Christmas!’);
```

Relevant topics:
DaynumberByTypeId

TKronos.DaynumberByTypeId

TKronos See also Example

Returns the year based daynumber that results from a successful search for a daytype id in a specified year.

function DaynumberByTypeId(AYear : Word; ADaytypeID : Word) : Word

Description

Use DaynumberByTypeId to retrieve the year based daynumber of a date that is registered with <ADaytypeId>. If no match the function returns 0.

DaynumberByTypeId example

```
if DaynumberByTypeId(2000, chChristmasDay) = 0 then
        ShowMessage('No Christmas this year!');
```

Relevant topics:
DaynumberByTypeName

TKronos.DateByDayOffset

TKronos See also

Returns the year and the year based daynumber that result from counting a specified number of days from the current date.

procedure DateByDayOffset(var AYear : Word; var ADaynumber : Word; OffsetValue : Integer; SkipHolidays : Boolean);

Description

Use DateByDayOffset to retrieve the year and the year based daynumber of the date that is positioned <OffsetValue> days from the date that currently is in focus. Use a negative offset value to count backwards, a positiv value to count forwards.

If you set the SkipHolidays parameter to True, holidays are not counted. Assume the weekday of current date is Friday. Offsetvalue is 3. Saturday and Sunday are week holidays. If SkipHolidays the procedure will return ADaynumber as the current daynumber + 5 (Saturday and Sunday are skipped) else daynumber + 3.

Note: Setting SkipHolidays to True may slow down performance (notably with big offset values) as each and every day in the interval has to be examined.

Relevant topics:
DateByWeekOffset
DateByMonthOffset

TKronos.DateByWeekOffset

TKronos See also

Returns the year and the year based daynumber that result from counting a specified number of weeks from the current date.

procedure DateByWeekOffset(var AYear : Word; var ADaynumber : Word; OffsetValue : Integer);

Description

Use DateByWeekOffset to retrieve the year and the year based daynumber of the date that is positioned <OffsetValue> weeks from the date that currently is in focus. Use a negative offset value to count backwards, a positiv value to count forwards.

Calling DateByWeekOffset is the same as calling the DateByDayOffset procedure with the OffsetValue parameter set to number of offset weeks * 7.

Relevant topics:
DateByDayOffset
DateByMonthOffset

TKronos.DateByMonthOffset

TKronos See also Example

Returns the year and the year based daynumber that result from counting a specified number of months from the current date.

procedure DateByMonthOffset(var AYear : Word; var ADaynumber : Word; OffsetValue : Integer);

Description

Use DateByMonthOffset to retrieve the year and the year based daynumber of the date that is positioned <OffsetValue> months from the date that currently is in focus. Use a negative offset value to count backwards, a positiv value to count forwards.

When calculating the daynumber the current monthday will be preserved if possible.

DateByMonthOffset example

Assume the current date is January 1. 2000:
DateByMonthOffset(AYear, ADaynumber, 1);
Result:
AYear = 2000
ADaynumber = 32 (February 1.)

Assume the current date is January 31. 2000:
DateByMonthOffset(AYear, ADaynumber, 1);
Result:
AYear = 2000
Daynumber = 60 (Februar 29. Cannot preserve monthday).

Relevant topics:
DateByDayOffset
DateByWeekOffset

TKronos.IsToday

TKronos See also

Checks to see if the date that is currently in focus is the date of today.

```
function IsToday(var AYear, ADaynumber : Word) : Boolean
```

Description

Use IsToday to determine if the current date is the date of today or to obtain the year and the year based daynumber of today. Returns True if AYear and ADaynumber match the current date.

Relevant topics:
IsTomorrow
IsYesterday

TKronos.IsTomorrow

TKronos See also

Checks to see if the date that is currently in focus is the date of tomorrow.

function IsTomorrow(var AYear, ADaynumber : Word) : Boolean

Description

Use IsToMorrow to determine if the current date is the date of tomorrow or to obtain the year and the year based daynumber of tomorrow. Returns True if AYear and ADaynumber match the current date.

Relevant topics:
IsToday
IsYesterday

TKronos.IsYesterday

TKronos See also

Checks to see if the date that is currently in focus is the date of yesterday.

function IsYesterday(var AYear, ADayNumber : Word) : Boolean

Description

Use IsYesterday to determine if the current date is the date of yesterday or to obtain the year and the year based daynumber of tomorrow. Returns True if AYear and ADaynumber match the current date.

Relevant topics:
IsToday
IsTomorrow

TKronos.IsThisWeek

TKronos See also

Checks to see if the week that is currently in focus is the week that contains the date of today.

function IsThisWeek(var AYear, AWeeknumber : Word) : Boolean

Description

Use IsThisWeek to determine if the current week is the week of today or to obtain the year and the weeknumber of today. Returns True if AYear and AWeeknumber match the values of the current date.

Relevant topics:
IsNextWeek
IsLastWeek

TKronos.IsNextWeek

TKronos See also

Checks to see if the week that is currently in focus is the week following the week of today.

```
function IsNextWeek(var AYear, AWeeknumber : Word) : Boolean
```

Description

Use IsNextWeek to determine if the current week is next week or to obtain the year and the weeknumber of next week. Returns True if AYear and AWeeknumber match the values of the current date.

Relevant topics:
IsThisWeek
IsLastWeek

TKronos.IsLastWeek

TKronos See also

Checks to see if the week that is currently in focus is the week previous to the week of today.

function IsLastWeek(var AYear, AWeeknumber : Word) : Boolean

Description

Use IsLastWeek to determine if the current week is last week or to obtain the year and the weeknumber og last week. Returns True if AYear and AWeeknumber match the values of the current date.

Relevant topics:
IsThisWeek
IsNextWeek

TKronos.IsThisMonth

TKronos See also

Checks to see if the month that is currently in focus is the month that contains the date of today.

```
function IsThisMonth(var AYear, AMonthnumber : Word) : Boolean
```

Description

Use IsThisMonth to determine if the current month is the month of today or to obtian the year and the monthnumber of today. Returns True if AYear and AMonthnumber match the values of the current date.

Relevant topics:
IsNextMonth
IsLastMonth

TKronos.IsNextMonth

TKronos See also

Checks to see if the month that is currently in focus is the month following the month of today.

```
function IsNextMonth(var AYear, AMonthnumber : Word) : Boolean
```

Description

Use IsNextMonth to determine if the current month is next month or to obtain the year and the monthnumber of next month. Returns True if AYear and AMonthnumber match the values of the current date.

Relevant topics:
IsThisMonth
IsLastMonth

TKronos.IsLastMonth

TKronos See also

Checks to see if the month that is currently in focus is the month pervious to the month of today.

```
function IsLastMonth(var AYear, AMonthnumber : Word) : Boolean
```

Description

Use IsLastMonth to determine if the current month is last month or to obtain the year and the monthnumber of last month. Returns True if AYear and AMonthnumber match the values of the current date.

Relevant topics:
IsThisMonth
IsNextMonth

TKronos.IsThisYear

TKronos See also

Checks to see if the year that is currently in focus is the year that contains the date of today.

```
function IsThisYear(var AYear : Word) : Boolean
```

Description

Use IsThisYear to determine if the current year is the year of today or to obtain the year of today. Returns True if AYear matches the year of the current date.

Relevant topics:
IsNextYear
IsLastYear

TKronos.IsNextYear

TKronos See also

Checks to see if the year that is currently in focus is the year following the year of today.

```
function IsNextYear(var AYear : Word) : Boolean
```

Description

Use IsNextYear to determine if the current year is next year or to obtain next year. Returns True if AYear matches the year of the current date.

Relevant topics:
IsThisYear
IsLastYear

TKronos.IsLastYear

TKronos See also

Checks to see if the year that is currently in focus is the year previous to the year of today.

```
function IsLastYear(var AYear : Word) : Boolean
```

Description

Use IsLastYear to determine if the current year is last year or to obtain last year. Returns True if AYear matches the year of the current date.

Relevant topics:
IsThisYear
IsNextYear

TKronos.GetNextDaytype

TKronos Example

Retrieves a daytype object from the daytype list.

function GetNextDaytype(var NextIndex : Word) : TDaytype

Description

GetNextDaytype provides a way to iterate over the daytype list. The NextIndex paramter represents the position in the list from where the next daytype is to be retrieved. The function increments this value for every time a daytype is returned.

This function is useful if you implement your own save procedure instead of or in addition to the SaveToFile procedure.

GetNextDaytype will return nil if NextIndex is outside the list boundaries. Note that the first index is 1, not zero.

The daytype list is sorted by identifier. Start with the value of FirstUserID to skip the predefined daytypes.

GetNextDaytype example

This example shows a skeleton procedure for saving a descendent object of TDaytype, here named TSpecialDaytype. The TSpecialDaytype is expanded with two extra fields, F1, and F2. The type TSpecialDef is declared to hold the daytype definiton:

```
Type
     TSpecialDef = record
          DaytypeDef : TDaytypeDef;
          F1, F2 : Integer;
     end;

procedure SaveSpecial;
   var
       i : integer;
       DT : TDaytype;
       DD : TDaytypeDef;
       SD : TSpecialDef;
       Index : Word;
       SpeciaDefs : File of TSpecialDef;
begin
        AssignFile(SpecialDefs,'Special.day');
        Rewrite(SpecialDefs);
        Index := FirstUserId;
        DT := GetNextDaytype(Index);
        while DT <> nil do
        begin
                DD := GetDaytypeDef(DT.Id, '');
            if DT is TSpecialDaytype then
                with DT as TSpecialDaytype do
                begin
                         SD.DaytypeDef := DD;
                         SD.F1 := F1;
                         SD.F2 := F2;
                         Write(SpecialDefs, SD);
                end;
                DT := GetNextDaytype(Index);
         end;
         CloseFile(SpecialDefs);
end;
```

TKronos.GotoDate

TKronos See also

Changes the current date to a date specified.

procedure GotoDate(AYear, AMonth, AMonthday : Word)

Description

Use GotoDate to change the current date to AYear, AMonth, AMonthday. This is similar to set the time uit properties directly:

```
BeginChange;
try
   Year := AYear;
   Month := AMonth;
   Monthday := AMonthday;
finally
   EndChange;
end;
```

Relevant topics
GotoDateDt
GotoDateDn

TKronos.GotoDateDt

TKronos See also

Changes the current date to a date specified.

```
procedure GotoDate(ADate : TDateTime)
```

Description

Use GotoDateDt to change the current date to ADate. This is the same procedure as GotoDate, except you pass a single TDateTime parameters instead of a year, month and monthday parameter.

Relevant topics:
GotoDate
GotoDateDn

TKronos.GotoDateDn

TKronos See also

Changes the current date to a date specified.

procedure GotoDateDn(AYear, ADaynumber : Word)

Description

Use GotoDateDn to change the current date to AYear, ADaynumber. This is the same procedure as GotoDate, except you pass a daynumber as parameter instead of month and monthday.

Relevant topics
GotoDate
GotoDateDt

TKronos.GotoToday

TKronos See also

Changes the current to the date of today.

procedure GotoToday

Description

Use GotoToday to change the current to the date of today

Relevant topics:
GotoTomorrow
GotoYesterday

TKronos.GotoYesterday

TKronos See also

Changes the current date to the date of yesterday.

procedure GotoYesterday

Description

Use GotoYesterday to change the current date to the date of yesterday.

Relevant topics:
GotoTomorrow
GotoToday

TKronos.GotoTomorrow

TKronos See also

Changes the current date to the date of tomorrow

procedure GotoTomorrow

Description

Use GotoTomorrow to change the current date to the date of tomorrow.

Relevant topics
<u>GotoYesterday</u>
<u>GotoToday</u>

TKronos.GotoThisWeek

TKronos See also

Changes the current week to the week that contains today .

procedure GotoThisWeek

Description

Use GotoThisWeek to change the current week to the week that contains today. The current weekday will not change.

Relevant topics:
GotoNextWeek
GotoLastWeek

TKronos.GotoNextWeek

TKronos See also

Changes the current week to the week following the one that contains today .

procedure GotoNextWeek

Description

Use GotoNextWeek to change the current week to the week following the one that contains today. The current weekday will not change.

Relevant topics:
GotoThisWeek
GotoLastWeek

TKronos.GotoLastWeek

TKronos See also

Changes the current week to the week previous the one that contains today .

procedure GotoLastWeek

Description

Use GotoLastWeek to change the current week to the week previous to the week one that contains today. The current weekday will not change.

Relevant topics:
GotoThisWeek
GotoNextWeek

TKronos.GotoThisMonth

TKronos See also

Changes the current month to the month that contains today .

procedure GotoThisMonth

Description

Use GotoThisMonth to change the current month to the month that contains today. The current monthday will not change, except if it does not fit the target month. In that case the monthday is set to the last day of the target month.

Relevant topics:
GotoNextMonth
GotoLastMonth

TKronos.GotoNextMonth

TKronos See also

Changes the current month to the month following the one that contains today .

procedure GotoNextMonth

Description

Use GotoNextMonth to change the current month to the month following the one that contains today. The current monthday will not change, except if it does not fit the target month. In that case the monthday is set to the last day of the target month.

Relevant topics:
GotoThisMonth
GotoLastMonth

TKronos.GotoLastMonth

TKronos See also

Changes the current month to the month previous the one that contains today .

procedure GotoLastMonth

Description

Use GotoLastMonth to change the current month to the month pervious to the one that contains today. The current monthday will not change, except if it does not fit the target month. In that case the monthday is set to the last day of the target month.

Relevant topics:
GotoThisMonth
GotoNextMonth

TKronos.GotoDaytype

TKronos
Changes the current date to a date that results from a successful search for a daytype in a specified year.

```
procedure GotoDaytype(AYear : Word; AnId : Word; ADaytypeName : String)
```

Description
Use GotoDaytype to change the current date to a date that matches the criterias in AYear and AnId/ADaytypeName. If no match the Exception EKronosError 'Daytype not found' is raised and the current date is not changed.

AnId/ADaytypeName is mutually exclusive. To search for an ID set ADaytypeName to an empty string. To serach for a name, set AnId to 0. If both AnId and ADaytypeName have values, the id value is the preferred key.

TKronos.GotoOffsetDay

TKronos See also

Changes the current date to a date that is <OffsetValue> days from the current date.

procedure GoToOffsetDay(OffsetValue : Integer; SkipHolidays : Boolean)

Description

Use GotoOffsetDay to change the current date to a date that is <OffsetValue> days from the current date. Negative values move backwards, positive forwards.

If you set the SkipHolidays parameter to True, holidays are not counted. Assume the current weekday is Friday. Saturday and Sunday are week holidays . GotoOffsetDay(1, False) will move to Saturday, while GotoOffsetDay(1, True) will move to Monday (Saturday and Sunday are skipped).

Note: Setting the SkipHolidays parameter to True may slow down performace (notably with big offset values) as each and every day in the offset interval has to be examined.

Relevant topics:
GotoOffsetWeek
GotoOffsetMonth

TKronos.GotoOffsetWeek

TKronos See also

Changes the current date to a date that is <OffsetValue> weeks from the current week.

procedure GoToOffsetWeek(OffsetValue : Integer)

Description

Use GotoOffsetWeek to change the current date to a date that is <OffsetValue> weeks from the current week. Negative values move backwards, positive forwards. The current weekday is not changed.

Relevant topics
GotoOffsetWeek
GotoOffsetMonth

TKronos.GotoOffsetMonth

TKronos See also

Changes the current date to a date that is <OffsetValue> months from the current month.

procedure GoToOffsetMonth(OffsetValue : Integer)

Description

Use GotoOffsetMonth to change the current date to a date that is <OffsetValue> months from the current month. Negative values move backwards, positive forwards. The current monthday will not change, except if it does not fit the target month. In that case the monthday is set to the last day of the target month.

Relevant topics:
GotoOffsetWeek
GotoOffsetDay

TKronos.DOWToWeekday

TKronos See also

Converts a day of week number to a TWeekday value.

function DOWtoWeekday(ADayOfWeekNumber : Word) : TWeekday

Description

Use DOWtoWeekday to obtain the corresponding Weekday. The result is calculated in connection with the value of the FirstWeekday property.

Relevant topics
DOWToDayNameIndex

TKronos.DOWToDayNameIndex

TKronos See also

Converts a day of week number to an number that can be used to access the Dayname array.

function DOWtoDayNameIndex(ADayOfWeekNumber:Word) : Word

Description

You might find it convinient to use the dayname array, even if you can extract daynames from the DateExt property. Use DOWtoWeekday to obtain the index to use with the array. The result is calculated in connection with the value of the FirstWeekday property.

Relevant topics
DOWToWeekday

TKronos.CDToDateTime

TKronos
Converts the date that currently is in focus to a TDateTime value.

function CDtoDateTime : TDateTime

Description
Merely a short hand way to DT := EncodeDate(Year, Month, Monthday);

TKronos.GetMIDayCell

TKronos See also Example

Returns the coordinates to a cell in a TMonthImage-table that contains a specified year based daynumber.

procedure GetMIDayCell(ADaynumber : Word; var ARow, ACol : Longint)

Description

Use this procedure to spot a cell that contains ADaynumber. This is especially useful when working with onscreen calendars. If ADaynumber is not found ARow and ACol is set to 0. Note that you cannot search for days that do not belong to the month.

GetMIDayCell example

Assume that your TKronos component initializes to today. You want to find the today-cell on the onscreen grid:

```
GetMIDayCell(Daynumber, ARow, ACol);
MyGrid.Cells[ACol, ARow] := ‘This is today.’;
```

Relevant topics:
GetFirstMIDayCell
GetMIWeekRow
TMonthImage

TKronos.GetMIWeekRow

TKronos See also

Returns the rownumber in a TMonthImage-table that contains a specified weeknumber.

function GetMIWeekRow(AWeekNumber : Word) : Word

Description

Use this procedure to spot a row that contains AWeekNumber in a TMonthImage table. If AWeekNumber is not found the function returns 0. Note that you cannot search for weeks that not at all belong to the month.

Relevant topics:
GetMiDayCell

TKronos.GetFirstMIDaycell

TKronos See also

Returns the coordinates to a cell in a TMonthImage-table that contains the first day of the month.

procedure GetFirstMIDayCell(var ARow, ACol : Longint)

Description

Use this procedure to spot the cell that starts the month.

Relevant topics:
GetLastMIDayCell
GetMiDayCell

TKronos.GetLastMIDaycell

TKronos See also

Returns the coordinates to a cell in a TMonthImage-table that contains the last day of the month.

procedure GetLastMIDayCell(var ARow, ACol : Longint)

Description

Use this procedure to spot the cell that ends the month.

Relevant topics:
GetFirstMIDayCell
GetMiDayCell

TKronos.DisableEvents

TKronos Example

Turns on/off event triggering for a TKronos component

procedure DisableEvents(Disable : Boolean)

Description

Use this procedure to protect against unwanted triggering of events during temporary changing of the current date. Passing True in the Disable parameter turns triggering off, False turns it on.

Note

DisableEvents only has effect for the OnChangeXXX events.

DisableEvents example

Although TKronos offers many ways to retrieve information from dates outside the current date, there might be situations when you temporarily want to switch to another date to perform some investigation. Then you possibly don’t wish any eventhandlers to fire. Assume you want to look at the daytypes for a certain date:

```
DisableEvents(True) //Turn off event triggering
SaveCD // Save the current date
GotoDate (2000, 10, 5) //Make temporarily antother date the current date
for i := 1 to DaytypeCount do
begin
        //Perform some action
        :
        :
end;
RestoreCD // Back to the real current date
DisableEvents(False) // Ready to handle events again
```

TKronos.SaveCD

TKronos See also

Saves the current date.

procedure SaveCD

Description

Use this procedure to save the current date for later to restore it with RestoreCD. The current date is saved in an internal variable that will be overwritten each time you make the call.

Relevant topics:
RestoreCD

TKronos.RestoreCD

TKronos See also

Restores the date previously saved by SaveCD

procedure RestoreCD

Description

Use this procedure to restore the current date that was previously saved with SaveCD. When RestoreCD is called the internal variable that holds the saved date is invalidated. Any subsequent call to RestoreCD will have no effect unless you first call the SaveCD procedure.

Relevant topics:
SaveCD

TKronos.BeginChange

TKronos See also Example

Starts a change transaction.

procedure BeginChange

Description

Use BeginChange in connection with EndChange to safely alter a date that involves setting more than one of the time unit properties. The change transaction mechanism enacpsulates several property settings into one logical operation. If one of the individual settings fails, the change is canceled and the original date is restored.

While a transaction is active there will be no event triggering. Event-handling is postponed until you call EndChange. Then only one of each event-kinds involved is triggered. If you don't use the transaction mechanism, one and the same event might be triggered several times.

Whenever possible use one of the Goto-procedures to change the date. The Goto-procedures handle change transaction internally, so you don't have to think about writing protected blocks of code.

A call to BeginChange when a transaction is active has no effects.

BeginChange example

Assume you want to put focus on the first weekday of the last week in a certain month. There are no Goto-procedures that perform this kind of navigation, so you must set the relevant time unit properties directly. You should do it like this.

Assume current date to be January 1.

```
BeginChange;
try
      Month := 2; // Go to february
      Week := MonthExt.LastWeek; // Move to last week
      Weekday := FirstWeekday
     // Move to first day of week
finally
      EndChange;
{If an exception were raised, say you attempted to set week to 55, the EndChange statement would have restored the current date to the original January 1.}
end;
```

Relevant topics
EndChange
General guidelines
GotoDate

TKronos.EndChange

TKronos See also

Ends a change transaction.

procedure EndChange

Description

Use EndChange to end a running change transaction. EndChange processes any events that might have occured during the change process. If an error was encountered during the process the orginal current date is restored and no events are triggered.

Calling EndChange when no transaction is active has no effect.

Relevant topics:
BeginChange
General guidelines

TKronos.UpdateInfo

TKronos See also Example

Manually updates the MonthExt and DateExt properties to reflect changes in the daytype list.

procedure UpdateInfo

Description

Normally UpdateInfo is called internally whenever needed. There are however a few exceptions.
After calls to AddDaytype and SpecifyStandardDay you should do a manual update to be sure that the current date reflects the changes. You only need to call UpdateInfo once in a configuiring sequence, not after each call to the mentioned procedures.

UpdateInfo example

```
var
      DaytypeDef : TDaytypeDef;
:
:
with DaytypeDef do
begin
               AName := ’10 days left to Easter’
               ADate = 0;
               AReldayType = chEasterSunday;
               AnOffset = -10;
               AFirstShowUp = 1;
               ALastShowUp := 9999;
               AShowUpFrequency = 1;
               AHoliday := False;
               AChurchday := False;
               AFlagDay := False;
               AUserCalc := False;
               ATag := 0;
               AddDaytype(TDaytype.Create(DaytypeDef));

               AName = ‘9 days left to Easter’;
               AnOffset := -9;
               AddDaytype(TDaytype.Create(DaytypeDef));

               AName := '’8 days left to Easter’;
               AnOffset := -8;
               AddDaytype(TDaytype.Create(DaytypeDef));
    end;
UpdateInfo;
:
:
```

Relevant topics:
Rechange

TKronos.Rechange

TKronos
Triggers all OnChange - eventhandlers.

procedure Rechange

Description
Use Rechange to force all the OnChange... eventhandlers to fire: OnChangeYear, OnChangeMonth, OnChangeMonthNumber, OnChangeWeek, OnChangeWeekNumber, OnChangeDate, OnChangeMonthDay and OnChangeWeekday.

This procedure is useful to when you want initial OnChange... events to take place after TKronos is loaded.

TKronos.LoadFromFile

TKronos See also

Loads a calendar profile from a disk file.

procedure LoadFromFile(AFileName : String; LoadAll : Boolean)

Description

Use LoadFromFile to load a set of daytypes and daynames/monthnames from an external disk file and make them the current daytype definitions. A daytype file may store any of the three daytype categories: Churchdays, common days and userdefined types.

To load the definition file from a directory different from the current directory qualify AFilename with a full directory path.

To use the LoadAll parameter see the paragraph “Predefined usertypes” below.

File format

A file that stores daytypes must be a textfile following these conventions:

Daynames

You define the daynames by means of a header section enclosed in brackets followed by a dayname list. The text of the header section must be ‘Daynames’. The list consists of keywords and values for each dayname. The keydwords are:

Sun
Mon
Tue
Wed
Thu
Fri
Sat

This is how a Daynames section could look like:

```
[Daynames]
Sun=Sunday
Mon=Monday
Tue =Tuesday
etc.
```

If you ommit the Daynames section the daynames in the Daynames array will be unchanged.

Monthnames

You define the monthnames by means of a header section enclosed in brackets followed by a monthname list. The text of the header section must be ‘Monthnames’. The list consists of keywords and values for each monthname. The keydwords are:

Jan
Feb
Mar
Apr
May
Jun
Jul
Aug

Sep
Oct
Nov
Dec

If you ommit the Monthnames section the monthnames in the Monthnames array will be unchanged.

This is how the Monthnames section could look like:

```
[Monthnames]
Jan=January
Feb=February
Mar =March
etc.
```

Week specifications
You define week specifications by means of a header section enclosed in brackets followed by a specification list. The text of the header section must be ‘Week’. The list consists of keywords and values for each specification. The keydwords are:

WeekHolidays
FirstWeekday

You specify the week holidays as numeric string where each digit corresponds to a weekday holiday. 0 is Sunday, 1 is Monday, etc. You specify FirstWeekday as a single digit.

This is how the week section could look like:

```
[Week]
WeekHolidays=067
;Sunday; Friday and Saturday
FirstWeekday= 1
;Monday
```

To make the calendar contain no week holidays, simply set the WeekHolidays value to blank.

If you ommit the Week section, WeekHoldidays will be set to Sunday and Saturday, FirstWeekday to Sunday.

Predefined church- and common days
Every predefined churchday and common day included in the file must have an entry consisting of a header section enclosed in brackets and a key section describing the fields and the field values. The text of the header section identifies the daytype by a daytype constant prefix and a daytype constant value (to see a listing of the daytype constants go to the Daytype constants topic).

The key-list, that follows immidiately after the header, describes the fields by
<Field name> = <Value>.
For Boolean values use 1 for True and 0 for False.

The fieldnames for churchdays and common days are:
Name
Holiday
Flagday

This example defines the Easter Sunday daytype:

```
[ch16]
Name=Easter Sunday
Holiday=1
Flagday=1
```

Ch is the constant prefix for churchdays. 16 is the constant value for Easter Sunday.

Predefined usertypes
If you work with a descendent of TKronos, and there are daytypes added through the SetCountrySpecifics method, these daytypes are part of the basic calendar profile. You may include those daytypes in the file to redefine names and other attributes. However, you have not the option to delete them from the caleandar, so ommitting them i the file will have no destructive effect.

You decribe theese predefined usertypes using “cs” as the daytype constant prefix. The constant value must correspond to the daytype you wish to redefine. These values should be documented by the component writer, but you can also see them by dumping the calendar definition to a file with the SaveToFile procedure.

Note! You can prevent the predefined usertypes from being modified, by setting the LoadAll parameter to False. This have the effect that daytypes with the cs header prefix are ignored. If no predefined usertypes exists in the file LoadAll has no effect.

Note also that you cannot change all the fields. Setting new values for the fields Date, Reldaytype, Offset, FirstShow, LastShow and ShowUpFreq will go unnoticed.

The fieldnames and values to use whith predefined usertypes are the same as for regular user defined daytypes. See the next paragraph.

This is how a predefined usertype could look like:

```
[cs27]
Name=Independence Day
Date=704
;Cannot be changed
FirstShow=1776
;Cannot be changed
Flagday=1
```

User defined daytypes
The list of userdefined daytypes is made up of the daytypes you want to include in addition to the predefined types. Every entry in the list must have “ud” as the daytype constant prefix. Start with 1 as the constant value and increment by 1. The field names to use with userdefined types are:

Name
Date
RelDaytype
Offset
FirstShow
LastShow
ShowUpFreq
Holiday
Flagday
Churchday
Calc
Tag

This is how a userdefined type could look like:

```
[ud1]
Name=My day
Date=610
;June 10
FirstShow=1990
LastShow=2000
Flagday=1
```

Default values
If you ommit fields from the definiton, standard values are used. The standard values are:

```
Name = ‘’
Date = 0
RelDaytype = 0
Offset = 0
FirstShow = 1
LastShow = 9999
ShowUpFreq = 1
Holiday = False
Flagday = False
Churchday = False
Calc = False
Tag = 0
```

Note
Before loading a calendar profile from disk the current daytype definitions, except the predefined, will be deleted.

The sequence in which the fieldnames are listed is not significant. If you ommit (or misspell) fieldnames, the standard values will be used.

If any of the section headers in the file contains illegal constant prefixes or values TKronos raises the exception EKronosError ‘Illegal section (<section>) in inputfile’ and the load process is terminated. There is no checking for logical errors (like you specify FirstShow to be later than LastShow).

Tip
To create a skelton daytype file to work with save the current definitions with SaveToFile.

Relevant topics
SaveToFile
SpecifyStandardDay
AddDaytype
Using daytypes
SetCountrySpecifics

TKronos.SaveToFile

TKronos
Saves the current daytype definitions to an external disk file.

procedure SaveToFile(AFilename : String);

Description
Use SaveTo file to save the current daytype definitions to an external diskfile. Alle the daytype categories are saved, that is chuchdays, common days and user defined types if any.

If a file with the same name as AFilename already exists it is overwritten.

To save the definitions to a directory different from the current directory qualify AFilename with a full directory path.

For a complete discussion of daytype files see LoadFromFile.

TKronos.OnChangeDate

TKronos
OnChangeDate occurs every time the current date changes.

property OnChangeDate: TNotifyEvent;

Description
Use the OnChangeDate event to write code that responds to the change-date event. In your code you can safely read the time unit and Ext properties as the event is not triggered until all updates are performed.

Caution
If you within the OnChange event handler change the current date you must first disable event triggering with DisableEvents else the event handler will recursively call itself.

TKronos.OnChangeMonth

TKronos See also Example

OnChangeMonth occurs every time the current month changes. That is whenever the monthnumber *or* the year changes.

property OnChangeMonth: TNotifyEvent

Description
Use the OnChangeMonth event to write code that responds to the change-month event. In your code you can safely read the time unit and Ext properties as the event is not triggered until all updates are performed.

Caution
If you within the OnChange event handler write code that might change the current month you must first disable event triggering with DisableEvents else the event handler will recursively call itself.

OnChangeMonth example

```
procedure MyForm.AlterMonth;
begin
 //Asume current month = 1. Both statements trigger the OnChangeMonth event
        with Kronos1 do
        begin
                GotoDate (1999,2,1); // Monthnumber changes
                Year := Year + 1; // Monthnumber is the same, but the year changes.
        end;
end;

procedure MyForm.Kronos1ChangeMonth(Sender : TObject);
begin
        with Kronos1 do
        if YearExt.LeapYear and (Month = 2) then
             ShowMessage(MonthExt.Monthname + ‘ has 29 days this year.’)
             // Relevant after Year := Year + 1
        else if (Month = 2) then
             // Relevant after GotoDate(1999,2,1)
             ShowMessage(MonthExt.Monthname + ‘ has 28 days this year.’)
end;
```

Relevant topics:
OnChangeMonthNumber

TKronos.OnChangeMonthNumber

TKronos See also

OnChangeMonthNumber occurs every time the current monthnumber changes.

property OnChangeMonthNumber: TNotifyEvent

Description

Use the OnChangeMonthNumber event to write code that responds to the change-monthnumber event. In your code you can safely read the time unit and Ext properties as the event is not triggered until all updates are performed.

Caution

If you within the OnChange event handler write code that might change the current monthnumber you must first disable event triggering with DisableEvents else the event handler will recursively call itself.

Relevant topics:
OnChangeMonth

TKronos.OnChangeMonthday

TKronos
OnChangeMonthday occurs every time the current monthday changes.

property OnChangeMonthday: TNotifyEvent;

Description
Use the OnChangeMonthday event to write code that responds to the change-monthday event. In your code you can safely read the time unit and Ext properties as the event is not triggered until all updates are performed.

Caution
If you within the OnChange event handler write code that might change the current monthday you must first disable event triggering with DisableEvents else the event handler will recursively call itself.

TKronos.OnChangeWeek

TKronos See also

OnChangeWeek occurs every time the current week changes. That is whenever the weeknumber *or* the year changes.

property OnChangeWeek: TNotifyEvent;

Description

Use the OnChangeWeek event to write code that responds to the change-week event. In your code you can safely read the time unit and Ext properties as the event is not triggered until all updates are performed.

Caution

If you within the OnChange event handler write code that might change the current week you must first disable event triggering with DisableEvents else the event handler will recursively call itself.

Relevant topics:
OnChangeWeeknumber

TKronos.OnChangeWeeknumber

TKronos See also

OnChangeWeekNumber occurs every time the current weeknumber changes.

property OnChangeWeeknumber: TNotifyEvent

Description

Use the OnChangeWeeknumber event to write code that responds to the change-weeknumber event. In your code you can safely read the time unit and Ext properties as the event is not triggered until all updates are performed.

Caution

If you within the OnChange event handler write code that might change the current weeknumber you must first disable event triggering with DisableEvents else the event handler will recursively call itself.

Relevant topics:
OnChangeWeek

TKronos.OnChangeWeekday

TKronos
OnChangeWeekday occurs every time the current weekday changes.

property OnChangeWeekday: TNotifyEvent

Description
Use the OnChangeWeekday event to write code that responds to the change-weekday event. In your code you can safely read the time unit and Ext properties as the event is not triggered until all updates are performed.

Caution
If you within the OnChange event handler write code that might change the current weekday you must first disable event triggering with DisableEvents else the event handler will recursively call itself.

TKronos.OnChangeYear

TKronos
OnChangeYear occurs every time the current year changes.

property OnChangeYear: TNotifyEvent;

Description
Use the OnChangeYear event to write code that responds to the change-year event. In your code you can safely read the time unit and Ext properties as the event is not triggered until all updates are performed.

Caution
If you within the OnChange event handler write code that might change the current year you must first disable event triggering with DisableEvents else the event handler will recursively call itself.

TKronos.OnToday

TKronos
OnToday occurs every time the current date changes to today.

property OnToday: TNotifyEvent;

Description
Use the OnToday event to write code that responds to the today event. In your code you can safely read the time unit and Ext properties as the event is not triggered until all updates are performed.

Caution
If you within the OnChange event handler write code that might change the current date to today you must first disable event triggering with DisableEvents else the event handler will recursively call itself.

TKronos.OnCalcDaytype

TKronos See also Example

OnCalcDaype occurs every time a daytype with the UserCalc property set to True is to be evaluated. This happens when the current date changes or when date information is retrieved with the FetchDateExt function.

property OnCalcDaytype : TCalcDaytypeEvent;

TCalcDaytypeEvent = procedure(Sender : TObject; Daytype : TDaytype ;
ADateExt : TDateExt; IsCurrentDate : Boolean;
var Accept : Boolean) of Object;

Description

Use OnCalcDaytype to write code that responds to the CalcDaytype event. Instead of letting the show up pattern be controlled by hard coded attributes of the daytype itself (that is the Date and the RelDaytype fields), you can obtain great flexibility by writing your own algorithms. These might be simple or complex calculations - anything you need to direct the showups the way you want.

To make a user calculated daytype show up several times within a year, you only need to add one instance of it to the daytype list. Not so with a "static" daytype, which is bound to show up only once in a year.

For eventhandling to take place, besides setting the Daytype's UserCalc field to True, you also must set the AllowUserCalc property to True.

Parameters

Daytype	The daytype object that is to be evaluated. This might a TDaytype object or an object derived from TDaytype.
AdateExt	Detailed information about the date that you might attach to the daytype
IsCurrentDate	True if information held by ADateExt corresponds to the date that actually is in focus. If OnCalDaytype is triggered as a result from changing the current date, IsCurrentDate is allways True. If triggering comes from retrieving information without changing the current date, the value will be False. For instance: If the current date is March 1. 1999 and the user calls FetchDateExt(2000,1,1) then AdateExt will contain information about January 1. 2000 while the properties of TKronos itself will reflect March 1. 1999. The point of reading this parameter is that you will know the effect of calling TKronos methods from within the event handler. If you for instance call the IsToday function and IsCurrentDate is False, then you actually checks the state of the current date and not the date held in AdateExt.
Accept	Set to True if you decide to attach Daytype to the date held in ADateExt

How it works

Every time you change the current date or retrieve information about it, TKronos queries the daytype list to find daytypes that matches the show up criterias in the daytype definition. For instance, if the current date changes to January 1, TKronos spots the daytypes with the value 101 in the date field and binds them to the current date. However, if TKronos finds a daytype in the list with the UserCalc field set to True, rather than looking into the date fields, it asks you wether to create a bind. You send your answer with the Accept parameter.

When to use the OnCalcDaytype event
Use this feature only when needed. If your daytype does well with a “static” declaration, declare it as not user calculated. Such daytypes do not carry the overhead of the user calc model and run somewhat faster. If you decide upon user calculation then be aware of a possible slowdown as your code is called every time the current date changes. If your calculations are lengthy and complex, then you might expect a sigificant loss of speed.

Note!
You might, and possibly will, make calls to other TKronos procedures or functions from inside the OnCalcDaytype event handler. To avoid deadlock situations, where the handler endlessly triggers itself, TKronos protects the handler from beeing called while code executes. This has the effect that user calulated daytypes will not be processed by any methods as long as they are originated from the handler itself.

Caution
Although possible you should within the event handler avoid operations that change the current date. As this might lead to conflicting date transactions, TKronos will prevent any date trasanction from starting while OnCalcDaytype executes.

Relevant topics:
AllowUserCalc property
Using daytypes
Processing daytype classes

TKronos.OnLoadDaytype

TKronos		See also

OnLoadDaytype occurs every time when a daytype is about to be loaded from a standard TKronos calendar file using the LoadFromFile method.

property OnLoadDaytype: TLoadDaytypeEvent;

TLoadDaytypeEvent = procedure(Sender : TObject; DaytypeDef : TDaytypeDef; var LoadIt : Boolean) of Object;

Description

Use the OnLoadDaytype event to control loading of userdefined (not predefined) daytypes. This event i useful when working with descendents of the TDaytype object and you wish to implement your own loading procedure, but still want the basic daytype definition to be stored in the standard file.

Parameters

DaytypeDef	The basic definition of the daytype which is ready to be loaded.
LoadIt	Set to True (standard value) if you want the daytype to be created and added to the list as a standard TDaytype object. Set to False if you don’t want the LoadFromFile procedure to add it to the list, but wish to create it yourself.

Note

You cannot prevent predefined daytypes from beeing added to the daytype list, but you can force them never to show up by setting the HidePredefineds propertry to True.

Tip

To find out if DaytypeDef is the basic definition of a descandant of TDaytype, it might be an idea to use the tag field to classify it.

Relevant topics:
OnSaveDaytype
Processing daytype classes

TKronos.OnSaveDaytype

TKronos See also

OnSaveDaytype occurs every time a daytype is about to be saved to a standard TKronos calendar file using the SaveToFile method.

property OnSaveDaytype: TSaveDaytypeEvent;

TSaveDaytypeEvent = procedure(Sender : TObject; Daytype : TDaytype; var SaveIt : Boolean) of Object;

Description

Use the OnSaveDaytype event to control saving of userdefined (not predefined) daytypes. This event i useful when working with descendents of the TDaytype object and you wish to implement your own storing procedure, instead of or in addition to storing the basic daytype definition in a standard file.

Parameters

Daytype	The daytype object which is ready to be saved.
SaveIt	Set to True (standard value) if you want the daytype defintion to be stored in the standard file. Set to False if you don’t want the SaveToFile procedure to store it in the file..

Note

You cannot prevent predefined daytypes from beeing stored in the standard file, as the event is not triggered for predefined types.

Tip

If you store a descendent of TDaytype in a standard file, it might be an idea to use the tag field to classify it.

TKronos properties

Events   Methods

AllowUserCalc
DateExt
Daynumber
DaytypeCount
Daytypes
DefaultToPresentDay
FirstUserId
FirstWeekday
HidePredefineds
MaxYear
MinYear
Monthday
MonthExt
Month
Weekday
WeekExt
WeekHolidays
Week
YearExt
Year

TKronos methods

Properties Events

Daytype definiton methods

AddDaytype
ClearUserDaytypes
DeleteUserDaytype
UpdateDaytype
GetDaytypeDef
GetNextDaytype
SpecifyStandardDay
SetCountrySpecifics
LoadFromFile
SaveToFile
ExistsDaytype

Methods that retireve information about time units

FetchYearExt
FetchMonthExt
FetchWeekExt
FetchDateExt
FetchDateExtDt
FetchDateExtDn
FetchDaytpe
FetchYeartype
DaynumberByTypeName
DaynumberByTypeId
IsLeapYear
IsLastDayOfMonth
IsLastWeekOfYear

Methods that perform calculation

DaysInInterval
DaysInIntervalDt
WeeksInInterval
MonthsInInterval
DateByDayOffset
DateByWeekOffset
DateByMonthOffset

Methods that return the status of the current date

IsToday
IsTomorrow
IsYesterday
IsThisWeek
IsNextWeek
IsLastWeek
IsThisMonth
IsNextMonth
IsLastMonth
IsThisYear
IsNextYear
IsLastYear

Methods that changes the current date

GotoDate
GotoDateDt

GotoDateDn
GotoToday
GotoTomorrow
GotoYesterday
GotoThisWeek
GotoNextWeek
GotoLastWeek
GotoThisMonth
GotoNextMonth
GotoLastMonth
GotoDaytype
GoToOffsetDay
GoToOffsetWeek
GoToOffsetMonth

Converting methods
DOWtoWeekday
DOWtoDayNameIndex
CDtoDateTime

Methods operating on the MonthImage-table
GetMIDayCell
GetMIWeekRow
GetFirstMIDayCell
GetLastMIDayCell

Other methods
DisableEvents
SaveCD
RestoreCD
BeginChange
EndChange
Rechange

TKronos events

OnChangeYear
OnChangeMonth
OnChangeMonthNumber
OnChangeWeek
OnChangeWeekNumber
OnChangeMonthday
OnChangeWeekday
OnChangeDate
OnCalcDaytype
OnLoadDaytype
OnSaveDaytype
OnToday

TKronos - using daytypes

Most calendars are not just listings of months and days, they also imform about what happens during a year. National and religious events are most often printed on calendars, aditionally calendars can devote themselves to certain themes, like litterature for instance. On a such a calendar you will know about birth and death of writers and when their most famous books were published.

Predefined and user defined

The Daytype feature of TKronos makes it easy to keep track of such annual events. TKronos comes with several predefined daytypes that conform to the most common Christian churchdays and international notification days. Furthermore you can define new daytypes, as many as you like. You attach daytypes to dates - or more generally to years - in a one to many releationship. Daytypes with no date reference are called *yeartypes*. Yeartypes are allways user defined (see FetchYeartype and AddDaytype to learn how to create and retrieve yeartypes).

The standard daytypes have different status, and of course, different names in different countries. So to use them you have to adjust them to your environment. If you don't make any adjustments you will see English names by default. The status atrributes Holiday and Flagday are both set to False.

You may choose between two strategies when redefining or creating daytypes. If you want a stable and easily reusable calendar component you ought to derive a new component from TKronos. If you often change between different sets of daytypes, it might be an idea to maintain libraries of daytypes to load and unload at runtime. Of course you might ride both horses.

Adjusting TKronos by deriving a new component

When TKronos initializes it calls a protected procedure named SetCountrySpecifics. This does nothing at all, but is there for you to override the standard names and attributes of the predefined daytypes or add your own. (To see a listing of the predefined types go to the Daytype Constants topic).

In the SetCountrySpesifcs procedure call two other procedures:
SpecifyStandardDay to adjust a prefefined church or international notification day
AddDaytype to add a daytype of your own.

Yous must make one call for each daytype you process. Here is a code fragment:

```
procedure TKronosNor.SetCountrySpecifics;
var
    DaytypeDef : TDaytypeDef;
begin

   inherited SetCountrySpecifics;
   SpecifyStandardDay(chNewYearEve,'Nyttaarsaften', False, False);
   SpecifyStandardDay(chNewYearDay, 'Nyttaarsdag', True, True);
   SpecifyStandardDay(chShroveTuesday, 'Fetetirsdag', False, False);
   SpecifyStandardDay(chAshWednesday, 'Askeonsdag', False, False);
   SpecifyStandardDay(chPalmSunday, 'Palmesoendag', True, False);
   SpecifyStandardDay(chMaundyThursday, 'Skjaertorsdag', True, False);
   SpecifyStandardDay(chGoodFriday, 'Langfredag', True, False);
   SpecifyStandardDay(chEasterEve, 'Paaskeaften', False, False);
   SpecifyStandardDay(chEasterSunday, '1. paaskedag', True, True);
   SpecifyStandardDay(chEasterMonday, '2. paaskedag', True, False);
   :
   :
   SpecifyStandardDay(coMayDay,'1. mai',   True, True);
   SpecifyStandardDay(coUNDay, 'FN-dagen',   False, False);
```

```
    SpecifyStandardDay(coWomensDay, 'Kvinnedagen',
    False, False);

    with DaytypeDef do
    begin
            AName := 'Frigjoeringsdag'; //Liberation day
            ADate := 508;
            AReldayType := 0;
            AnOffset := 0;
            AFirstShowUp := 1945;
            ALastShowUp := 9999;
            AShowUpFrequency := 1;
            AHoliday := False;
            AChurchday := False;
            AFlagDay := True;
            AUserCalc := False;
            ATag := 0;
            AddDaytype(TDaytype.Create(DaytypeDef));

            AName := 'Grunnlovsdag'; // National day
            ADate := 517;
            AFirstShowUp := 1814;
            AHoliday := True;
            AddDaytype(TDaytype.Create(DaytypeDef));

            AName := 'Olsok'; // Local religious day
            ADate := 729;
            AFirstShowUp := 1000;
            AChurchday := True;
            AFlagDay := True;
            AddDaytype(TDaytype.Create(DaytypeDef));

            AName := 'Election year'; // Year type. Election every 4. year
            ADate := 0;
            AFirstShowUp := 1900;
            AddDaytype(TDaytype.Create(DaytypeDef));
   end;
end;
```

These are Norwegian daytypes, don’t mind the mysterious names.

Changing daynames and monthnames
Besides manipulating daytypes, you can also override the standard TKronos day- and monthnames. TKronos maintains two array variables, Daynames and Monthnames which are the sources of the names presented to you through the DateExt and MonthExt properties. When TKronos initializes the Delphi LongDaynames and LongMonthNames-arrays are copied into the corresponding TKronos arrays. The names are country spesific, so in most cases you don’t need to change them, but if you want you can do it. Simply fill in the names like this:

```
Daynames[1] := ‘Sunday’;
Daynames[2] := ‘Monday’
:
Monthnames[1] := ‘January’
:
```

Adjusting at runtime

If you don’t whish to derive a new component, you might obtain the same result by calling the two procedures upon creation of the form that contains TKronos. You can also save different daytype sets on disk and load them by calling the LoadFromFile procedure.

```
procedure MyForm.FormCreate;
begin
        //Call the above mentioned procedures or:
         KronosNor.LoadFromFile(‘c:\MyDir\Norway.kdt’, True);
        :
        :
end;
```

User calculated daytypes
A TKronos standard daytype permits only fixed date definitions or simple offset calculation. Such daytypes will only show up once i a year and mostly have their mission in a traditional calendaric scheeme. But probably you will sometimes need to mark up days in a more sophisticated manner. Possibly you’ll need to figure out an event by means of caculations far beyond the capabilities of a standard daytype. Through the OnCalcDaytype event TKronos provides a mechanism that puts you in total control over the show up pattern. Simply declare a daytype as user calculated, then every time the current date changes you are notified to deciede if this is the date for the daytype to show up.

This flexibility, may be in connection with derived daytype classes, render you a powerful tool to process almost any kind of chronological events you might think of.

Relevant topics:
Processing daytype classes

TKronos.Daynames array

The Daynames array stores the names of the weekdays.

Daynames : array[1..7] of String

Description

The daynames must start with Sunday, then Monday and so on.

Daynames is the sources of the names presented to you through the DateExt propertiy. When TKronos initializes the Delphi LongDaynames array is copied into the Daynames array. The names are by default country spesific, so in most cases you don't need to change them, but if you want you can do so. Simply fill in the names like this:

```
Daynames[1] := 'Sunday';
Daynames[2] := 'Monday'
```

By manipulating the Daynames array you can override the country dayname definitions of the user's machine.

Relevant topics:
Monthnames array
Using daytypes
LoadFromFile

TKronos.Monthnames array

TKronos See also

The Monthnames array stores the names of the months.

Monthnames : array[1..12] of String

Description

The monthnames must start with January, then February and so on.

Monthnames is the source of the monthnames presented to you through the MonthExt property. When TKronos initializes the Delphi LongMonthnames-array is copied into the Monthnames array. The names are by default country spesific, so in most cases you don’t need to change them, but if you want you can do so. Simply fill in the names like this:

```
Monthnames[1] := ‘January’;
Monthnames[2] := ‘February’
```

By manipulating the Monthnames array you can override the country monthname definitions of the user’s machine.

Relevant topics:
Daynames array
Using daytypes
LoadFromFile

TKronos - general guidelines

TKronos

The current date

TKronos offers three ways to define the current date (the date that currently is in focus).

The current date may bee defined as a comibination of
either
Year and Daynumber
or
Year, Month and Monthday
or
Year, Week and Weekday.

The properties Year, Month, Monthday, Week, Weekday and Daynumber ar referred to as *time unit properties*.

Changing the date

Altering one of the time unit properties will cause an imidiate cascading update of any other time unit properties affected. If you, for instace, change the weekday from Wednesday to Thursday then the daynumber and monthday, possibly also the month and year, will change too.

The rule is that TKronos avoids updates if it is not necessary. Say the current monthday is 31. Changing the month will not alter the monthday, unless it does not fit the month moved to. Moving from January to March will leave monthday 31 intact; moving to February would change it to 28 (or 29).

Extended information

The time unit properties let you see basic aspects of the current date. There are however other properties you can read to obtain a lot more information. These are the YearExt, MonthExt, WeekExt and DateExt-properties. These are referred to as *Ext properties*. As with the time unit properties the values of the Ext properties keep in pace with changes of the current date.

A chapter of itself is the Daytypes property. To learn about daytypes go to the topic Using daytypes.

Applying the correct sequence

When manipulating more than one time unit property to form a new date you should allways set the "topmost" property first. For instance, to change the date to a new year, a new month and a new monthday this is the recommended sequence:

```
Yaer := ANewYear; // Year first
Month := ANewMonth; // Month second
Monthday := ANewMonthday; // Monthday third
```

Change Transactions

As you will notice in the example above, one logical operation is broken down into three different tasks. If one of them fails the current date might be left invalid. Whenever you perform a date change by means of two or more time unit properties you should make it a change transaction:

```
BeginChange;
try
      // Change properties as needed
finally
     EndChange;
end;
```

This ensures that if any error occurs the current date will be left intact. Furthermore transaction control optimizes the flow of events so that event triggering only takes place when strictly necessary. Observe the difference between:

```
Yaer := ANewYear;
{As a minimum OnChangeYear, OnChangeWeek, OnChangeMonth, OnChangeDate fires}
Month := ANewMonth;
{As a minimum OnChangeMonth, OnChangeMonthNumber, OnChangeDate fires }
Monthday := ANewMonthday;
{As a minimum OnChangeMonthday,   OnChangeDate fires}

BeginChange;
try
    Year := ANewYear;
    Month := ANewMonth;
    Monthday := ANewMonthday;
finally
     EndChange;
{OnChangeYear, OnChangeMonth,   OnChangeWeek, OnChangeDate... fires -only one of each kind}
end;
```

To simplify change of the current date even more use whenever possible one of the Goto... procedures. The Goto...procedures handle change transactions internally, so you don't have to write protected blocks of code. To change the date simply type:

```
GotoDate (ANewYear, ANewMonth, ANewMonthday);
```

Looping the calendar
When working with Tkronos you will undoubtly face the need for iterating over days, weeks, months or years. You might perform repetitive actions by changing the current date for each turn of a loop, thereby reloading all or a lot of the Ext properties. However, this is waste of time when you only need information about one or a few of the time units. TKronos offers a set of functions you can use to make loops as effecient and fast as possible - only generating the kind of information relevant to the task. These are the Fetch functions, one for each time unit: FetchDateExt, FetchWeekExt, FetchMonthExt, etc. Use them whenever possible, they are fast and direct methods to access time units that are not in focus.

See the Fetch example to get a brief demonstration of how to perform loops.

TKronos - processing daytype classes

The standard TDaytype class defines basic calendar information. But suppose you want to put more into a daytype than the standard attributes can tell? Then you have to create a new class of daytype to use with TKronos. The following is a practical discussion of how to use daytype classes, thereby demonstrating different useful TKronos features.

Our task is to create an application that keeps track of some popular astronomic events, that is the phases of moon and earth. To do that we need a new daytype class, we name it TAstro:

Defining the class

```
Type
   TMoonPhase = (mpNew, mpHalfUp, mpHalfDown, mpFull, mpNeither);
   TEarthPhase = (epSpringEquinox, epMidsummer, epAutumnEquinox, epMidwinter, epNeither);

TAstro = class(TDaytype)
  private
    FMoonPhase : TMoonPhase;
    FEarthPhase : TEarthPhase;
  public
    function GetMoonPhase(ADate : TDateTime) : TMoonPhase;
    function GetEarthPhase(ADate : TDateTime) : TEarthPhase;
    constructor Create(DaytypeDef : TDaytypeDef);
  published
    property MonPhase : TMoonPhase read FMoonPhase;
    property EarhPhase : TEarthPhase read FMoonPhase;
end;
```

This is our new class, including some useful types. Note that new daytype classes *must* descend from TUserDaytype. The Moon- and EarthPhase are implemented as read only properties to prevent the user from accidentally changing them. The two functions GetMoonPhase and GetEarthPhase will do the calculations to decide if a particular date is qualified:

(Note: Theese method of calculating moon and earth phases are inexact (+/- a day or so))

```
function TAstro.GetMoonPhase;
var
    Y, M, D: word;
    TempResult : Integer;
    MoonAge : Integer;
begin
        DecodeDate(ADate, Y, M, D);
        TempResult := (Y mod 100) mod 19;
        if TempResult > 9 then
            TempResult := TempResult - 19;
        TempResult := (TempResult * 11) mod 30 + D;
        if M = 1 then
            inc(TempResult, 3)
        else if M = 2 then
           inc(TempResult, 4)
        else inc(TempResult, M);
        TempResult := TempResult * 10;
        if Y < 2000 then
           dec(TempResult, 40)
        else
           dec(TempResult, 83);
        MoonAge := Round((TempResult mod 300) / 10);
         {This is the age of the moon}

        case TempResult of
        0 : FMoonPhase := mpNew;
        7 : FMoonPhase := mpHalfUp;
        14 : FMoonPhase := mpFull;
        21 : FMoonPhase := mpHalfDown;
        else FMoonPhase := mpNeither;
        end;
        Result := FMoonPhase;
end;

function TAstro.GetEarthPhase;
var
    Y, M, D: word;
    TempResult : Integer;
    MoonAge : Integer;
begin
     DecodeDate(ADate, Y, M, D);
     if (M=3) and (D=21) then
       FEarthPhase := epSpringEquinox
     else if (M=9) and (D=23) then
       FEarthPhase := epAutumnEquinox
     else if (M=12) and (D=22) then
       FEarthPhase := epMidwinter;
     else if (M=6) and (D=22) then
       FEarthPhase := epMidsummer
     else
       FEarthPhase := epNeither;
     Result := FEarthPhase;
end;
```

Creating an instance
Now, in our application's FormCreate handler, create and add the astro daytype object. As Astro is a user calculated daytype, the Kronos1 component must have its AllowUserCalc property set to True.

```
procedure AstroApp.FormCreate;
var
    Astro : TAstro;
    DaytypeDef : TDaytypeDef;
begin
    with DaytypeDef do
    begin
          AName := 'Astro'
          ADate := 0;
          AReldayType := 0;
          AnOffset := 0;
          AFirstShowUp := 1;
          ALastShowUp := 9999;
          AShowUpFrequency := 1;
          AHoliday := False;
          AChurchday := False;
          AFlagDay := False;
          AUserCalc := True;
          ATag := 0;
```

```
   end;
   AddDaytype(TAstro.Create(DaytypeDef));
   :
   :
end;
```

Calculating
To decide the show up dates we attach code to the OnCalcDaytype event handler:

```
procedure AstroApp.Kronos1CalcDaytype(Sender: TObject; Daytype: TDaytype;
ADateExt: TDateExt; IsCurrentDate: Boolean; var Accept: Boolean);
var
     Mp : TMoonPhase;
     Ep : TEarthPhase;
     D : TDateTime;
begin
     if Daytype is TAstro then
     with Daytype as TAstro do
     begin
            with ADateExt do
                D := EncodeDate(Year, Monthnumber, Monthday);
             Mp := GetMoonPhase(D);
             Ep := GetEarthPhase(D);
     end;
     Accept := (Mp <> mpNeither) or (Ep <> epNeither);
end;
```

Reading it back
This procedure tracks the astro events in a spesific month:

```
procedure AstroApp.MakeMonthCalendar;
var
   i, j : Integer;
   DayType : TDaytype;
   ADateExt : TDateExt;
begin
    with Kronos1 do
    begin
            GotoDate(Year, 3, 1); //March for example
            for i := 1 to MonthExt.Numdays do
            begin
                    ADateExt := FetchDateExt(Year, Month, i);
                    for j := 1 to ADateExt.DaytypeCount do
                    begin
                            Daytype := FetchDaytype(ADateExt, j);
                            if Daytype is TAstro then
                            with Daytype as TAstro do
                            begin
                                    case MoonPhase of
                                     mpNew : // Draw new moon
                                     mpHalfUp: /Draw rising moon
                                     //etc
                                     end;

                                     case EarthPhase of
                                     epSpringEquniox : //Draw symbol
```

```
                                        epMidsummer : //Draw anoth symbol
                                        //etc
                                        end;
                              end;
                    end;
            end;
    end;
end;
```

Saving and loading
You can use the SaveToFile and LoadFromFile methods with descendent daytype classes. Alas, TKronos will treat them as regular TDaytypes, so you will not be able to save the extended part of the objects. Descendents must be saved in other kinds of files than the standarized textfile used by TKronos.

The load and save procedures, however, have some features that faciliates your own laoding and saving methods. If you write code for the OnSaveDaytype and OnLoadDaytype events you will be notified every time TKronos is about to save and load a daytype from file. At saving point you can choose to write the daytype to your own file or not write it at all, likewise at loading point you can read the daytype from your own file or create it another way:

```
procedure AstroApp.Save;
{You would probably never choose to save the astro object as you create it at runtime. But suppose you
deal with other daytypes you want to save, then you must prevent the object beeing written to the
standard file along with the other daytypes:}
begin
    Kronos1.SaveToFile('Astro.kdt');
end;

procedure AstroApp.Kronos1SaveDaytype(Sender : TObject; Daytype : TDaytype;
var SaveIt : Boolean);
begin
    SaveIt := (Daytype is not TAstro);
    {Prevents the Astro daytype from beeing saved}
end;
```

Suppose you of some kind of reason wish to save Astro to the standard file. Then you had to prevent it from beeing loaded, else it would be created twice:

```
procedure AstroApp.FormCreate;
begin
    Kronos1.LoadFromFile('Astro.Kdt', True);
end;

procedure AstroApp.Kronos1LoadDaytype(Sender : TObject; DaytypeDef : TDaytypeDef;
var LoadIt : Boolean)
var
    Astro : TAstro;
    DaytypeDef : TDaytypeDef;
begin
    LoadIt := (DaytypeDef.AName <> 'Astro');
    if not LoadIt then

 {Load from file if object is not a TAstro object, else create the astro object. Note that you cannot at this
point test for type of object as all daytypes loaded from a standard file are created as regular TDaytype
objects. Here we test the name field, but probably you should use the tag field to classify daytypes}
```

```
    AddDaytype(TAstro.Create(DaytypeDef));
end;
```

Time Unit Properties

Time unit properties are the properties that make up the current date. These are

Year
Month
Week
Monthday
Weekday
Daynumber

Changing one of the time unit properties will immedeatly cause the other TUPs (and Ext properties) to synchronize.

Ext Properties

Ext properties are properties that provide extended information about the current year, month, week or date. These are

YearExt
MonthExt
WeekExt
DateExt
Daytypes

The values of the Ext properties are allways synchronized with the current date.

Daytype list

The daytype list holds the current daytype definitions for the TKronos calendarium. It is made up of the predefined (possibly redefined) daytypes, plus any daytypes you might have added.

You access the list by the Daytypes property.

Daytype

A TKronos daytype is a description of which “role” a date plays on the calendar. Daytypes can be anything from churchdays to birthdays. TKronos comes with numerous predefined daytypes, that is the most common Christian churchdays and international notification days. You can also add your own daytypes, as many as you want. One single date can be attched to as many as 255 daytypes.

All the defined daytypes make up the daytype list.

Yeartype
A TKronos yeartype is a daytype that is not attached to a particaluar date, but works as a notification for the year as a whole. See FetchYeartype and Using daytypes for more information.

Predefined daytypes
Predefined daytypes are those which form the stable part of a TKronos calendar. The standard churchdays and common days are part of the predefined definition as well as any new daytypes added in descendent TKronos components through the SetCountrySpecifics method.

Predefined daytypes cannot be deleted, but some of their attributes might be redefined.