maXbox

Time is on my side

An Introduction to World Time Routines V2.1

1.1 We program time

As you may know the RTL (Runtime Library) includes global routines, utility classes such as those that represent streams and lists and a lot of time routines.
Those routines for working with date/time values (defined in the `System.SysUtils` and `System.DateUtils` units) deserve a closer look.
Let's start with a common mistake to get the UTC time. Greenwich Mean Time (GMT) is often interchanged or confused with Coordinated Universal Time (UTC). But GMT is a time zone and UTC is a time standard.
Therefore UTC is not a time zone, but a time standard that is the basis for civil time and time zones worldwide. This also means that no country or territory officially uses UTC as a local time.
In the language of a function GMT and UTC share the same current time in practice so we make the proof of the pudding:

```
function NowUTC: TDateTime;
var
  system_datetime: TSystemTime;
begin
  GetSystemTime(system_datetime);
  Result:= SystemTimeToDateTime(system_datetime);
end;
```

The result is: UTC: **05/12/2015 15:18:08**
And the same goes with GMT

```
function GMTNow: TDateTime;
begin
  Result:= LocaleToGMT(Now);
end;
```

Again the result is: **05/12/2015 15:18:08**

This is based on my local time of **05/12/2015 16:18:08.**
But what about daylight saving and my local time plus 1 hour? Neither UTC nor GMT ever change for daylight saving time (DST). However, some of the countries that use GMT (officially used in some European and African countries) switch to different time zones during their DST period for example England itself.
We do also update or calculate our locale time relative to GMT with the help of `TTimeZoneInformation`.

This Windows API function `GetTimeZoneInformation` returns this useful record, `TTimeZoneInformation`, with most you want to know especially the bias:

- Bias - the offset in minutes to add to local time to get UTC (Universal Time Coordinated, also/formerly known as GMT, Greenwich Mean Time).
- Time zone name (array of char)
- Daylight savings name (array of char)
- Date and time for start of Daylight savings time
- Daylight savings time adjustment amount

`GetTimeZoneInformation()` also returns a data structure (a record) that contains the current time-zone settings, and the information needed to convert between local and UTC times:

```
function GetGMTBias: Integer;
var
  ainfo: TTimeZoneInformation;
  mode: DWord;
begin
  mode:= GetTimeZoneInformation(ainfo);
  Result:= ainfo.Bias;
```

```
  case mode of
    TIME_ZONE_ID_INVALID:
      RaiseLastOSError;
    TIME_ZONE_ID_STANDARD:
      Result:= Result + ainfo.StandardBias;
    TIME_ZONE_ID_DAYLIGHT:
      Result:= Result + ainfo.DaylightBias;
  end;
end;

function LocaleToGMT(const Value: TDateTime): TDateTime;
begin
  Result:= Value + (GetGMTBias/MinsPerDay);
end;
```

We now have enough information to convert local times to UTC/GMT, and vice versa. As you can see UTC also formerly known as GMT is interchangeable but You should store all your dates as UTC in a DB or another persistence layer. Your server's time zone is really irrelevant to this problem. It's the conversion from UTC to the time zone of your clients that should be your concern!

As a rule: UTC = Local time + Bias

The bias is the difference, in minutes, between UTC and local time.

Who cares about changes in time zones? Your users might. Consider the relatively trivial example of a telephone dialer or a remote control app: wouldn't it be nice and helpful if users were notified of the local time when calling a phone number outside of the local calling area?

If all your application needs is the current UTC or local time, we could simply call the Win32 API procedures `GetSystemTime` or `GetLocalTime`. These functions return a data structure of type `SystemTime`:

```
type
     SystemTime = record
     wYear: Word;
     wMonth: Word;
     wDayOfWeek: Word;
     wDay: Word;
     wHour: Word; wMinute: Word; wSecond: Word;
     wMilliseconds: Word;
end;
```

To obtain your local time here in Europe, you need to subtract (see below) a certain number of hours from UTC depending on how many time zones you are away from Greenwich (England). A table or list can show the standard difference from UTC time to local time.

Besides, mobile computing is so pervasive, a user might easily work in multiple time zones in one single day.

```
function GMTToLocale(const Value: TDateTime): TDateTime;
begin
  Result:= Value - (GetGMTBias / MinsPerDay);
end;
```

```
Example Output of my Time Zone:
Current Bias from local to UTC: -60 minutes
        (UTC = Local time + Bias)
Time zone name: W. Europe Standard Time
Daylight savings starts:  March 29 02:00 am
Daylight savings ends:  October 25 03:00 am
Daylight savings name: W. Europe Summer Time
Daylight savings bias: -60 minutes
We are not currently in the daylight savings time period!
```

The switch to daylight saving time does not affect UTC. It refers to time on the zero or Greenwich meridian, which is not that adjusted to reflect changes either to or from Daylight Saving Time. Now we switch to local time with another API function:

```
function NowLocalTime: TDateTime;
var
  system_datetime: TSystemTime;
begin
  GetLocalTime(system_datetime);
  Result:= SystemTimeToDateTime(system_datetime);
end;
```

We simply use the `getLocalTime` function and format the time:

```
writeln(FormatDateTime('dd-mmm-yyyy  hh:nn:ss', NowLocal()));
```

With this function we return a string representation of `DateTime`.

Now its easy to understand how the now function works using `GetLocalTime` or `GetCurrentTime`:

```
function emulateNow: TDateTime;
var  ST: SystemTime;
     DT: TDateTime;
begin
//Get UTC with GetSystemTime().
  GetLocalTime(ST);
  with ST do
```

```
    DT:= EncodeDate(wYear, wMonth, wDay) +
          EncodeTime(wHour, wMinute, wSecond,wMilliseconds);
   result:= DT;
end;
```

If you need to convert only from UTC to ONE local time you only need to apply the rules for that local time. Most rules are very easy if the time is after year ~1970. Most local times in Europe only have 2 rules, one to enter daylight saving and one to exit from it.
Some types of programs are vitally concerned with time-zone changes, particularly technical programs, or those relating to navigation and astronomy, to name a few.

So we should rename the theory of relativity to law of relativity cause it works!

```
A bref history of time routines End.
```

1.2 Time Zones Table

We have seen there's a description for standard time. For example, "EST" could indicate Eastern Standard Time or W for W. Europe Standard Time. The string will be returned unchanged by the `GetTimeZoneInformation` function. This string can be empty. But where's the whole world time zone information which you can use for a world clock?
Right, settings for each time zone are stored in the following registry key:

```
Const
  SKEY_NT = '\SOFTWARE\Microsoft\
                    Windows NT\CurrentVersion\Time Zones\';
```

Each time zone entry includes several registry values which we can catch in a loop:

```
procedure TXRTLTimeZones_Refresh;
   var
     osV: TOSVERSIONINFO;
     TimeZonesKey: string;
     KeyNames: TStringList;
     Registry: TRegistry;
     i: Integer;
     FItems: TStringlist; //TObjectlist;
     aTimeZone: TXRTLTimeZone;
     FIndex: DWORD;
   begin
     //FItems.Clear;
     osV.dwOSVersionInfoSize:= SizeOf(osV);
     GetVersionEx(osV);
     if osV.dwPlatformId = VER_PLATFORM_WIN32_NT then
       TimeZonesKey:= SKEY_NT
     else
       TimeZonesKey:= SKEY_9X;
```

```
    Registry:= Nil;
    KeyNames:= Nil;
    try
      Registry:= TRegistry.Create;
      Registry.RootKey:= HKEY_LOCAL_MACHINE;
      if not Registry.OpenKeyReadOnly(TimeZonesKey) then Exit;
      KeyNames:= TStringList.Create;
      Registry.GetKeyNames(KeyNames);
      for i:= 0 to KeyNames.Count - 1 do begin
        if not Registry.OpenKeyReadOnly(TimeZonesKey + KeyNames[i])
         then Continue;
          writeln(Registry.ReadString('Display')+' --->');
          writeln(Registry.ReadString('Dlt') + ' :
                          '+Registry.ReadString('Std'));
          writeln(' ');
      end;
     //fitems.Sort;
    finally
      Registry.CloseKey;
      Registry.Free;
      Registry:= NIL;
      KeyNames.Free;
      KeyNames:= NIL;
      //FreeAndNil(Registry);
    end;
  end;
```

Output: (UTC+09:30) Darwin --->
AUS Central Summer Time : AUS Central Standard Time

You can find this script at:
http://www.softwareschule.ch/examples/322_timezones2.TXT

1.3 Time machine

Programmers never die, they just GOSUB without return. You may also know, in the beginning was nothing which exploded ;). So let the jokes aside. Our last step is an eternal clock that goes back in the time like I said before: we code a time machine. So this piece of code can then be translated and run on various platforms and frameworks as well. So what's the solution to run this time forever? Answer: a do forever loop or at least one hour. With the call of another function we set the time one hour back in every second (sign: –i).

```
18  for i:= 1 to 3600 do begin
19    Writeln(TEXTOUT + TimeToStr(AddHours(Time,-i)));
20    Delay(1000)
21  end;
```

Be careful, the loop counter 3600 will last long, so change it step by step on your own experience. The clock goes one hour back into the past every second, but the seconds tick forward like a normal clock. In the film “back to the future” they call it the flux-comparator.

 If you want to stop or **break a loop**, just override the loop counter in line 18 and recompile (F9) it during the execution!

Repeat until Key-pressed is also possible with the snippet:

```
repeat {for it:= 1 to n do} until isKeypressed;
//keypress in output window below (memo2 as console)
```

When you call the function `AddHours` that takes two argument and returns another time, then we say the function call `TimeToStr(AddHours(Time,-i))` is nested. A nested call contains other functions within a statement. Let me explain:
First we call the time function, the result we pass to the `AddHours` function its result is passed to the `TimeToStr` function and the whole we pass again to `Writeln`!

When we want to travel back to the past, maybe in the year of 1759, further information is missing. Right, there is no date. Easier done than said ;-).

```
18  for i:= 1 to round(Power(2,4)) do begin
19    Writeln(TEXTOUT + DateTimeToStr(AddHours(Now,-i)));
```

Did you see the difference? We replaced the `Time` function with the `Now` function and the string converter to `DateTimeToStr`. And with `Power` big numbers are possible (Power is like 2^4). Now we’re ready to go back to middle age.

  How can you accelerate your time machine? One hour back per second takes to much time, simply the loop must step faster.
You got it; we change the parameter of the delay procedure:

```
20    Delay(10);
```

Remember: The clock rate is still the same, but our time machine can go faster to the past. The time[1] is flushing by with this speed; the calculation of how many lines we get is also interesting:

Suppose we have `Power(2,12)` as the for counter limit, how many lines we get?
Answer: 2^12 = 4096.
And the next by `Power(2,30)` could be also of interest, but its huge and your app runs long!:
Can you imagine where in the past we are landing, middle age or stone age or maybe far out of our history line?
The calculation is simple: (2^30/24)/365 is rounded to 122573 years. This would be Stone Age! How can you do that in maXbox: 2^30 / (24*365) is another solution.

```
  Writeln(intToStr(round(Power(2,30)/24/365)));
```

[1]Time is just measuring movement of our solar system

We come closer to the end and had to re-factor just one thing:

```
for i:= 1 to round(Power(2,N)) do begin
  Writeln(IntToStr(i)+TEXTOUT +
            DateTimeToStr(AddHours(Now,-i)));
  Delay(SN);    //speed of time machine
end;
```

As you can see I introduced two parameters to be more flexible, the counter limit `N` and the speed of time machine `SN`.
Means also 2 more variables to add in your code.

By the way: DYNAMIC_TIME_ZONE_INFORMATION specifies settings for a time zone and dynamic daylight saving time. For more information about the Dynamic DST key, see DYNAMIC_TIME_ZONE_INFORMATION and the special function `GetDynamicTimeZoneInformation`.

Both StandardName and DaylightName are localized according to the current user default UI language.

So far we have learned something about time routines and the difference between a UTC and a GMT. Now its time to reflect over those used functions:

Function	Explanation and Purpose…
TimeStampToMSecs	Convert Timestamp to number of millicseconds
DecodeTime()	Decode DateTime to hours, minutes and seconds
DateTimeToStr()	Converts a variable of type TDateTime to a string.
AddHours()	Change the hours of a TDateTime function.
Now	Returns the current date and time.
EncodeTime()	Encode hours, minutes and seconds to DateTime
SystemTimeToDateTime	Convert system time to datetime
GetTimeZoneInformation	Returns a timezone relevant record

If you want to look at the whole script you can find the file at:

http://www.softwareschule.ch/examples/650_time_routines.txt

A least a piece of code which makes UTC time for further studies:

```
function MakeUTCTime(DateTime: TDateTime): TDateTime;
var TZI: TTimeZoneInformation;
```

```
begin
  case GetTimeZoneInformation(TZI) of
    TIME_ZONE_ID_STANDARD:
      begin
        Result := DateTime + (TZI.Bias/60/24);
      end;
    TIME_ZONE_ID_DAYLIGHT:
      begin
        Result:= DateTime + ((TZI.Bias+TZI.DaylightBias)/60/24);
     end
  else
    raise
      //Exception.Create('Error converting to UTC Time. Time zone
could not be determined.');
  end;
end;
```

In UTC time seconds can be in the range 0 to 60,

If there is a leap second planned, seconds can have the value 60.

You can have official information about planned leap seconds from the "International earth rotation and reference systems service (iers)" at http://hpiers.obspm.fr/iers/bul/bulc/bulletinc.dat

Below, a cut and paste from Bulletin C:

A positive leap second will be introduced at the end of June 2015.
The sequence of dates of the UTC second markers will be:

```
                        2015 June 30,     23h 59m 59s
                        2015 June 30,     23h 59m 60s
                        2015 July  1,      0h  0m  0s
```

```
type
  PTimeZoneInformation = ^TTimeZoneInformation;
  _TIME_ZONE_INFORMATION = record
    Bias: Longint;
    StandardName: array[0..31] of WCHAR;
    StandardDate: TSystemTime;
    StandardBias: Longint;
    DaylightName: array[0..31] of WCHAR;
    DaylightDate: TSystemTime;
    DaylightBias: Longint;
  end;
  //{$EXTERNALSYM _TIME_ZONE_INFORMATION}
  TTimeZoneInformation = _TIME_ZONE_INFORMATION;
```

Feedback @ max@kleiner.com

Literature: Kleiner et al., Patterns konkret, 2003, Software & Support

`Links of maXbox and Time Routines:`

http://www.softwareschule.ch/maxbox.htm

http://en.wikipedia.org/wiki/Time_travel

http://www.timeanddate.com/time/gmt-utc-time.html

http://www.freepascal.org/docs-html/rtl/sysutils/datetimeroutines.html