maXbox

maXbox Starter 23

Start with maXbox Real Time

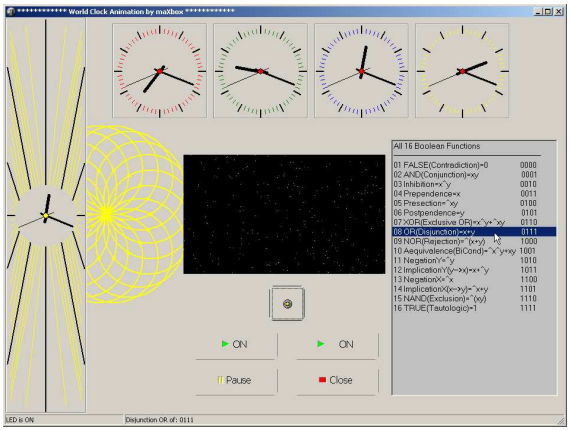
1.1 From Time to Real Time

Real-time systems have evolved over the past decades in a relatively calm manner - performance has increased, one can say dramatically, but the main paradigms were pretty stable since the mid 80s. This has changed now. The big change that is moving into the embedded field is multicore - and that is not an adaptation of our current methods but a redesign from scratch in quite a few cases - notably of our way of thinking about real-time.

So what's real time in a definition?

A real-time system is a type of hardware or software that operates with a time constraint.

Most of the time it's a short time constraint and the response time has to be fast enough but it can also be a precise constraint in the meaning of accuracy of time.
Sort of a wait condition can also be a constraint to wait in time! A wait property in maXbox determines whether control is returned to the application before a resume method has completed.



If a maXbox script or app is programmed to assume a time standard, it is always started relative to the time your operating system provides or the host itself is:

```
  Label1.Caption:= 'The time is ' + TimeToStr(Time);
```

The Time function returns the current time. Especially the `TMediaPlayer` has a sophisticated time format. The `TimeFormat` property determines the format used to specify position information in tracks (run-time only).

```
property TimeFormat: TMPTimeFormats;
```

TimeFormat determines how the StartPos, Length, Position, Start, and EndPos properties are interpreted. For example, if position is 180 and TimeFormat is tfMilliseconds, the current position is 180 milliseconds into the medium. If position is 180 and TimeFormat is tfMSF, the current position is 180 minutes into the medium.

Not all formats are supported by every device. If you try to set an unsupported format, the assignment is ignored.
The current timing information is always passed in a 4-byte integer. In some formats, the timing information returned is not really one integer, but single bytes of information packed in the long integer (casting is possible).

It is a useful property function where you always get a meaning of multimedia.
The following code declares a HMSRec record with four byte fields.
If TimeFormat is tfHMS, the first field specifies hours, the second field specifies minutes, the third field specifies seconds, and the fourth field corresponds to the unused most-significant byte of the tfHMS time format. A LongInt variable is typecast to an HMSRec record, then the hours, minutes, and seconds of the Length of the loaded media are displayed [1].

```
procedure TimeFormatDemo;
```

```
49:procedure TimeFormatDemo;
50:var
51: TheLength: LongInt;
52: aplayer: TMediaPlayer;
53: ahmsrec: HMSrec;
54: begin
55:   aplayer.TimeFormat:= tfHMS; {Set time format - note some devices don't
                                   support tfHMS }
56:   TheLength:= aplayer.Length; { Store length of currently loaded media in var }
57:   with aHMSRec {(TheLength)} do begin{ Typecast TheLength as a HMSRec record }
58:     writeln(IntToStr(longint(Hours))); { Display Hours in Label1 }
59:     writeln(IntToStr(Minutes)); { Display Minutes in Label2 }
60:     writeln(IntToStr(Seconds)); { Display Seconds in Label3 }
61:   end;
62: end;
```

As you know the configuration of maXbox and time settings is also possible with a boot loader script and a simple ini-file too. Extensions are possible with the Open Tools API and a small CLI (Command Line Interface). Hope you did already read Starters 1 till 22 at:

http://sourceforge.net/apps/mediawiki/maxbox/

First we start with a meaning of real time and the well known application.processmessages.
A hard real-time system (also known as an immediate real-time system) is hardware or software that must operate within the confines of a stringent deadline.
The application may be considered to have failed if it does not complete its function within the allotted time span. Examples of hard real-time systems include components of pacemakers, anti-lock brakes, robot navigation and aircraft control systems.

1.2 ProcessMessages;

In this lesson we deal with the way to get events in a certain time.
The ProcessMessages method interrupts the execution of your application so that Win can respond to events. For example, the user might want to move a form on the screen while your

[1] tfHMS: Minutes, seconds, and frames packed into a 4-byte integer.

application is doing some complex processing that would ordinarily prevent Windows from responding to keyboard or mouse events.
So the confines of a stringent deadline is not guaranteed, no response no function!
By calling `ProcessMessages`, your application permits Win to process these events at the time `ProcessMessages` is called. The `ProcessMessages` method cycles the Win message loop until it is empty and then returns control to your application.

```
277  procedure TForm1GridClick(Sender: TObject);
278: begin
279:   showMessageBig('Grid has been clicked');
280: end;
281:
282: procedure TForm1Button1Click(Sender: TObject);
283: var
284:   I,J,X,Y: Word;
285: begin
286:   I:= 0; J:= 0;
287:   myform.Canvas.Textout(20,235,S_RepeatChar(90,' '));
288:   while I < 64000 do begin
289:     Randomize;
290:       while J < 64000 do begin
291:         Y:= Random(J);
292:         Inc(J);
293:       end;
294:     X:= Random(I);
295:     mygrid.Cells[3,3]:= intToStr(X);
296:     Inc(I);
297:   end;
298:   //TForm(sender).Canvas.TextOut(10, 10, 'handler is finished');
299:   myform.Canvas.TextOut(20, 235, 'Ignore Button1Click handler is finished');
300: //writeln(objectToStr(sender));
301: end;
302:
```

```
303: procedure TForm1Button2Click(Sender: TObject);
304: var I,J,X,Y: Word;
306: begin
307:   I:= 0; J:= 0;
308:   myform.Canvas.Textout(20,255,S_RepeatChar(90,' '));
309:   while I < 64000 do begin
310:     Randomize;
311:     while J < 64000 do begin
312:       Y:= Random(J);
313:       Inc(J);
314:       //Application.ProcessMessages;
315:     end;
316:     X:= Random(I);
317:     mygrid.Cells[3,3]:= intToStr(X);
318:     Inc(I);
319:     Application.ProcessMessages;
320:   end;
321:   myform.Canvas.TextOut(20, 255, 'Process Button2Click handler is finished');
322: end;
```

This example uses two buttons that are long enough to accommodate lengthy captions on a form. When the user clicks the button with the caption `Ignore Messages`, the code begins to generate a long series of random numbers (run rundom ;-)).

If the user tries to resize the form while the handler is running, nothing happens until the handler is finished. When the user clicks the button with the caption `Process Messages` (see picture below), more random numbers are generated, but Windows can still respond to a series of mouse events, such as resizing the form.

Note how quickly these event handlers run depends on the microprocessor of your computer. So no real time can be calculated in a certain way. A message appears on the form informing you when the handler has finished executing.
You can find the example at:

http://www.softwareschule.ch/examples/374_realtime_random2.txt

You know you can create multiple instances of maXbox. You can create multiple instances of the same app to execute parallel code, just type <F4>. For example, you can launch a new instance within a script of the box in response to some user action, allowing each script to perform the expected response.

```
ExecuteShell(ExePath+'maxbox3.exe','"'+ExePath+'examples\'+ascript+'"');
ShellExecute3(ExePath+'maxbox3.exe',ExePath+'examples\'+ascript,secmdopen);
```

So creating multiple instances results in launching multiple instances of the application and the scripts too. There's no good way to launch one application (maXbox) with multiple scripts in it cause time behaviour is dependent of the CPU.

An external script or a least a second one could also be a test case to compare the behaviour. Each test case and test project is reusable and rerunnable, and can be automated through the use of shell scripts or console commands.

An advantage of using `ShellExecute3` is the no wait condition, means less CPU overhead and consumption and loose coupling.
The same you can do with runfile sync or async with no wait condition:

```
Function RunFile( FileToRun: string; Params: string; Dir: string; Wait:
                                                    boolean) : cardinal');
Function RunFile_( Cmd, WorkDir : string; Wait : boolean) : Boolean');
```

Next we check how you can run an endless loop (hope it isn't endless) to study the time behaviour and responsiveness.

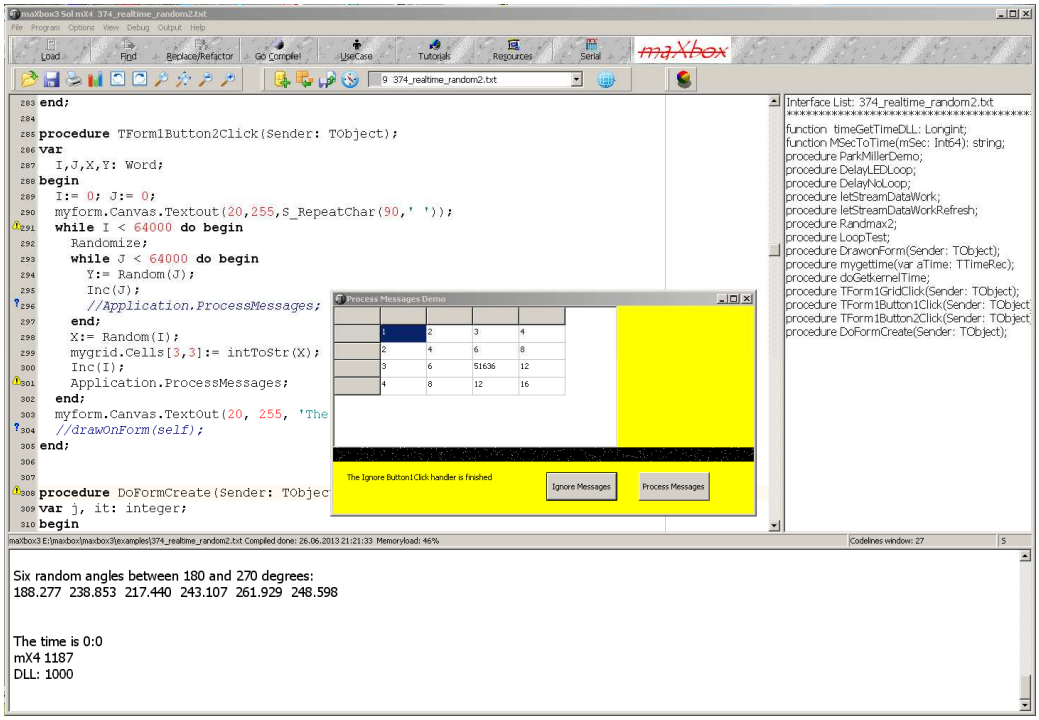
To stop for example a repeat loop, just use `isKeypressed` in your scripts!

```
220: procedure LoopTest;
221: begin
222:   Randomize;
223:   REPEAT
224:     //TextAttr:=Random(256);
225:     Writeln(inttoStr(Random(256*256)));
226:   UNTIL isKeypressed; //on memo2 output
227:   if isKeypressed then writeln('key has been pressed!');
228: end;
```

Or with a <while NOT> loop:

```
228: procedure LoopTest2;
229: begin
230:   Randomize;
231:   WHILE NOT isKeypressed do
232:     Writeln(intToStr(Random(256*256)));
233: end;
```

When I start a loop through another one small UI like a button a service or script has started successfully and it is running in the machine. It is running even I restart another instance. But then you can stop the loop with a key press on the output window cause the loop has a condition on the while or repeat statement. Normally you would handle this in a form but for scripts without a form or a timer it's a useful way of control a loop flow in real time[2].



2: Feel the Difference

1.3 Real Time Test

As you already know the event handlers run depends on the microprocessor and your overhead of running tasks. If the system running your program has multiple processors, you can improve performance by dividing the work into several threads and letting them run simultaneously on separate processors.
But your system has multitasks also, you can improve performance by just let the operating system dividing the work into several tasks and letting them run pseudo simultaneously on different time slices. Not all operating systems implement true multi-processing, even when it is supported by the underlying hardware.

☝ In maXbox you do have to design and code multithreading but for multitasking advantage nothing has to be done or at least start a new instance of the box.

Lets check this with a simple sleep procedure and the measuring of time.

[2] Real Time in this sense means you press the key and it stops immediately!

The `timeGetTime` function retrieves the system time, in milliseconds. The system time is the time elapsed since Windows was started.

Try to find out which time is hiding behind this windings:?

When developing real time applications, choosing to spawn a new thread or task depends on the nature of the service being provided, the anticipated number of connections and resources, and the expected number of processors and tasks on the computer running; means the more tasks and services in the background (system load) the less real time behaviour you get.
With a simple test we can prove that:

```
387: timeres:= timeGetTime;
388: Sleep(1000);
389: Writeln('mX4 '+intTostr(timeGetTime-timeres));
```

So the difference or result should be 1000 milliseconds (one second) but there's no certainty. `Sleep` suspends execution of a program for a certain time. Sleep suspends the execution of the program for the specified number of milliseconds (milliseconds). After the specified period has expired, program execution resumes.

The indicated time is not exact, i.e. it is a minimum time. No guarantees are made as to the exact duration of the suspension.

You might say to try it with the procedure `delay` but it's the same problem. Delay waits a specified number of milliseconds. The number of specified seconds is an approximation, and may be off a lot, if system load is high.

Another function to measure time is `timeGetSystemTime`. The only difference between this function and the `timeGetSystemTime` function is that `timeGetSystemTime` uses the `MMTIME` structure to return the system time. The `timeGetTime` function has less overhead than `timeGetSystemTime`.

I did also test it with the direct DLL function and the mX System function:

```
function  timeGetTimeDLL: Longint;
   external 'timeGetTime@winmm.dll stdcall';

  Writeln('DLL: '+inttostr(timeGetTimeDLL-timeres));
```

Note that the value returned by the `timeGetTime` function is a `DWORD` value. The return value wraps around to 0 every 2^32 milliseconds, which is about 49.71 days. This can cause problems in code that directly uses the `timeGetTime` return value in computations, particularly where the value is used to control code execution. You should always use the difference between two `timeGetTime` return values in computations.

The default precision of the `timeGetTime` function can be five milliseconds or more, depending on the machine. You can use the `timeBeginPeriod` and `timeEndPeriod` functions to increase the precision of `timeGetTime`. If you do so, the minimum difference between successive values returned by `timeGetTime` can be as large as the minimum period value set using `timeBeginPeriod` and `timeEndPeriod`. Use the `QueryPerformanceCounter` and `QueryPerformanceFrequency` functions to measure short time intervals at a high resolution,

I ended up using `QueryPerformanceCounter()` to get a more precise timing than using just a sleep() or delay, but sleep works better than delay().

Now let's take a look at a "real" real time library.

SuperPascal is based on Niklaus Wirth's 'sequential language Pascal, extending it with features for safe and efficient concurrency. Pascal itself was used heavily as a publication language in the 1970s; it was used to teach structured programming practices and featured in text books, for example, on compilers and programming languages. Brinch Hansen had earlier developed the language Concurrent Pascal one of the earliest concurrent languages for the design of operating systems and real-time control systems.

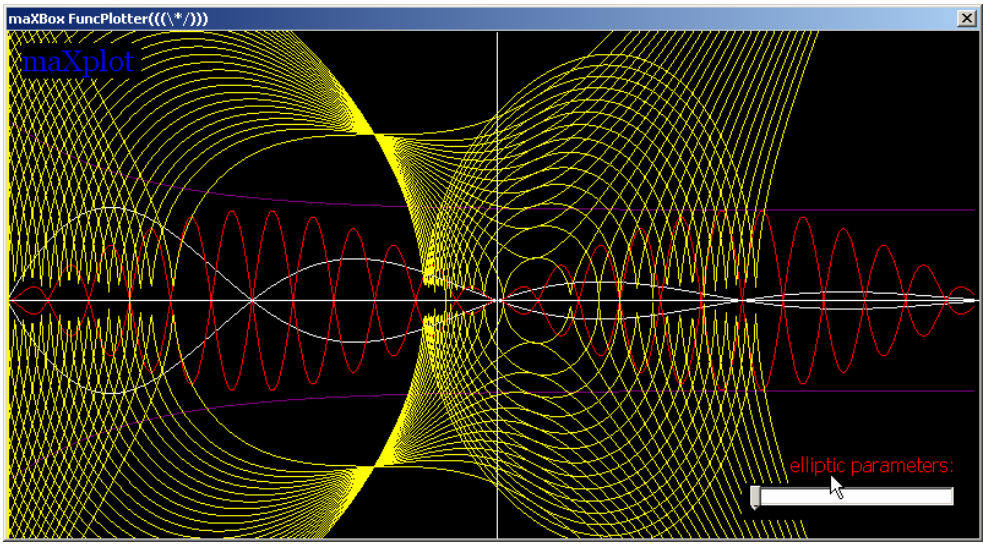
1.3.1 FP-RTOS

A real time kernel for embedded development. Primarily developed for ARM devices. Written entirely in Pascal and inline assembler.

- Pre-emptive multitasking
- Synchronization primitives(Mutex, Critical section, spinlock, and signals)
- Thread safe queue and delays
- Optional safety features: Deadlock detection, priority inheritance, etc

If we are going to depend on real-time programming systems in our daily lives, we must be able to find such obscure errors before the systems are put into operation.
Fortunately, a compiler can detect many of these errors if processes and monitors are represented by a structured notation in a high-level programming language.
In addition, we must exclude low-level machine features (registers, addresses, and interrupts) from the language and let a virtual machine control them. If we want real-time systems to be highly reliable, we must stop programming them in assembly language.

If you only want to "install" a new maXbox with file or directory names, be sure the ini-file will not be overwritten by unpacking the zip (so let's make a copy before). Maybe you just know that by starting the maXbox it checks on the internet the last version.



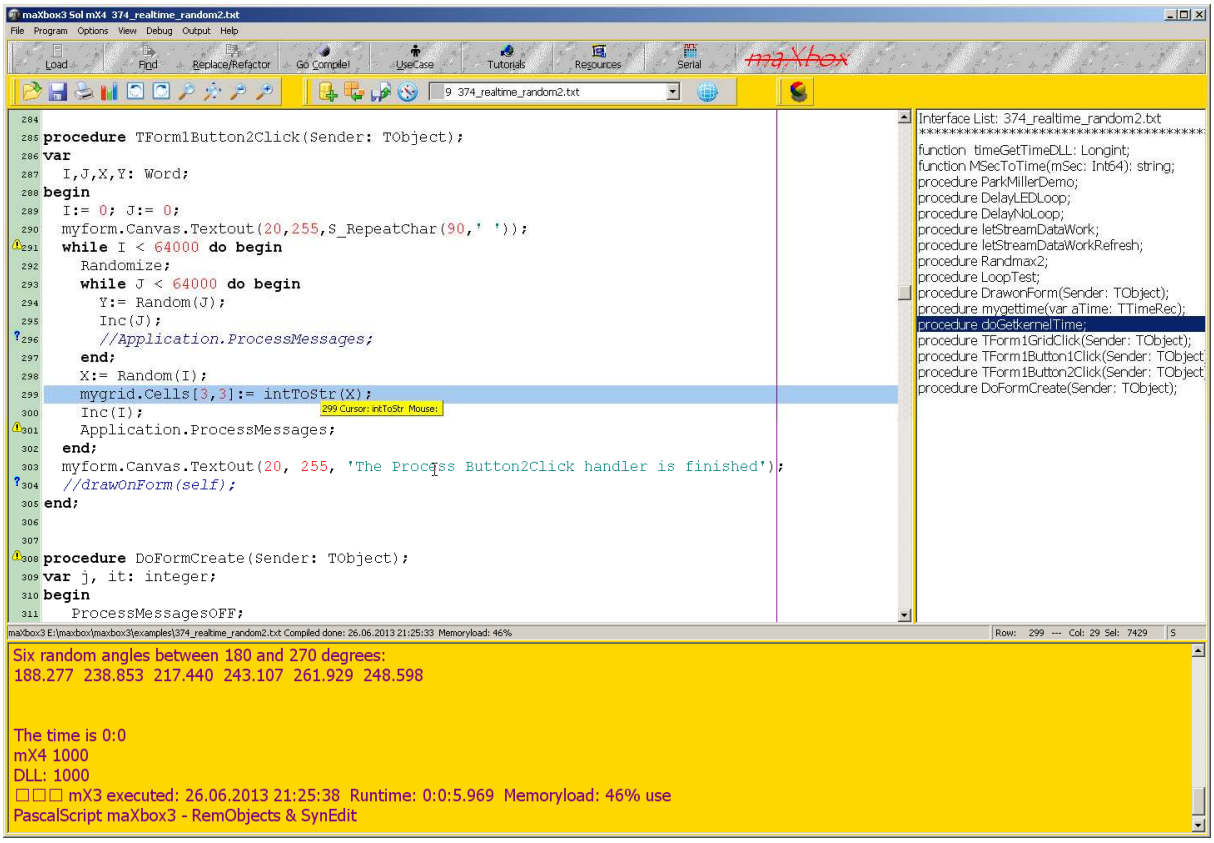
3: Another function of time

Let's do the last step with the above mentioned function. Unfortunately the `QueryPerformance Frequency` does not work in a script cause of missing pointer but the DLL call works (see below)! It retrieves the frequency of the high-resolution performance counter, if one exists. If the function fails, the return value is zero. The frequency cannot change while the system is running.

```
406: if QueryPerformanceFrequency(PerfFreq) then
407:   QueryPerformanceCounter(StartExec);
408: QueryPerformanceCounter(EndExec);
409: writeln(Format('Debug: (readcount = %d), ExecTime = %.3f ms',
410: [4{ReadCount}, ((EndExec - StartExec)*1000.0)/PerfFreq]))

{$I ..\maxbox3\examples\305_eliza_engine.INC}
```

If its not find a valid pointer it says:
Access violation at address 7C8256F8 in module 'kernel32.dll'. Read of address 00000000.



4: With or without ProcessMessages

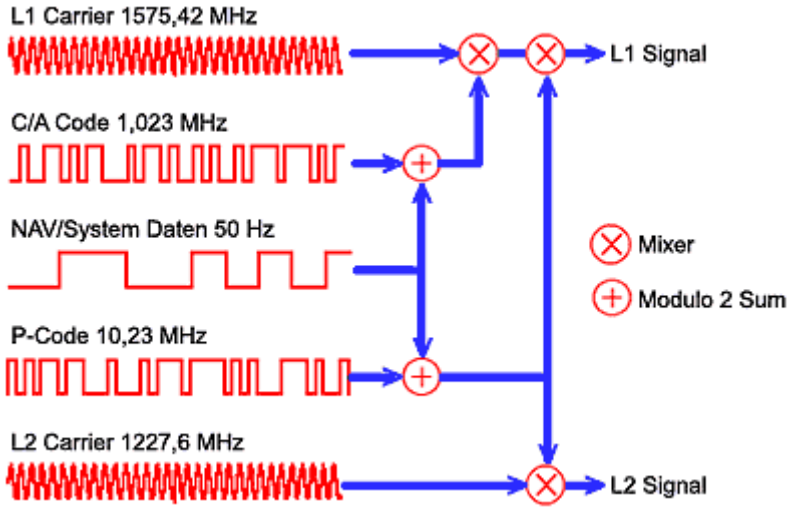
1.4 The OS Lib

Linux in embedded systems is well established, ranging from consumer electronics to network devices and increasingly industrial applications including safety related systems. The technological resources suitable for high-availability, real-time, and safety critical systems have been continuously expanding and improving - allowing to cover the entire development life cycle of industrial projects based on open-source tools.

At the core of this development is the availability of stable operating systems with reliable real-time properties. Extending and improving these real-time properties of open-source RTOS is continuous research and development effort that OSADL documents in the form of the annual Real Time Linux Workshop.

Below an internal extract with some time function macros from the help file “All Functions List” `maxbox_functions_all.pdf`:

```
//------------------------------------------------------------------------------
10181: //**************mX4 Macro Tags ************************************
10182: //------------------------------------------------------------------------
10183:
10184: #name, ı#date, ı#host, ı#path, ı#file, ı#head
10185:
10186: SearchAndCopy(memo1.lines, '#name', getUserNameWin, 11);
10187: SearchAndCopy(memo1.lines, '#date', datetimetoStr(now), 11);
10188: SearchAndCopy(memo1.lines, '#host', getComputernameWin, 11);
10189: SearchAndCopy(memo1.lines, '#path', fpath, 11);
10190: SearchAndCopy(memo1.lines, '#file', fname, 11);
10191: SearchAndCopy(memo1.lines, '#locs', intToStr(getCodeEnd), 11);
10192: SearchAndCopy(memo1.lines, '#perf', perftime, 11);
10193: SearchAndCopy(memo1.lines, '#head',Format('%s: %s: %s %s ',
10194: [getUserNameWin,getComputernameWin,datetimetoStr(now),Act_Filename]),11);
//------------------------------------------------------------------------------
```



5: GPS satellite signals coding

Try to find out which real time constraints works with a GPS?

A GPS receiver calculates its position by precisely timing the signals sent by GPS satellites high above the mother Earth.

Discuss: I hope you are aware that these are supposed to be mapped at the JNI layer or a DLL, OCX to corresponding GPS vendor APIs. You are right when you say that these are implemented as part of HAL layer. The HAL layer will be part of GPS chipset Vendor code.

I was wondering if it's possible to use the GPS provider with the Network Provider at the same time and then get the best result of both? And is it also possible to let me know when the GPS provider provided me with an GPS Location?

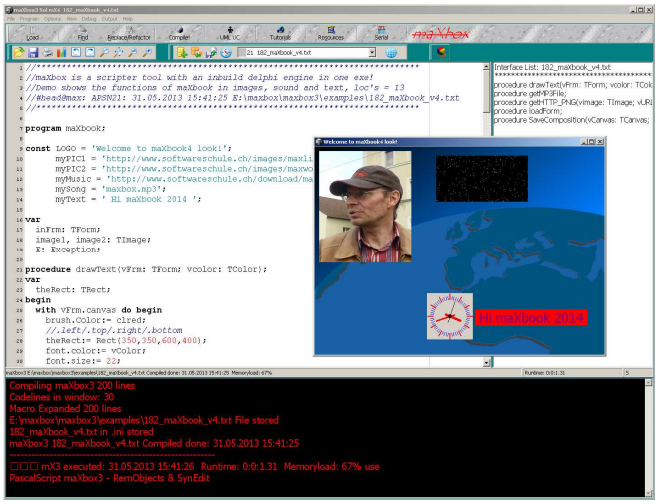
Try to use GPS, as long as it is not available use the `NETWORK_PROVIDER`. But when GPS get available switch to it. When the GPS is lost switch back to the `NETWORK_PROVIDER`.

Feedback: max@kleiner.com

Links of GPS, maXbox and DelphiWebStart:
http://en.wikipedia.org/wiki/Global_Positioning_System
http://en.wikipedia.org/wiki/GPS_navigation_software
http://www.gpsvisualizer.com/

http://www.softwareschule.ch/maxbox.htm
http://sourceforge.net/projects/maxbox
http://sourceforge.net/apps/mediawiki/maxbox/
http://sourceforge.net/projects/delphiwebstart

http://www.softwareschule.ch/maxbox_mainscreen.png




```
function  QueryPerformanceFrequencyDLL(var lfreq: int64): bool;
  external 'QueryPerformanceFrequency@kernel32.dll stdcall';
function  QueryPerformanceCounterDLL(var lcount: int64): bool;
  external 'QueryPerformanceCounter@kernel32.dll stdcall';

var StartExec, EndExec, PerfFreq: Int64;

 if QueryPerformanceFrequencyDLL(PerfFreq) then
   QueryPerformanceCounterDLL(StartExec);
 Sleep(1000);
 QueryPerformanceCounterDLL(EndExec);
 writeln(Format('Perf Counter Debug:: (readcount = %d), ExecTime = %.3f ms',
                [1 {ReadCount}, ((EndExec - StartExec)*1000.0)/PerfFreq]))
```

Remarks out of scope maXbox:

On a multiprocessor computer, it should not matter which processor is called. However, you can get different results on different processors due to bugs in the basic input/output system (BIOS) or the hardware abstraction layer (HAL). To specify processor affinity for a thread, use the `SetThreadAffinityMask` function.

1.5 Appendix

EXAMPLE: TimeFormat

Working with TimeFormat

The following table lists the possible values for the *TimeFormat* property:
Value Time format

tfMilliseconds Milliseconds are stored as a 4-byte integer variable.

tfHMS Hours, minutes, and seconds packed into a 4-byte integer. From least significant to most significant byte, the data values are Hours (least significant byte) Minutes Seconds Unused (most significant byte)

tfMSF Minutes, seconds, and frames packed into a 4-byte integer. From least significant to most significant byte, the data values are Minutes (least significant byte) Seconds Frames Unused (most significant byte)

tfFrames Frames are stored as a 4-byte integer variable.

tfSMPTE24 24-frame SMPTE packs values in a 4-byte variable. From least significant to most significant byte, the data values are Hours (least significant byte) Minutes Seconds Frames (most significant byte) SMPTE (Society of Motion Picture and Television Engineers) time is an absolute time format expressed in hours, minutes, seconds, and frames. The standard SMPTE division types are 24, 25, and 30 frames per second.

tfSMPTE25 25-frame SMPTE packs data into a 4-byte variable in the same order as 24-frame SMPTE.

tfSMPTE30 30-frame SMPTE packs data into the 4-byte variable in the same order as 24-frame SMPTE.

tfSMPTE30Drop 30-drop-frame SMPTE packs data into the 4-byte variable in the same order as 24-frame SMPTE.

tfBytes Bytes are stored as a 4-byte integer variable.

tfSamples Samples are stored as a 4-byte integer variable.

tfTMSF Tracks, minutes, seconds, and frames are packed in the 4-byte variable. From least significant to most significant byte, the data values are Tracks (least significant byte) Minutes Seconds Frames (most significant byte)

Note that MCI uses continuous track numbering.

Functions provided with MCI to help you decode the 4-byte integer specified in a given time format are documented under *MCI Macros for Encoding and Decoding Time Data* in the MMSYSTEM.HLP Help file.