Timer in Delphi

Einführung

Mit Timern kann man in festen Zeitlichen Abständen periodisch Code ausführen. Oder wie es in der Delphi Hilfe steht: "Mit einem Timer kann nach Ablauf einer bestimmten Zeitspanne ein Ereignis ausgelöst werden." In Delphi gibt es verschiedene Möglichkeiten um einen Timer zu realisieren, die ich in diesem Tutorial vorstellen möchte.
Timer werden in vielen Bereichen der Programmierung eingesetzt, beispielsweise bei Animationen oder in Spielen allerdings sollten Timer nicht Events ersetzten.
So sollte man zum Beispiel nicht mit einem Timer durch ständiges abfragen des Status darauf warten, dass ein Button angeklickt wird. Dafür gibt es entsprechende Events.
Ein wichtiger Vergleichswert zwischen den Timer Varianten ist das minimale Intervall. Damit ist gemeint, wie viel Zeit in Millisekunden minimal zwischen den Timer aufrufen vergehen kann.

Uptime

Als erstes werde ich 2 Funktionen vorstellen mit denen man die Uptime, also die Zeit die Seit dem Windowsstart vergangen ist, ermitteln kann. Mit diesen Funktionen ist es Möglich die Vergangene Zeit zu ermitteln. Man kann so z.B. die Zeit ermitteln die Ausführen einer Funktion vergeht oder die Zeit die seit dem letzten Timer Aufruf vergangen ist.
Da gibt es einmal die Funktion **Timegettime** mit der man die Uptime in Millisekunden (1/1.000 Sekunde) ermitteln kann und die Funktion **QueryPerformanceCounter** (1/1.000.000 Sekunde). Bei Timegettime ist zu beachten, dass die unit mmsystem eingebunden werden muss.
QueryPerformanceCounter funktioniert nur auf "neueren" Prozessoren.(Ich hab das jetzt nicht getestet aber ich denke so ab Pentium sollten es mit den Prozessoren kein Problem geben.) Mit der Funktion QueryPerformanceFrequency kann man prüfen ob die Procedure verfügbar ist.
Und noch ein kleines Beispiel bei dem ich die Funktion timegettime mit QueryPerformanceCounter(Zeit) nach gebaut habe. Das bringt zwar nicht viel zeigt aber gut die Funktionsweise.
Der Vollständigkeit halber möchte ich auch noch auf die Funktion **GetTickCount** hinweisen, die aber ungenau ist und deshalb nicht verwendet werden sollte.

```
function Customtimegettime: int64;
var
  Frequenz: int64;
  Zeit: int64;
begin
  if QueryPerformanceFrequency(Frequenz) = true then{ist die Funktion verfügbar}
  begin
    QueryPerformanceCounter(Zeit);
    result := Zeit div (Frequenz div 1000); {umrechnen auf 1/1000 Sekunde}
  end
  else
  begin
    result := timegettime; {Wenn QueryPerformanceFrequency(Frequenz) nicht geht}
  end;
end;
```

So dann hätten wir das auch geklärt und können zum wesentlich kommen. Im Folgenden werde ich einige Möglichkeiten zeigen wie so ein Timer zu realisieren ist und dabei auf Vor und Nachteile der jeweiligen Methode ansprechen.

1. VCL Timer

Fangen wir mit leichtesten Methode dem VCL Timer (Standard Timer) an. Die Eigenschaften sind selbsterklärend und es gibt nur ein Event. Zu finden ist die Komponente in der Registerkarte System.
Minimal Intervall:
NT/XP/2000 = ca.10
95/98 = 55
Die 55 ergibt sich aus dem Interrupt 1ch der 18,2 mal pro Sekunde aufgerufen wird. Dem einen oder anderen ist das vielleicht noch aus den DOS Zeiten bekannt. Wo man den Interrupt (1ch) verbiegen konnte und so einen Timer hatte.

Und jetzt ein Beispiel um es noch etwas "Interessanter" zu machen wird der Timer erst zur Laufzeit erstellt. Das Programm zeigt die maximale Geschwindigkeit des Timers an.

```
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ExtCtrls, mmsystem; //!!!!  mmsystem einbinden (für timegettime)

type
  TForm1 = class(TForm)
    procedure FormCreate(Sender: TObject);
  private

  public
    FTimer: TTimer;
    Zeit: int64;
    Counter: integer; //Eine Variable zum Zählen der Timer Durchläufe
    procedure OnTimer(Sender: TObject);//Event Handler
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.OnTimer(Sender: TObject);
begin
  inc(Counter);   //Zähler um 1 erhöhen
  if timegettime-Zeit>1000 then //Jeweils nach einer Sekunde
  begin
    Canvas.TextOut(10,10,inttostr(counter)); //Zähler ausgeben
    zeit:= timegettime; //neue Start Zeit ermitteln
    Counter := 0;   //und Zähler wieder auf 0 setzen.
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  FTimer := TTimer.create(self); {Man kann auch statt self nil einsetzen
   dann muss Aber auch noch Ftimer wieder freigeben.}
  FTimer.Interval := 1;
  FTimer.OnTimer := OnTimer;    //Eventhandler wird zugewiesen
  FTimer.Enabled := true;
  //Der Timer wurde erstellt
  Counter := 0;
  Zeit := Timegettime           //Die Startzeit wird ermittelt
end;
```

2. Non VCL Timer

Es handelt sich um die gleiche Technik wie bei 1. (Win API) nur das diesmal die Win API direkt genutzt wird. Dazu verwendet man die Funktion Settimer aus der Windows Api.
Der Funktion muss der Handle des Fensters das die Nachricht erhalten soll, eine TimerID die man selber wählen kann und ein Intervall übergeben werden. Der letzte Parameter, die Adresse einer Timer procedure ist, ist für uns nicht relevant.
Nach dem Aufruf der Funktion wird an das Fenster die Nachricht WM_TIMER (wParam enthält die ID des Timers) in dem definierten Intervall gesendet. Auf diese Nachricht kann man dann im Programm reagieren. Mit KillTimer wird der Timer wieder freigegeben.
Man kann zwar wie auch bei dem VCL Timer ein Intervall von 1 einstellen allerdings beträgt das minimal mögliche Intervall wieder nur:

NT/XP/2000 = ca.10
95/98 = 55

In dem folgenden Beispiel wird wieder die maximale Geschwindigkeit des Timers ermittelt.

```
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ExtCtrls, mmsystem; //!!!!  mmsystem einbinden (für timegettime)

type
  TForm1 = class(TForm)
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  public
    Zeit: int64;
    Counter: integer; //Eine Variable zum Zählen der Timer Durchläufe
    procedure OnTimer;
    procedure WndProc(var Msg: TMessage); override;
  end;

const
  TIMER_ID = 1;  //Eine Constande als Timer ID
var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.OnTimer;
begin
  inc(Counter);   //Zähler um 1 erhöhen
  if timegettime-Zeit>1000 then //Jeweil nach einer Sekunde
  begin
    Canvas.TextOut(10,10,inttostr(counter)); //Zähler ausgeben
    zeit:= timegettime; //neue Start Zeit ermitteln
    Counter := 0;   //und Zähler wieder auf 0 setzen.
  end;
end;

procedure TForm1.WndProc(var Msg: TMessage);
var
  Point: TPoint;
begin
  if Msg.Msg = WM_TIMER  then //Nachricht abfangen
  begin
    if Msg.WParam = TIMER_ID then
    begin
```

```
      OnTimer; //Timercode ausführen
    end;
  end;
  inherited;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  SetTimer(self.Handle,TIMER_ID,1,nil); //Der Timer wurde erstellt
  Counter := 0;
  Zeit := Timegettime ;          //Die Startzeit wird ermittelt
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  killTimer(self.Handle,TIMER_ID); // das freigeben nicht vergessen
end;
```

3. Multimedia Timer

Der so genanter Multimedia Timer wird mit der Funktion TimeSetEvent aus der MCI realisiert. Der Funktion muss unter anderem die Adresse einer Procedure über geben werden. Diese Callback Procedure wird dann automatisch aufgerufen. Leider kann man der Funktion aber keine Methode übergeben, bei Methoden gibt es immer noch einen versteckten self Parameter, so das man damit nicht viel anfangen kann. Als Lösung bietet sich an in der Procedure eine Message an ein Ziel Fenster zuschicken und dann auf diese Message zu reagieren.
Minimal Intervall:
NT/XP/2000 = 1
95/98 = 1
Auch hier wieder ein Beispiel mit dem ich die maximale Geschwindigkeit des Timers ermittelt habe.

```
unit U_Main;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ExtCtrls, mmsystem; //!!!!  mmsystem einbinden (für timegettime)

type
  TForm1 = class(TForm)
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private-Deklarationen }
  public
    { Public-Deklarationen }
    Zeit: int64;
    Counter: integer; //Eine Variable zum Zählen der Timer Durchläufe
    Timerid: integer;
    procedure OnTimer;
    procedure WndProc(var Msg: TMessage); override;
  end;

const
  Time_Elapsed_Msg = WM_User + 2;//Das +2 ist frei wählbar
var
  Form1: TForm1;

implementation
```

```
{$R *.dfm}

procedure TForm1.OnTimer;
begin
  inc(Counter);   //Zähler um 1 erhöhen
  IF timegettime-Zeit>1000 then //Jeweils nach einer Sekunde
  begin
    Canvas.TextOut(10,10,inttostr(counter)); //Zähler ausgeben
    zeit:= timegettime; //neue Start Zeit ermitteln
    Counter := 0;   //und Zähler wieder auf 0 setzen.
  end;
end;

procedure TimeCallBack(TimerID, Msg: Uint; dwUser, dw1, dw2: DWORD); stdcall; pascal;
begin
  postmessage(HWnd(dwUser),Time_Elapsed_Msg,0,0);//eine Message wird verschickt
end;

procedure TForm1.WndProc(var Msg: TMessage);
var
  Point: TPoint;
begin
  if Msg.Msg = Time_Elapsed_Msg  then //Nachricht abfangen
  begin
    OnTimer; //Timercode ausführen
  end;
  inherited;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Timerid := TimeSetEvent(1, 0, @TimeCallBack, self.Handle, TIME_PERIODIC);
  //Der Timer wurde erstellt
  Counter := 0;
  Zeit := Timegettime ;    //Die Startzeit wird ermittelt
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  timeKillEvent(Timerid); //Timer freigeben
end;

end.
```

Nachtrag: Es gibt eine alternative Lösung: http://www.michael-puff.de/Artikel/2007_08_02_CallbackMethod.php die ich aber nicht getestet habe.

4. Thread Timer

Ich möchte jetzt nicht noch erklären was Thread sind. Dafür gibt es andere Tutorials. Nur soviel zum Thema: mit Threads kann code quasi "parallel" zur Anwendung(main Thread) ausgeführen lassen. Bei den Thread Timern gibt es 2 Varianten. Bei der ersten wird in execute ein sleep(intervall) eingebaut. Durch das sleep wird die Anwendung nicht selber nicht pausiert da es ja in einem Thread ausgeführt wird. Der Thread pausiert also eine gewünschte Zeit lang und führt dann den Timer code aus. Der Vorteil ist hierbei, dass die CPU Belastung geringe bleibt. Der Nachteil gegen über der zweiten Variante ist das maximale Intervall von 1 ms. Bei der zweiten Variante wird in execute durch eine einfache IF Abfrage überprüft ob eine bestimmte Zeit abgelaufen ist. Der Nachteil hierbei ist, dass durch die ständige Abfrage eine CPU Belastung von 100% entsteht. Allerdings hat man so einen sehr genauen Timer hat.

Und wieder Die Beispiele:

Sleep Variante:

```
unit U_Main;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ExtCtrls, mmsystem, StdCtrls; //!!!!  mmsystem einbinden (für timegettime)

type
  TThreadTimer = class(TThread)
  private
    FCanvas: TCanvas;
    Zeit: int64;
    Counter: integer; //Eine Variable zum Zählen der Timer durchläufe
  published
    procedure Execute; override;
    procedure OnTimer;
    constructor CreateMitCanvas(NewCanvas: TCanvas); virtual;
  end;

  TForm1 = class(TForm)
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private-Deklarationen }
  public
    { Public-Deklarationen }
    ThreadTimer: TThreadTimer;
    Timerid: integer;
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

//Dem constructor wird eine zeichenfläche übergeben,
//damit klar ist wohin gezeichnet werden soll
constructor TThreadTimer.CreateMitCanvas(NewCanvas: TCanvas);
begin
  inherited create(true);
  FCanvas := NewCanvas;
  Counter := 0;
  Zeit := Timegettime ;          //Die Startzeit wird ermittelt
  resume;
end;

procedure TThreadTimer.Execute;
begin
  while not Terminated do
  begin
    sleep(1);
    OnTimer;
  end;
end;

procedure TThreadTimer.OnTimer;
begin
```

```
  inc(Counter);   //Zähler um 1 erhöhen
  if timegettime-Zeit>1000 then //Jeweil nach einer Sekunde
  begin
    FCanvas.Lock;              //locken wegen syncronisation
    FCanvas.Rectangle(0,0,0,0);
    FCanvas.TextOut(10,10,inttostr(counter)); //Zähler ausgeben
    FCanvas.Unlock;

    zeit:= timegettime; //neue Start Zeit ermitteln
    Counter := 0;   //und Zähler wieder auf 0 setzen.
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ThreadTimer := TThreadTimer.CreateMitCanvas(canvas);
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  ThreadTimer.free; //Timer freigeben
end;

end.
```

High Performance Variante:

```
unit U_Main;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ExtCtrls, mmsystem; //!!!!  mmsystem einbinden (für timegettime)

type
  TThreadTimer = class(TThread)
  private
    FCanvas: TCanvas;
    Zeit: int64;
    Counter: integer; //Eine Variable zum Zählen der Timer durchläufe
    aktuelleZeit,altezeit: int64;
  published
    procedure Execute; override;
    procedure OnTimer;
    constructor CreateMitCanvas(NewCanvas: TCanvas); virtual;
  end;

  TForm1 = class(TForm)
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private-Deklarationen }
  public
    { Public-Deklarationen }
    ThreadTimer: TThreadTimer;
    Timerid: integer;
  end;

var
  Form1: TForm1;
```

```
implementation

{$R *.dfm}

constructor TThreadTimer.CreateMitCanvas(NewCanvas: TCanvas);
begin
  inherited create(true);
  NewCanvas.Pen.Color := clred;
  NewCanvas.Rectangle(0,0,100,100);

  FCanvas := NewCanvas;
  Counter := 0;
  Zeit := Timegettime ;          //Die Startzeit wird ermittelt
  resume;
  QueryPerformanceCounter(altezeit);
end;

procedure TThreadTimer.Execute;
begin
  while not Terminated do
  begin
    QueryPerformanceCounter(aktuelleZeit);
    if (aktuelleZeit-altezeit)>0 then
    begin //hier könnte man ein Intervall einstellen
      ontimer;
      QueryPerformanceCounter(altezeit);
    end;
  end;
end;

procedure TThreadTimer.OnTimer;
begin
  inc(Counter);   //Zähler um 1 erhöhen
  if timegettime-Zeit>1000 then //Jeweil nach einer Sekunde
  begin
    FCanvas.Lock;
    FCanvas.Rectangle(0,0,0,0);
    FCanvas.TextOut(10,10,inttostr(counter)); //Zähler ausgeben
    FCanvas.Unlock;
    zeit:= timegettime; //neue Start Zeit ermitteln
    Counter := 0;   //und Zähler wieder auf 0 setzen.
  end;
end;


procedure TForm1.FormCreate(Sender: TObject);
begin
  ThreadTimer := TThreadTimer.CreateMitCanvas(canvas);
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  ThreadTimer.free; //Timer freigeben
end;

end.
```

Man bekommt hier aber leicht Probleme bei der Synchronisation also wenn man in 2 Threads auf den gleichen Speicherbereich zugreift.

5. OnIdle Timer

Das Prinzip ist vergleichbar mit dem Thread Timer nur das hier das OnIdle genutzt wird. Das OnIdle Ereignis wird immer aufgerufen wenn die Anwendung nichts macht.
Sleep sollte man allerdings nicht benutzen da sonst die gesamte Anwendung pausieren würde. Der Timer von DelphiX ist Beispielsweise so ein Onidle Timer. Wichtig ist auch, dass man in OnIdle Ereignis done auf false setzt damit OnIdle nicht nur einmal aufgerufen wird. Ein großer Nachteil ist, dass solch eine Verwendung des OnIdle Ereignisses zu 100% CPU Belastung führt. Es biete sich also eigentlich nur an wenn das Programm sowieso alle freien Ressourcen nutzen soll wie beispielsweise ein Spiel.

Jetzt folgt das Beispiel (Funktion ist wie gehabt):

```
unit U_Main;


interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, AppEvnts, mmsystem;

type
  TForm1 = class(TForm)
    procedure FormCreate(Sender: TObject);
  private
    { Private-Deklarationen }
  public
    { Public-Deklarationen }
    Zeit: int64;
    Counter: integer; //Eine Variable zum Zählen der Timer durchläufe
    procedure OnIdle(Sender: TObject; var Done: Boolean);
    procedure OnTimer;
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.OnIdle(Sender: TObject; var Done: Boolean);
begin
  done:=false;
  OnTimer;
end;

procedure TForm1.OnTimer;
begin
  inc(Counter);   //Zähler um 1 erhöhen
  if timegettime-Zeit>1000 then //Jeweil nach einer Sekunde
  begin
    Canvas.TextOut(10,10,inttostr(counter)); //Zähler ausgeben
    zeit:= timegettime; //neue Start Zeit ermitteln
    Counter := 0;   //und Zähler wieder auf 0 setzen.
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Application.OnIdle := OnIdle;
```

```
end;

end.
```

Timer und Spiele (time based movement)

Nun möchte ich noch kurz zu einem Spezialfall bei Spielen eingehen. Das hat zwar nur am Rande was mit Timern zu tun passt aber auch zum Thema.
Wenn man Figuren oder vergleichbares bewegen will könnte man ja auf die Idee kommen das in einem Timer zu machen. Also beispielsweise pro Timer Durchlauf: Figur.left:=Figur.left+1 oder ähnliches. Das Problem dabei ist nur, dass man sich nie sicher sein kann, ob das Ontimer event auch aufgerufen wird. Dadurch kann es dazu kommen, dass das Spiel auf unterschiedlichen Rechnern unterschiedlich schnell läuft. Als Lösung dieses Problems kann man dem Objekt eine Geschwindigkeit zuzuordnen z.B.: 100 Pixel in der Sekunde. Dann muss nur noch die Zeit die Seit dem letzten Timer Durchlauf vergangen ist ermittelt werden. (Stichwort Uptime) Jetzt kann man auch die Strecke berechnen die das Objekt zurückgelegt hat. (v=s/t)
Das Spiel sollte jetzt auf allen Rechner gleich schnell laufen es kann nur sein, dass es auf manchen (alten) Rechnern ruckelt.

Fazit

Jetzt da ich 5 verschiedenen Varianten erläutert habe stellt sich die Frage welche Variante die Beste ist. Leider kann man das so pauschal nicht sagen jede Variante hat ihre Vor und Nachteile. Wenn man nur mal schnell einen Timer braucht und es nicht auch eine große Genauigkeit ankommt ist der VCL Timer die richtige Wahl. Die direkte Nutzung der Windows Api (Non VCL Timer) halte ich für überflüssig, da sich keine Vorteile gegenüber dem VCL Timer ergeben. Einzig sinnvoll wäre der Einsatz in Non-VCL Programmen.
Will man einen Timer mit dem man ein Genauigkeit von einer Millisekunde bietet sich ein Multimedia Timer oder Thread Timer an. Beide Varianten sind etwas schwerer zu programmieren und beim Thread ergibt sich noch das Problem der Synchronisation.
Will man einen sehr genauen Timer so bietet sich der Thread Timer mit einer hohen Priorität an. Ein OnIdle Timer ist eigentlich nur sinnvoll wenn man das Onidle Event eh schon verwendet z.B zum rendern in einem Spiel.
Allgemein kann man sagen, dass man sich nie sicher sein kann das OnTimer Events ausgeführt wird da Windows ein Multi-Prozess-Betriebssystem ist. Andere Programme können so viele Ressourcen verbrauchen, dass für das eigene Programm nicht mehr "genug" Ressourcen vorhanden sind. Bei den VCL,Non VCL, Multimedia und Thread Timer, kommt als zusätzliche Unsicherheit noch hinzu, dass das Timer Ereignis über eine Nachricht ausgelöst wird. (Beim Thread Timer nur wenn man die Methode Synchronize verwendet wird) Da es aber immer mehrere Nachrichten gibt, kann man sich nie sicher sein, dass das Timer Ereignis auch wirklich dann ausgeführt wird wenn es ausgeführt werden soll.

Zum Schluss noch eine kleine Übersicht.

	VCL Timer	**Non VCL Timer**	**Multimedia Timer**	**Thread Timer (sleep/Maximal)**	**OnIdle Timer**
Min. Intervall	55	55	1	0,0025*	0,005*
Vorteil	Einfach zu programmieren/ bedienen	nicht auf VCL angewiesen	Geschwindigkeit reicht für die meisten Anwendungen, geringe CPU Belastung	Unabhängig von Messages. wird nicht durch Main Prozess beeinträchtigt	Ausführsicherheit
Nachteil	auf 98 und XP unterschiedlich schnell, geringe Geschwindigkeit	wie bei VCL Timer und aufwendiger zu programmieren	keine Ausführsicherheit	Synchronisation, erhöhter Programmieraufwand	100% CPU Belastung

*abhängig von verfügbaren Ressourcen (Zahlen sind von einem Pentium M 1400)

So und damit bin ich soweit fertig. Ich hoffe es war informativ und verständlich. Konstruktive Kritik Verbesserungs Vorschläge und ähnliches sind natürlich gerne gesehen.

mfg Henning Brackmann (aka Gandalfus)