Daemon alternatives Beispiel

Hier nun die Variante in der FreeOnTerminate des Threads auf false steht.

thedaemon.lpr (/code/thedaemon_alt.lpr?mode=download) Pascal (5,75 kByte) 02.12.2016 13:32

```
//
******************************************************************************

//  Title.............. :  The Daemon Example
//
//  Modulname ......... :  thedaemon.lpr (project file)
//  Type .............. :  Unit
//  Author ............ :  Udo Schmal
//  Development Status  :  01.12.2016
//  Operating System .. :  Win32/Win64
//  IDE ............... :  Lazarus
//
******************************************************************************

program thedaemon;

{$mode objfpc}{$H+}

uses
  HeapTrc,
{$IFDEF UNIX}{$IFDEF UseCThreads}
  CThreads,
{$ENDIF} Cmem,{$ENDIF}
  Classes, SysUtils, EventLog, DaemonApp;

type
  TTheThread = class(TThread)
    procedure Execute; override;
    destructor Destroy; override;
  end;

  TTheDaemon = class(TCustomDaemon)
  private
    FThread: TTheThread;
  public
    procedure ThreadStopped(Sender: TObject);
    function Install: boolean; override;
    function UnInstall: boolean; override;
    function Start: boolean; override;
    function Stop: boolean; override;
    function Pause: boolean; override;
    function Continue: boolean; override;
    function Execute: boolean; override;
    function ShutDown: boolean; override;
  end;
```

```
  TTheDaemonMapper = class(TCustomDaemonMapper)
  public
    constructor Create(AOwner: TComponent); override;
    procedure ToDoOnInstall(Sender: TObject);
    procedure ToDoOnRun(Sender: TObject);
    procedure ToDoOnUninstall(Sender: TObject);
    procedure ToDoOnDestroy(Sender: TObject);
  end;

function BoolToStr(AVal: Boolean): String;
begin
  if AVal = True then result := 'true' else result := 'false';
end;

procedure TTheThread.Execute;
var i: integer;
begin
  i := 0;
  Application.Log(etDebug, 'Thread.Execute');
  try
    repeat
      Sleep(1000); //milliseconds
      inc(i);
      Application.Log(etDebug, 'Thread.Loop ' + Format('Tick :%d', [i]));
    until Terminated;
  finally
    Application.Log(etDebug, 'Thread.LoopStopped');
    OnTerminate(self);
  end;
end;

destructor TTheThread.Destroy;
begin
  Application.Log(etDebug, 'Thread.Destroy');
  inherited Destroy;
end;

{$REGION ' - Daemon - '}
procedure TTheDaemon.ThreadStopped(Sender: TObject);
begin
  Application.Log(etDebug, 'Daemon.ThreadStopped');
  if FThread <> nil then
    FreeAndNil(FThread);
end;

function TTheDaemon.Install: boolean;
begin
  result := inherited Install;
  Application.Log(etDebug, 'Daemon.installed: ' + BoolToStr(result));
```

```
end;

function TTheDaemon.UnInstall: boolean;
begin
  result := inherited UnInstall;
  Application.Log(etDebug, 'Daemon.Uninstall: ' + BoolToStr(result));
end;

function TTheDaemon.Start: boolean;
begin
  result := inherited Start;
  Application.Log(etDebug, 'Daemon.Start: ' + BoolToStr(result));
  FThread := TTheThread.Create(true);
  FThread.OnTerminate := @ThreadStopped;
  FThread.FreeOnTerminate := false;
  FThread.Resume;
end;

function TTheDaemon.Stop: boolean;
begin
  Application.Log(etDebug, 'Daemon.Stop');
  FThread.Terminate;
  repeat
    sleep(1000);
  until FThread=nil;
  result := inherited Stop;
  Application.Log(etDebug, 'Daemon.Stop: ' + BoolToStr(result));
end;

function TTheDaemon.Pause: boolean;
begin
  result := inherited Pause;
  Application.Log(etDebug, 'Daemon.Pause: ' + BoolToStr(result));
  FThread.Suspend;
end;

function TTheDaemon.Continue: boolean;
begin
  result := inherited Continue;
  Application.Log(etDebug, 'Daemon.Continue: ' + BoolToStr(result));
  FThread.Resume;
end;

function TTheDaemon.Execute: boolean;
begin
  result := inherited Execute;
  Application.Log(etDebug, 'Daemon.Execute: ' + BoolToStr(result));
end;

function TTheDaemon.ShutDown: boolean;
```

```
begin
  result := inherited ShutDown;
  Application.Log(etDebug, 'Daemon.ShutDown: ' + BoolToStr(result));
end;
{$ENDREGION}

{$REGION ' - DaemonMapper - '}
constructor TTheDaemonMapper.Create(AOwner: TComponent);
begin
  Application.Log(etDebug, 'DaemonMapper.Create');
  inherited Create(AOwner);
  with DaemonDefs.Add as TDaemonDef do
  begin
    DaemonClassName := 'TTheDaemon';
    Name := 'theDaemon';
    Description := 'The Daemon Exsample';
    DisplayName := 'The Daemon';
    RunArguments := '--run';
    Options := [doAllowStop,doAllowPause];
    Enabled := true;
    with WinBindings do
    begin
      StartType := stBoot;
      WaitHint := 0;
      IDTag := 0;
      ServiceType := stWin32;
      ErrorSeverity := esNormal;//esIgnore;
    end;
//    OnCreateInstance := ?;
    LogStatusReport := false;
  end;
  OnInstall := @Self.ToDoOnInstall;
  OnRun := @Self.ToDoOnRun;
  OnUnInstall := @Self.ToDoOnUninstall;
  OnDestroy := @Self.ToDoOnDestroy;
  Application.Log(etDebug, 'DaemonMapper.Createted');
end;

procedure TTheDaemonMapper.ToDoOnInstall(Sender: TObject);
begin
  Application.Log(etDebug, 'DaemonMapper.Install');
end;

procedure TTheDaemonMapper.ToDoOnRun(Sender: TObject);
begin
  Application.Log(etDebug, 'DaemonMapper.Run');
end;

procedure TTheDaemonMapper.ToDoOnUnInstall(Sender: TObject);
begin
```

```
  Application.Log(etDebug, 'DaemonMapper.Uninstall');
end;

procedure TTheDaemonMapper.ToDoOnDestroy(Sender: TObject);
begin
  //doesn't comes here
  Application.Log(etDebug, 'DaemonMapper.Destroy');
end;
{$ENDREGION}

{$R *.res}

begin
  RegisterDaemonClass(TTheDaemon);
  RegisterDaemonMapper(TTheDaemonMapper);
  RegisterDaemonApplicationClass(TCustomDaemonApplication);
  heaptrc.SetHeapTraceOutput(ChangeFileExt(ParamStr(0), '.heap'));
  with Application do
  begin
    Title := 'Daemon Application';
    EventLog.LogType := ltFile;
    EventLog.DefaultEventType := etDebug;
    EventLog.AppendContent := true;
    EventLog.FileName := ChangeFileExt(ParamStr(0), '.log');
    Initialize;
    Run;
  end;
end.
```

Autor: Udo Schmal (#udo.schmal) , veröffentlicht: 01.12.2016, letzte Änderung: 06.09.2023

(/)