Daemon - Sevice

Content management system - Daemon

Hier kommt nicht die Lazarus Komponente zum Einsatz sondern direkt die Free Pascal Klasse.

Der Hintergrund Service erledigt wiederkehrende Aufgaben und erkennt selber, ob überhaupt welche anliegen:

- einmal am Tag überprüft er, ob alte Logfiles (älter als 7 Tage) im Verzeichnis **/logging** liegen und löscht diese
- löscht Session-Files die älter als einer Stunde sind im Verzeichnis **/sessions**, sorgt also für das Einhalten des Session-Timeouts
- überwacht das Verzeichnis **/www/css** ob StyleSheets neu hinzugefügt oder geändert wurden und legt sofort eine minimized Version an, die dann auch an Stelle des original StyleSheet vom ISAPI-Filter ausgeliefert werden kann
- überwacht das Verzeichnis **/www/scripts** ob JavaScripts neu hinzugefügt oder geändert wurden und legt sofort eine minimized Version an, die dann auch an Stelle des original JavaScript vom ISAPI-Filter ausgeliefert werden kann
- überwacht das Verzeichnis **/www/code** ob neue Code-Dateien hinzugefügt oder geändert wurden und legt sofort eine HTML-Codierte Version mit Syntax highlighting an, um sie im Seiteninhalt einbetten zu können
- einmal am Tag wird überprüft, ob eine neue browscap.ini auf browscap.org (https://browscap.org/) zur Verfügung steht und gegebenenfalls die Datei **browscap.ini** im Verzeichnis **/index** aktualisiert
- überwacht das Verzeichnis **/index/gocher** und legt im Verzeichnis **/index/gocher-pdf** die PDF-Version ab
- überwacht das Verzeichnis **/www/downloads** und legt im Verzeichnis **/index/gocher-downloads** die HTML-Version für die Volltextsuche ab
- überwacht das Verzeichnis **/www/media** und optimiert geänderte oder neue Bilder

Also im allgemeinen Optimierungs- und Aktualisierungsaufgaben.

daemon.lpr Pascal (23,15 kByte) 14.07.2013 12:15

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


//  Title.............. :  ISAPI CMS Daemon
//
//  Modulname ......... :  cmsdaemon.lpr (project file)
//  Type .............. :  Unit
//  Author ............ :  Udo Schmal
//  Development Status  :  01.11.2012
//  Operating System .. :  Win32/Win64
//  IDE ............... :  Lazarus
//
********************************************************************************
```

```
program cmsdaemon;

{$mode objfpc}{$H+}

{ TODO 1 -ous : load project ini-File for settings }
{ TODO 2 -ous : reloade ini-File if changed }
{ TODO 3 -ous : create XML-File for image optimisation }
{ TODO 4 -ous : manage status flags }
{ TODO 5 -ous : send newsletters }
{ TODO 6 -ous : Webserver logfile analysis }
{ TODO 7 -ous : import data packages }

uses
{$IFDEF UNIX}{$IFDEF UseCThreads}
  CThreads,
{$ENDIF} Cmem,{$ENDIF}
  Windows, Classes, SysUtils, EventLog, DateUtils, DaemonApp,
  UniParser, GMTUtils, {dExif,} CSS, JS, code2html, Inet, usIndexer;

type
  TCmsThread = class(TThread)
  private
    FList: TStringList;
    FLocalPath, FProj, FHost: string;
  public
    procedure Execute; override;
    property LocalPath: string read FLocalPath write FLocalPath;
    property Proj: string read FProj write FProj;
    property Host: string read FHost write FHost;
  end;

  TCmsDaemon = class(TCustomDaemon)
  private
    FThread: TCmsThread;
  public
    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;

  TCmsDaemonMapper = 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;

var
  path: string;

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

function WinExecAndWait(const ACmd: string; wVisibility: word = SW_HIDE):
boolean;
var si : TStartUpInfo;
    pi : TProcessInformation;
    Proc: THandle;
begin
  Application.Log(etDebug, 'WinExecAndWait: ' + ACmd);
  result := false;
  FillChar(si, SizeOf(TStartUpInfo), 0);
  with si do
  begin
    cb := SizeOf(TStartUpInfo);
    dwflags := STARTF_USESHOWWINDOW or STARTF_FORCEONFEEDBACK;
    wShowWindow := wVisibility;
  end;
  if CreateProcess(nil,
                   PChar(ACmd),           {pointer to command line string}
                   nil,                   {pointer to process security
attributes}
                   nil,                   {pointer to thread security
attributes}
                   true,                  {handle inheritance flag}
                   Normal_Priority_Class, {creation flags}
                   nil,                   {pointer to new environment block}
                   nil,                   {pointer to current directory name}
                   si,                    {pointer to STARTUPINFO}
                   pi) then               {pointer to PROCESS_INF}
  begin
    Proc := pi.HProcess;
    CloseHandle(pi.hThread);
    if WaitForSingleObject(Proc, Infinite) <> Wait_Failed then result :=
true;
    CloseHandle(Proc);
  end;
end;

procedure TCmsThread.Execute;
  procedure Indexer();
```

```
  var
    Indexer: TIndexer;
    Index: TIndexFile;
  begin
    Indexer := TIndexer.Create(nil);
    try
      Index := TIndexFile.Create(nil);
      if FileExists(FLocalPath + 'index' + PathDelim + FProj + '.dat') then
        SysUtils.DeleteFile(FLocalPath + 'index' + PathDelim + FProj +
'.dat');
      Index.FileName := FLocalPath + 'index' + PathDelim + FProj + '.dat';
      Index.Connect;
      Index.WriteOnCommit:=True;
      Indexer.Index := Index;
      Indexer.FileMask := '*.html'; //semicolon separated list
      Indexer.SearchRecursive := true;
      IgnoreListManager.LoadIgnoreWordsFromFile('de', FLocalPath +
'de.txt');
      Indexer.Language := 'de';
      Indexer.UseIgnoreList := true;
      Indexer.SearchPath := FLocalPath + 'index' + PathDelim + FProj;
      Application.Log(etInfo, 'CmsThread.Indexing: ' +
IntToStr(Indexer.Execute()));
      Indexer.SearchPath := FLocalPath + 'index' + PathDelim + FProj + '-
downloads';
      Application.Log(etInfo, 'CmsThread.Indexing: ' +
IntToStr(Indexer.Execute()));
    finally
      Indexer.Index.free;
      FreeAndNil(Indexer);
    end;
  end;

  procedure Minimizer(minPath: string);
  var
    SearchRec: TSearchRec;
    sExt: string;
  begin
    if FindFirst(minPath + '*.*', faAnyFile, SearchRec) = 0 then
    repeat
      if ((SearchRec.FindData.dwFileAttributes and faDirectory) <> 0) and
         (SearchRec.Name<>'.') and (SearchRec.Name<>'..') then
        Minimizer(minPath + SearchRec.Name + PathDelim)
      else
      begin
        sExt := ExtractFileExt(SearchRec.Name);
        if ((lowercase(sExt)='.css') or (lowercase(sExt)='.js')) then
          if lowercase(ExtractFileExt(ChangeFileExt(SearchRec.Name, ''))) <>
'.min' then
            if not FileExists(minPath + ChangeFileExt(SearchRec.Name,
```

```
'.min'+sExt)) or
              (FileTimeGMT(SearchRec)<>FileTimeGMT(minPath +
ChangeFileExt(SearchRec.Name, '.min'+sExt))) then
            begin
              Application.Log(etInfo, 'Thread.Execute: minimize ' + minPath +
SearchRec.Name );
              if (lowercase(sExt)='.css') then
                CSSMinFile(minPath + SearchRec.Name)
              else //if (lowercase(sExt)='.js') then
                JSMinFile(minPath + SearchRec.Name);
              FileTimeCopy(minPath + SearchRec.Name, minPath +
ChangeFileExt(SearchRec.Name, '.min'+sExt));
              Application.Log(etDebug, 'CmsThread.Execute: ' + sExt + 'min: '
+ minPath + SearchRec.Name);
            end;
      end;
    until FindNext(SearchRec) <> 0;
    SysUtils.FindClose(SearchRec);
  end;

  procedure CodeToHtml(codePath: string);
  var
    SearchRec: TSearchRec;
    MemoryStream: TMemoryStream;
    sExt: string;
    sText: RawByteString;
  begin
    if FindFirst(codePath + '*.*', faAnyFile, SearchRec) = 0 then
    repeat
      if ((SearchRec.FindData.dwFileAttributes and faDirectory) <> 0) and
         (SearchRec.Name<>'.') and (SearchRec.Name<>'..') then
        CodeToHtml(codePath + SearchRec.Name + PathDelim)
      else
      begin
        sExt := ExtractFileExt(SearchRec.Name);
        if pos(lowercase(sExt), '.lpr.pas.pp.js.css.htm')>0 then
          if not FileExists(codePath + SearchRec.Name + '.html') or
            (FileTimeGMT(SearchRec)<>FileTimeGMT(codePath + SearchRec.Name +
'.html')) then
          begin
            sText := ConvertCodeToHtml(codePath + SearchRec.Name);
            if (Length(sText) > 0) then
            begin
              sText :=
'<html>'#13#10'<head>'#13#10'<title>'+SearchRec.Name+'</title>'#13#10'</head
>'#13#10'<body>'+sText+'</body>'#13#10'</html>';
              MemoryStream := TMemoryStream.Create;
              try
                MemoryStream.WriteBuffer(sText[1], Length(sText));
                MemoryStream.SaveToFile(codePath + SearchRec.Name + '.html');
```

```
{$ifdef info}WriteLog(etInfo, 'CodeToHtml: save html Version (' +
ChangeFileExt(AFilename, sExt + '.html') + ')');{$endif}
              finally
                MemoryStream.Free;
              end;
            end;
          end;
      end;
    until FindNext(SearchRec) <> 0;
    SysUtils.FindClose(SearchRec);
  end;

{  procedure CheckImage(AFilename: string);
  var
    ImgData: TImgData;
    dt: TDateTime;
  begin
    if FileExists(AFilename) then
    begin
      ImgData := TimgData.Create;
      try
        ImgData.BuildList := GenAll;
        ImgData.ProcessFile(AFilename);
        dt := ImgData.ExifObj.GetImgDateTime;
        SetCreationTime(AFilename, dt);
      finally
        FreeAndNil(ImgData);
      end
    end;
  end;
}
  function FindTextInFile(FullPathName, TextToFind: string): boolean;
  var f: textfile;
      line, lcTextToFind: string;
  begin
    lcTextToFind := LowerCase(TextToFind);
    result := false;
    assignfile(f, FullPathName);
    reset(f);
    while (not eof(f)) and (not result) do
    begin
      readln(f, line);
      line := lowercase(line);
      result := (pos(lcTextToFind, line) > 0);
    end;
    closefile(f);
  end;

var
  i, iDay: integer;
```

```
  cssPath, jsPath, xinhaPath, imagesPath, downloadsPath, idxPath, pdfPath,
idxdownloadsPath, codePath, sFile, s: string;
  SearchRec: TSearchRec;
  SEOFile: TStringList;
  dt: TDateTime;
  optimize, changed: boolean;
//  fChangeNotify : DWORD;
begin
  //inherited Execute;
  Application.Log(etDebug, 'CmsThread.Execute');
  try
    FList := TStringList.Create;
    cssPath := FLocalPath + 'wwwroot' + PathDelim + 'styles' + PathDelim;
    jsPath := FLocalPath + 'wwwroot' + PathDelim + 'scripts' + PathDelim;
    xinhaPath := FLocalPath + 'wwwroot' + PathDelim + 'Xinha' + PathDelim;
    downloadsPath := FLocalPath + 'wwwroot' + PathDelim + 'downloads' +
PathDelim + FProj + PathDelim;
    imagesPath := FLocalPath + 'wwwroot' + PathDelim + 'images' + PathDelim
+ FProj + PathDelim;
    idxPath := FLocalPath + 'index' + PathDelim + FProj + PathDelim;
    pdfPath := FLocalPath + 'index' + PathDelim + FProj + '-pdf' +
PathDelim;
    idxdownloadsPath := FLocalPath + 'index' + PathDelim + FProj + '-
downloads' + PathDelim;
    codePath := FLocalpath + 'wwwroot' + PathDelim + 'code' + PathDelim;
    iDay := 0;

    FList.LoadFromFile(FLocalpath + 'index' + PathDelim + FProj + '-
files.txt');
    repeat
      Sleep(5000); //milliseconds
      changed := false;

      if iDay <> DayOf(Now()) then
      begin
//        Application.Log(etInfo, 'CmsThread.Execute: delete old logfiles');
        Application.EventLog.Active := false;
        Application.EventLog.FileName := path + 'logging' + PathDelim +
ChangeFileExt(ExtractFileName(ParamStr(0)), FormatDateTime('dd"."mm"."yyyy',
Now) + '.log');
        Application.EventLog.Active := true;
        if FindFirst(FLocalPath + 'logging' + PathDelim + '*.log', 0,
SearchRec) = 0 then
        repeat
          dt := FileTimeGMT(SearchRec);
          if (dt+7)<NowGMT() then
          begin
            sFile := FLocalPath + 'logging' + PathDelim + SearchRec.Name;
            Application.Log(etINfo, 'CmsThread.Exceute: delete: ' + sFile + '
' + DateTimeToString(GMTToLocalTime(dt), tsLog));
```

```
            DeleteFile(sFile);
          end;
        until FindNext(SearchRec) <> 0;
        FindClose(SearchRec);

        dt := FileTimeGMT(FLocalPath + 'index' + PathDelim +
'browscap.ini');
        if (dt+1)<NowGMT() then
        begin
//          Application.Log(etInfo, 'CmsThread.Execute: get browscap.ini');
          sFile := FLocalPath + 'index' + PathDelim + 'browscap~.ini';
//          if Download('http://browsers.garykeith.com/stream.asp?
BrowsCapINI', sFile) then
          if Download('http://tempdownloads.browserscap.com/stream.php?
BrowsCapINI', sFile) then
          begin
            if not FindTextInFile(sFile, '<html') then
            begin
              Application.Log(etInfo, 'CmsThread.Execute: update: ' +
FLocalPath + 'index' + PathDelim + 'browscap.ini');
              CopyFile(PChar(sFile), PChar(FLocalPath + 'index' + PathDelim +
'browscap.ini'), false);
            end
            else
              Application.Log(etError, 'CmsThread.Execute: update: ' +
FLocalPath + 'index' + PathDelim +'browscap.ini failed!');
            DeleteFile(sFile);
          end;
        end;
        iDay := DayOf(Now());
      end;

//      Application.Log(etInfo, 'CmsThread.Execute: delete session files
after session timeout');
      if FindFirst(FLocalPath + 'sessions' + PathDelim + '*.ini', 0,
SearchRec) = 0 then
      repeat
        dt := FileTimeGMT(SearchRec);
        if (dt+(1/24))<NowGMT() then //session timeout: one hour
        try
          sFile := FLocalPath + 'sessions' + PathDelim + SearchRec.Name;
          Application.Log(etInfo, 'CmsThread.Exceute: delete: ' + sFile + ' '
+ DateTimeToString(GMTToLocalTime(dt), tsLog));
          DeleteFile(sFile);
        except
          Application.Log(etError, 'CmsThread.Exceute: delete: ' + sFile + '
' + DateTimeToString(GMTToLocalTime(dt), tsLog));
        end;
      until FindNext(SearchRec) <> 0;
      FindClose(SearchRec);
```

```
//      Application.Log(etInfo, 'CmsThread.Execute: js & css minimizer');
      Minimizer(cssPath);
      Minimizer(jsPath);
      Minimizer(xinhaPath);
      CodeToHtml(codePath);
//      Application.Log(etInfo, 'CmsThread.Execute: HTMLtoPDF: generate PDF
from index file');
      if FindFirst(idxPath + '*.html', 0, SearchRec) = 0 then
      repeat
        if not FileExists(pdfPath + ChangeFileExt(SearchRec.Name, '.pdf'))
or
          (FileTimeGMT(SearchRec)>FileTimeGMT(pdfPath +
ChangeFileExt(SearchRec.Name, '.pdf'))) then
        begin
          changed := true;
          Application.Log(etInfo, 'CmsThread.Execute: create PDF: ' +
pdfPath + ChangeFileExt(SearchRec.Name, '.pdf'));
          WinExecAndWait(FLocalPath + 'htmltopdf.exe --ignore-load-errors' +
                         ' "http://' + FHost + '/' +
ChangeFileExt(SearchRec.Name, '') + '?
media=print&name=pdfgen&password=mypassw"' +
                         ' "' + pdfPath + ChangeFileExt(SearchRec.Name,
'.pdf'));
          FileTimeCopy(idxPath + SearchRec.Name, pdfPath +
ChangeFileExt(SearchRec.Name, '.pdf'));
        end;
      until FindNext(SearchRec) <> 0;
      FindClose(SearchRec);

      if FindFirst(pdfPath + '*.pdf', 0, SearchRec) = 0 then
      repeat
        if not FileExists(idxPath + ChangeFileExt(SearchRec.Name, '.html'))
then
        begin
          changed := true;
          Application.Log(etInfo, 'CmsThread.Execute: delete: ' + pdfPath +
SearchRec.Name);
          DeleteFile(pdfPath + SearchRec.Name);
        end;
      until FindNext(SearchRec) <> 0;
      FindClose(SearchRec);

      if FindFirst(downloadsPath + '*.pdf', 0, SearchRec) = 0 then
      repeat
        if not FileExists(idxdownloadsPath + ChangeFileExt(SearchRec.Name,
'.html')) or
          (FileTimeGMT(SearchRec)>FileTimeGMT(idxdownloadsPath +
ChangeFileExt(SearchRec.Name, '.html'))) then
        begin
```

```
          Application.Log(etInfo, 'CmsThread.Execute: PDFtoHTML: create
index file: '+ downloadsPath + SearchRec.Name);
          if WinExecAndWait(FLocalPath + 'pdftohtml -i -noframes -q ' +
downloadsPath + SearchRec.Name, SW_HIDE) and
            FileExists(downloadsPath + ChangeFileExt(SearchRec.Name,
'.html')) then
          begin
            with TParser.Create do
            try
              LoadFromFile(downloadsPath + ChangeFileExt(SearchRec.Name,
'.html'));
              SaveAsUtf8File(idxdownloadsPath +
ChangeFileExt(SearchRec.Name, '.html'));
              changed := true;
            finally
              DeleteFile(downloadsPath + ChangeFileExt(SearchRec.Name,
'.html'));
              Free;
            end;
            Application.Log(etInfo, 'CmsThread.Execute: create Index: ' +
idxdownloadsPath + ChangeFileExt(SearchRec.Name, '.html'));
            FileTimeCopy(downloadsPath + SearchRec.Name, idxdownloadsPath +
ChangeFileExt(SearchRec.Name, '.html'));
          end;
        end;
      until FindNext(SearchRec) <> 0;
      FindClose(SearchRec);

//      Application.Log(etInfo, 'CmsThread.Execute: delete index file if PDF
not exists anymore');
      if FindFirst(idxdownloadsPath + '*.html', 0, SearchRec) = 0 then
      repeat
        if not FileExists(downloadsPath + ChangeFileExt(SearchRec.Name,
'.pdf')) then
        begin
          Application.Log(etInfo, 'CmsThread.Execute: delete: ' +
idxdownloadsPath + SearchRec.Name);
          DeleteFile(idxdownloadsPath + SearchRec.Name);
          changed := true;
        end;
      until FindNext(SearchRec) <> 0;
      FindClose(SearchRec);

      if changed then Indexer();

//      Application.Log(etInfo, 'Thread.Execute: optimize images');
      changed := false;
      if FindFirst(imagesPath + '*.*', faAnyFile { or faSymLink}, SearchRec)
= 0 then
      repeat
```

```
        optimize := false;
        if (SearchRec.Name<>'.') and (SearchRec.Name<>'..') then
        begin
          if ((SearchRec.FindData.dwFileAttributes and faDirectory) = 0) and
            (pos(lowercase(ExtractFileExt(SearchRec.Name)),
'.png.jpg.jpeg')>0) then
          begin
            i := FList.IndexOfName(imagesPath + SearchRec.Name);
            optimize := (i = -1);
            if not optimize then
              optimize := (StringToDateTime(FList.ValueFromIndex[i], tsGER)
<> FileTimeGMT(SearchRec));
            if optimize then
            begin
              if (lowercase(ExtractFileExt(SearchRec.Name))='.png') then
              begin
                Application.Log(etDebug, 'CmsThread.Execute: optimize PNG: '
+ imagesPath + SearchRec.Name);
                WinExecAndWait(FLocalPath + 'optipng.exe -o7 "' + imagesPath
+ SearchRec.Name + '"', SW_HIDE);
              end
              else if (pos(lowercase(ExtractFileExt(SearchRec.Name)),
'.jpg.jpeg')>0) then
              begin
                Application.Log(etDebug, 'CmsThread.Execute: optimize JPEG:
' + imagesPath + SearchRec.Name);
                WinExecAndWait(FLocalPath + 'jpegoptim.exe --strip-all "' +
imagesPath + SearchRec.Name + '"', SW_HIDE);
              end;
              if (i = -1) then
                FList.Add(imagesPath + SearchRec.Name + '=' +
DateTimeToString(FileTimeGMT(SearchRec), tsGER))
              else
                FList.ValueFromIndex[i] :=
DateTimeToString(FileTimeGMT(SearchRec), tsGER);
              changed := true;
            end;
          end;
        end;
      until FindNext(SearchRec) <> 0;
      FindClose(SearchRec);

      if changed then
      begin
        Application.Log(etDebug, 'CmsThread.Exceute: update: ' + FLocalPath
+ 'index' + PathDelim + FProj + '-files.txt ' + DateTimeToStr(dt));
        FList.SaveToFile(FLocalpath + 'index' + PathDelim + FProj + '-
files.txt');
      end;
```

```
//      Application.Log(etInfo, 'CmsThread.Execute: Webparser for SEO-
Analyse');
      if FindFirst(FLocalPath + 'seo' + PathDelim + '*.ini', 0, SearchRec) =
0 then
      repeat
        SEOFile := TStringList.Create;
        SEOFile.LoadFromFile(FLocalPath + 'seo' + PathDelim +
SearchRec.Name);
        s := ' --domain=' + SEOFile.Values['domain'] +
             ' --path=' + SEOFile.Values['path'] +
             ' --type=' + SEOFile.Values['type'];
        if SEOFile.Values['zip']='true'  then s := s + ' --zip';
        SEOFile.Values['start'] := NowToGMT(tsGER);
        WinExecAndWait(FLocalPath + 'Webparser.exe' + s, SW_HIDE);
        SEOFile.Values['end'] := NowToGMT(tsGER);
        SEOFile.SaveToFile(FLocalPath + 'seo' + PathDelim + SearchRec.Name);
        SEOFile.Free;
        RenameFile(FLocalPath + 'seo' + PathDelim + SearchRec.Name,
FLocalPath + 'seo' + PathDelim + SearchRec.Name + '.done')
      until FindNext(SearchRec) <> 0;
      FindClose(SearchRec);

    until Terminated;
    FreeAndNil(FList);
  except
    on E: Exception do
      Application.Log(etError, 'CmsThread.Execute: ' + E.Message);
  end;
end;

{$REGION ' - CmsDaemon - '}
function TCmsDaemon.Install: boolean;
begin
  result := inherited Install;
  Application.Log(etDebug, 'CmsDaemon.Installed: ' + BoolToStr(result));
end;

function TCmsDaemon.Start: boolean;
begin
  result := inherited Start;
  if not assigned(FThread) then
  begin
    FThread := TCmsThread.Create(true);
    FThread.FreeOnTerminate := true;
    FThread.LocalPath := path;
    FThread.Proj := 'gocher';
    FThread.Host := 'www.gocher.me';
    FThread.Resume;
  end;
  Application.Log(etDebug, 'CmsDaemon.Start: ' + BoolToStr(result));
```

```
end;

function TCmsDaemon.Stop: boolean;
begin
  result := inherited Stop;

  if assigned(FThread) then
  begin
    FThread.Terminate;
    FThread.WaitFor;
    FThread := nil;
  end;
  Application.Log(etDebug, 'CmsDaemon.Stop: ' + BoolToStr(result));
end;

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

function TCmsDaemon.Pause: boolean;
begin
  result := inherited Pause;
  if assigned(FThread) then
  begin
    FThread.Suspend;
    result := true;
  end;
  Application.Log(etDebug, 'CmsDaemon.Pause: ' + BoolToStr(result));
end;

function TCmsDaemon.Continue: boolean;
begin
  result := inherited Continue;
  if assigned(FThread) then
  begin
    FThread.Resume;
    result := true;
  end;
  Application.Log(etDebug, 'CmsDaemon.Continue: ' + BoolToStr(result));
end;

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

function TCmsDaemon.ShutDown: boolean;
```

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

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

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

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

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

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

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

{$R *.res}

begin
  path := ExtractFilePath(ParamStr(0));
//  heaptrc.SetHeapTraceOutput(path + 'logging' + PathDelim +
ChangeFileExt(ExtractFileName(ParamStr(0)), '.heap'));
  RegisterDaemonClass(TCmsDaemon);
  RegisterDaemonMapper(TCmsDaemonMapper);
  with Application do
  begin
    Title := 'ISAPI CMS Daemon Application';
    EventLog.LogType := ltFile;
    EventLog.DefaultEventType := etDebug;
    EventLog.AppendContent := true;
    EventLog.FileName := path + 'logging' + PathDelim +
ChangeFileExt(ExtractFileName(ParamStr(0)), FormatDateTime('dd"."mm"."yyyy',
Now) + '.log');
    Initialize;
    Run;
  end;
end.
```

basierend auf Daemon (Service)

Autor: Udo Schmal, veröffentlicht: 29.10.2012, letzte Änderung: 29.08.2024