Delphi DLL Tutorial – Wie exportiere ich Komponenten aus einer DLL?

Von Padmalcom (20.03.2009)
www.jofre.de
Nach dem Tutorial[1] von Leonardo M. Ramé

---

1 Ja, es geht wirklich!

Im Internet sind tausende Forenbeinträge zu finden in denen ausdrücklich gesagt wird, dass es nicht möglich ist eine Komponente aus einer DLL in eine Delphi-Applikation zu exportieren. Die Argumentation klingt plausibel: Eine DLL kann auch in Anwendungen geladen werden die nicht im Delphi-Dialekt geschrieben wurde (z.B. C#), jedoch ist zu beachten, dass die Objektstrukturen andere sind. Wer sich schon einmal mit DLLs beschäftigt hat, weiß, dass es schon bei Strings anfängt da lediglich PChar zu den Zeichenketten anderer Sprachen kompatibel sind. Wie soll das ganze dann bei Komponenten funktionieren? Richtig, gar nicht! Oder doch?

2 Trickserei

Zugegeben, der Lösungsansatz ist nicht der schönste und hat einen gewissen Overhead (nämlich den der Größe einer TForm). Was wir tun werden ist eine Komponente oder auch mehrere auf ein TPanel zu setzen, dann werden wir die Form auf der das Panel sitzt aus der DLL in unser Parent-Applikation laden und dem Panel einen neuen Parent geben, es quasi umtopfen. Danach kommt noch ein Kniff den ich beim Rumexperimentieren und durch einige Internetquellen herausgefunden habe und der das ganze überhaupt möglich macht (Wie der Flux-Kompensator ☺).

3 Wir programmieren

Zur Entwicklung habe ich *Delphi 7 Personal* benutzt. Diese Entwicklungsumgebung reicht völlig für unsere Zwecke aus und ist zu dem noch kostenlos. Mit etwas suchen kann man sie noch auf der polnischen Seite von Borland finden.
Anmerkung: Meine IDE ist auf Englisch gestellt, deswegen sind die Begriffe im Folgenden teilweise auf Englisch, was jedoch für einen Programmierer kein Problem darstellen sollte.

3.1 Die DLL

Zuerst erstellen wir die DLL. Das geht über *File → New →Other… Dll Wizard.* Wir speichern unser Projekt als „TestPlugin“ und sollten dann folgenden Quellcode vor uns sehen:

```
library TestPlugin;

uses SysUtils, Classes;

{$R *.res}

begin
end.
```

Nun erstellen wir unsere Form, die uns als Container dienen wird. Dazu gehen wir auf *File → New →Form* und speichern die dazu erzeugte Unit als *PForm1.pas* ab.

Aus dem Interface-Teil entfernen wir nun folgende Deklaration…

```
var Form1:TForm1;
```

… und ersetzen sie durch:

```
 TMyFormClass = class of TForm1;
function ParentForm:TFormClass; stdcall; export;
```

Damit erzeugen wir eine einfache Klasse namens TMyFormClass die unsere Form1 inklusive aller der darauf befindlichen Komponenten abbildet. Die **function ParentForm** formulieren wir nun im Implementation-Teil:

```
 function ParentForm: TFormClass; stdcall;
begin
    Result := TForm1;
end;
```

Hier geben wir einfach nur unsere Form als Result zurück.
Damit wir diese Funktion in der DLL benutzen können müssen wir in unserem DLL-Hauptfile folgende Zeile hinzufügen:

```
 exports ParentForm;
```

Nun müssen wir unsere Form füllen. Um den Anwendungen in die wir unsere DLL einbinden eine einheitliche Schnittstelle zu bieten, müssen wir eine Struktur festlegen:
Unser Container-Panel **muss** als erstes auf die Form gezogen werden, da wir später das *Control[0]* von der Form holen. Falls gewünscht wird, dass sich das Panel auf der gesamten Fläche der neuen Komponente ausbreitet, muss das *Align* des Panels auf *alClient* gesetzt werden. Auf dem Panel können wir nun unser Plugin so designen wie es später aussehen soll. Ich habe im Folgenden nur einen Button darauf platziert der eine Nachricht zeigt. Fertig gestellt sieht unser Quelltext nun so aus:

3.2 Die Library-Datei:

```
library TestPlugin;

uses SysUtils, Classes, PForm1 in 'PForm1.pas';

{$R *.res}

exports ParentForm;

begin
end.
```

3.3 Die Form-Datei:

```
unit PForm1;

interface

uses Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls, ExtCtrls;

type
  TForm1 = class(TForm)
  Panel1: TPanel;
  Button1: TButton;
  procedure Button1Click(Sender: TObject);
private
public
end;

TMyFormClass = class of TForm1;
function ParentForm:TFormClass; stdcall; export;

implementation

{$R *.dfm}

function ParentForm: TFormClass; stdcall;
begin
  Result := TForm1;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  showmessage('Hello World!');
end;

end.
```

3.4 Der Kniff mit dem Runtime-Packages

Da wir wie schon erwähnt keine Komponenten / Objekte aus einer DLL exportieren können ohne, dass wir einen Fehler bekommen (Häufig „Can not assign a tfont to a tfont“), müssen wir einen Haken bei *Project → Options… → Packages → Build with runtime packages* setzen. Danach löscht man alles aus dem Edit darunter bis auf **vcl.**
Die gleiche Prozedur muss man später an der Parent-Anwendung durchführen.

3.4 Die Parent-Anwendung

Zuerst wollen wir kurz klären wie unser Plugin / unsere Plugins in die Anwendung kommen. Ich hab das Beispiel so konzipiert, dass wir ein Pagecontrol haben auf dem wir für jedes Plugin ein Tabsheet hinzufügen und dort unser Panel aus der DLL hineinladen. Die Plugins werden dynamisch aus einem Pluginordner ausgelesen. Um alle Dateien aus einem Verzeichnis aufzulisten habe ich eine Unit namens TFindFile[2] verwendet. Allerdings lässt sich so ein Algorithmus leicht selber entwerfen oder aber man lädt nur eine einzige DLL deren Pfad man direkt angibt.

Vorbereitungen: Wir erstellen also über *File → New → Application* eine neue Anwendung und ziehen eine TPageControl darauf. In diesem Beispiel trägt dieses den wundervollen Namen **PageControl1**. Wir speichern die Unit der Form als **PMain.pas** und erstellen im Projektverzeichnis einen Ordner namens *„Plugins“.*
Dann erstellen wir in unserer Anwendung eine neue Unit namens *PPlugins*. Dort updaten wir die Uses-Klausel:

```
uses FindFile, Windows, Forms, Classes, ComCtrls, SysUtils, PMain, Dialogs, ExtCtrls;
```

FindFile muss wie schon gesagt vorher runtergeladen und ins Projektverzeichnis kopiert werden. Die Unit **PMain** ist die Unit unserer Parent-Anwendung, diese muss eingebunden werden damit die Objekte unserer Hauptform der Plugin-Unit bekannt gemacht werden.

Im Interface deklarieren wir nun 2 Prozeduren und einige Variablen und Arrays:

```
procedure loadplugins;
procedure unloadplugins;

var FLoadedForm: Array of TForm;
    FLibHandle : Array of Cardinal;
    PluginCount: Integer;
    myFindFile : TFindFile;
```

Loadplugins: Diese Prozedur wird dafür verantwortlich sein unsere Plugins zu laden. Wir rufen sie im *FormCreate* unsere Hauptform aus.
Unloadplugins: Diese Prozedur gibt die DLLs und alle verwendeten Objekte wieder frei.
FLoadedForm: Ist ein Array von Forms die wir aus dem DLLs laden.
FLibHandle: Ein Array der die Handles unserer DLLs beinhaltet.

PluginCounter: Zählt die Anzahl der DLLs die wir laden. Wird für die Freigabe am Ende benötigt.
MyFindFile: Das TFindFile-Objekt, das uns beim Durchsuchen des PluginOrdners hilft.

3.5 Die LoadPlugins-Prozedur

```
procedure loadplugins;
  type TMyFormClass = function: TFormClass; stdcall;
  var lMyFormClass: TMyFormClass;
  Plugins    : TStringList;
  i          : Integer;
  newtabsheet : TTabSheet;
begin
  myFindFile := TFindFile.Create(nil);
  myFindFile.FileMask := '*.dll';
  myFindFile.Path := ExtractFilePath(Application.ExeName)+'Plugins\';
  Plugins := TStringList.create;
  Plugins := myFindFile.SearchForFiles;
  setlength(FLibHandle,0);
  setlength(FLoadedForm,0);
  PluginCount := -1;
  for i := 0 to plugins.Count-1 do
  begin
    setlength(FLibHandle,length(FLibHandle)+1);
    setlength(FLoadedForm,length(FLoadedForm)+1);
    inc(PluginCount);
    FLibHandle[length(FLibHandle)-1] := LoadLibrary(PChar(Plugins.strings[i]));
   if not Assigned(FLoadedForm[length(FLoadedForm)-1]) then
   begin
      lMyFormClass := GetProcAddress(FLibHandle[length(FLibHandle)-1], 'MyFormClass');
      if @lMyFormClass <> nil then
      begin
       FLoadedForm[length(FLoadedForm)-1] := lMyFormClass.Create(nil);
       newtabsheet := TTabSheet.Create(nil);
       newtabsheet.PageControl := Form1.PageControl1;
       newtabsheet.Parent := Form1.PageControl1;
       FLoadedForm[length(FLoadedForm)-1].Controls[0].Parent := newtabsheet;
      newtabsheet.Caption := ExtractFileName(copy(Plugins.Strings[i],1,length(Plugins.Strings[i])-4));
     end
     else showmessage('Plugin "'+Plugins.Strings[i]+'" could not be loaded!');
   end;
  end;
end;
```

Ganz grob was hier passiert:

1. Wir erstellen ein Objekt des Typs TFindFile um das Unterverzeichnis „Plugins“ nach allen Dateien der Endung „dll“ zu durchsuchen und die Dateien in der StringList *Plugins* zu speichern.
2. Jede einzelnen Bibliothek laden wir mittels **LoadLibrary** und schauen darin nach ob ein Objekt des Typs **MyFormClass** existiert. Falls ja, laden wir die Form, erstellen ein neues Tabsheet für unser PageControl und ziehen von der Form das Control[0], also unser Panel auf das Tabsheet.
3. Die Caption des Tabsheets ermitteln wir aus dem Namen der DLL (ohne die Endung „.dll“)
4. Sollte eine DLL keine **MyFormClass** enthalten, so wird ein Fehler ausgegeben.

3.6 Die UnloadPlugins-Prozedur

```
procedure unloadplugins;
  var i:integer;
begin
  for i := 0 to PluginCount do
  begin
    FLoadedForm[i].Free;
    FreeLibrary(FLibHandle[i]);
  end;
end;
```

Hier werden alle Bibliotheken und alle Forms aus den DLLs wieder freigegeben. Die Prozedur wird am besten im **Destroy** der Hauptform aufgerufen.

3.7 Auch hier wieder der Package-Kniff

Auch für unsere Parent-Applikation müssen wir den Haken bei *Project → Options… → Packages → Build with runtime packages* setzen und alle Einträge bis auf **vcl** entfernen.
Wenn wir nun unsere compilierte DLL in das Unterverzeichnis „Plugins“ unserer Anwendung legen, dann wird sie automatisch zu Start hin geladen und es wird dem *PageControl* eine weitere Seite mit dem Titel des Namens der DLL hinzugefügt.

4 Probleme und worauf wir achten müssen

Wichtig ist, dass wir unserer Vateranwendung alle Units manuell hinzufügen die benötigt werden um die Komponenten zu bedienen die wir aus der DLL kopieren. Im Regelfall genügen **ComCtrls, StdCtrls** und **ExtCtrls**.

Ein dickes Problem was bei einigen meiner Anwendungen auftrat ist, dass beispielsweise Fonts nicht richtig dargestellt werden, wenn man **Schritt 3.7** ausführt. Solange ich keine Lösung für dieses Problem gefunden habe liefere ich dieses Tutorial nicht in Version 1.0 aus. Solltet ihr eine Lösung haben, bitte nehmt mit mir Kontakt über www.jofre.de auf.

5 Appendix

[1] http://leonardorame.blogspot.com/2008/10/delphi-plugin-by-example.html

[2] http://delphi.about.com/od/vclwriteenhance/a/tfindfile.htm