DLL-uri în DELPHI

Paul-Horațiu Stan

În cadrul acestui articol vom prezenta pe scurt modalitatea prin care pot fi create și utilizate librăriile legate dinamic în limbajul Borland Delphi.

Librăriile legate dinamic (***D**ynamic **L**inked **L**ibraries - **DLL***) sunt module executabile care conțin cod și date care pot fi utilizate de către alte aplicații *Windows*.

Motivul principal pentru care au fost introduse *DLL*-urile este următorul: ocuparea a cât mai puțină memorie în cazul în care rulează mai multe aplicații care folosesc aceeași rutină.

Astăzi acest motiv nu mai pare suficient de important având în vedere evoluția extraordinară a calculatoarelor, dar trebuie să ne gândim la primele versiuni ale *Windows-ului* care rulau pe calculatoare care aveau la dispoziție doar 256 KB de memorie.

Totuși, era necesar ca în acest spațiu să poată rula mai multe aplicații. Soluția a venit prin introducerea *DLL*-urilor.

Avantajele acestor librării sunt multiple: economisirea spațiului pe disc, posibilitatea de a înnoi anumite aplicații fără a le schimba în întregime etc.

Conceptual un *DLL* este foarte asemănător cu un unit *Deplhi*; există unele mici deosebiri și anume:

- un *unit* poate fi folosit doar de proiecte *Delphi*, datorită faptului că are un format recunoscut doar de aplicațiile *Pascal*; un *DLL* poate fi folosit de către orice aplicații *Windows*, datorită faptului că el conține direct cod executabil.
- un *unit* poate exporta funcții, proceduri și date; un *DLL* poate exporta numai funcții sau proceduri;
- un *unit* trebuie să fie prezent încă din faza de editare de legături a aplicațiilor, în timp ce un *DLL* poate lipsi.
- un *unit* este legat static de aplicație, în timp ce un *DLL* este legat dinamic de aplicație; astfel, dacă două aplicații care rulează în același timp folosesc aceeași funcție dintr-un unit înseamnă că două rutine identice sunt prezente în sistem în același timp; acest lucru nu se întâmplă în cazul *DLL*-urilor.

Crearea DLL-urilor

Mediul *Delphi* ușurează munca depusă de utilizator în crearea ***DLL***-urilor, și anume prin selectarea itemului (***File/New/New/DLL***) este creat un fișier având următoarea structură:

```
library Project1;

{ Important note about DLL
memory management: ShareMem must
be the  first unit in your
library's USES clause AND your
project's (select  Project-View
Source) USES clause if your DLL
exports any procedures or
functions that pass strings as
parameters or function results.
This  applies to all strings
passed to and from your DLL--
even those that  are nested in
records and classes. ShareMem is
the interface unit to  the
BORLNDMM.DLL shared memory
manager, which must be deployed
along  with your DLL. To avoid
using BORLNDMM.DLL, pass string
information  using PChar or
ShortString parameters. }

uses
  SysUtils,
  Classes;

begin
end.
```

Primul cuvânt al fișierului generat este **`library`**; aceasta arată faptul că dorim să definim o bibliotecă de funcții și proceduri.

În comentariul care apare în fișierul generat suntem avertizați că în cazul în care dorim să folosim șiruri de caractere trebuie ca primul unit din clauza **`USES`** să fie `SHAREMEM.pas` care este, de fapt, o interfață la `DELPHIMM.DLL`.

Declararea și crearea funcțiilor și procedurilor într-un *DLL* se realizează la fel ca în cazul unui *unit*. Exportarea lor se realizează cu ajutorul construcției **`EXPORTS`** în care vor fi enumerate toate metodele exportate spre exterior.

Sintaxa unei astfel de construcții este următoarea:

```
exports
  Identificator metodă clauză_de_export
...
;
```

unde *clauză_de_export* poate fi:

- ***index** număr de ordine* (un număr cuprins între `1` și `2.147.483.647`); dacă nici un număr de ordine nu este atribuit i se asignează automat unul.
- ***nume** șir de caractere constant* (un șir de caractere care arată numele cu care poate fi importată metoda); în cazul în care acest șir nu este specificat importul după nume se poate realiza folosind numele real al metodei.

Observații

O funcție sau procedură trebuie mai întâi declarată și apoi exportată.

Este de preferat exportul după index deoarece este mai rapid.

Indexul trebuie să fie o valoare cât mai mică.

Vom prezenta acum un exemplu în care este creat un *DLL* care exportă o funcție numită `max`:

```
library Project1;
uses SysUtils,Classes;

function max(x,y:Integer):
                          Integer;
begin
  if x > y then
    max := x
  else
    max := y;
end;

exports
  max index 1;

begin
end.
```

Datorită faptului că modul de apel în *Pascal* este implicit *registry*, iar în alte medii de programare modul de apel implicit nu este același, *DLL*-ul creat nu va funcționa decât cu *Delphi*.

Pentru ca *DLL*-ul să funcționeze pentru orice aplicație *Windows* va trebui să definim funcția `max` cu directiva **`stdcall`** (în *Windows* pe 16 biți funcțiile erau exportate folosind convenția `PASCAL`, dar în *Win32* este de preferat apelul `STDCALL` deoarece este recunoscut și de către alte limbaje de programare).

```
library Project1;
uses SysUtils,Classes;

function
max(x,y:Integer):Integer;
                          stdcall;
begin
  if x > y then
    max := x
  else
    max := y;
end;

exports
  max index 1;

begin
end.
```

Utilizarea DLL-urilor

Importul metodelor dintr-un *DLL* se poate realiza atât static, cât și dinamic. În cazul importului static trebuie precizat mai întâi modul de import, care poate fi:

- prin ***nume***

```
procedure import_prin_nume; external
                        'testlib.dll';
```

- prin ***nume nou***

```
procedure import_prin_nume_nou;
          external 'testlib.dll' name
                                 nume_real;
```

în momentul definirii procedura a fost numită `nume_real`, iar în momentul în care este folosită de aplicația care importă *DLL*-ul, ea va fi apelată folosindu-se numele `import_prin_nume_nou`.

- prin ***număr de ordine*** (***index***)

```
procedure import_prin_numar;
       external 'testlib.dll' index 5
```

procedura `import_prin_numar` este intrarea cu indexul `5` în *DLL*.

Exemple

Prezentăm în continuare o aplicație care importă libăria legată dinamic `project1.dll`, și constă dintr-o formă, două căsuțe de text și un buton. La apăsarea butonului va apărea un mesaj care va conține maximul dintre cele două numere din cele două căsuțe de text.

```
unit Unit1;
interface
uses Windows, Messages, SysUtils,
   Classes, Graphics, Controls, Forms,
                     Dialogs, StdCtrls;
type
  TForm1 = class(TForm)
    Edit1, Edit2: TEdit;
    Button1: TButton;

    procedure Button1Click(
                       Sender:TObject);
      private
            { Private declarations }
      public
              { Public declarations }
  end;

var Form1: TForm1;

implementation
{$R *.DFM}

function max(x,y:Integer):Integer;
       stdcall;external 'c:\proj.dll',
                                 index 1;

procedure TForm1.Button1Click(Sender:
                                 TObject);
begin
  showmessage(inttostr(
            max(strtoint(edit1.text),
              strtoint(edit2.text))))
end;
end.
```

Încărcarea dinamică a unui *DLL* se realizează în momentul execuției programului, prin folosirea a trei metode:

- `LoadLibrary` este o funcție *API* cu un singur parametru de tip șir de caractere care reprezintă numele (și opțional calea) fișierului `.DLL`; căutarea fișierului `.DLL` se realizează (în această ordine) în:
  - directorul din care este încărcată aplicația;
  - directorul curent;
  - directorul `System` al sistemului de operare *Windows*;
  - directorul în care este instalat sistemul de operare *Windows*;
  - directoarele care apar în variabila sistem `PATH`;
- `GetProcAdres` este o funcție care returnează adresa unei funcții exportate de către un *DLL* sub forma unui pointer fără tip; ea are doi parametri: un *handle* la *DLL* (obținut cu ajutorul funcției `LoadLibrary`) și numele metodei exportate.
- `FreeLibrary` este o funcție *API* care are ca parametru un *handle* la *DLL*-ul încărcat și eliberează libăria; se returnează `true` în caz de succes și `false` în caz contrar.

Următoarea aplicație încarcă dinamic funcția `max` din `project1.dll`.

```
unit Unit1;
interface
uses Windows, Messages, SysUtils,
   Classes, Graphics, Controls, Forms,
                     Dialogs, StdCtrls;
type
  TForm1 = class(TForm)
    Edit1, Edit2: TEdit;
    Button1: TButton;

    procedure Button1Click(
                       Sender:TObject);
      private
            { Private declarations }
      public
              { Public declarations }
  end;
  Tmax = function(x,y:Integer):
                                 Integer;

var Form1: TForm1; Hand: THandle;
    Max: Tmax;

implementation
{$R *.DFM}

procedure TForm1.Button1Click(Sender:
                                 TObject);
begin
  hand := LoadLibdary('c:\proj.dll');
  if hand <> 0 then
  begin
    @max := getprocadress(hand,'max');
    if max <> nil then
    begin
      showmessage(inttostr(
            max(strtoint(edit1.text),
              strtoint(edit2.text))))
    end;
    freelibrary(hand);
  end;
end;
end.
```