Biblioteki DLL

W tym artykule mam zamiar omówić wszystkie najważniejsze pojęcia dotyczące bibliotek DLL - sposobu ich pisania, wykorzystanie, łączenie z programem itp. Zacznijmy więc.

Biblioteka DLL

DLL to skrót do Dynamic Link Library i jest to plik, w którym znajduje się skompilowany kod źródłowy, który to może być później wykorzystany w połączeniu z aplikacją.

Dobrze, już wiesz, że biblioteka dll może zawierać procedury, które możesz wykorzystać w swoim programie. Teraz pewnie zapytasz: "Po co właściwie wykorzystywać biblioteki, skoro kod można zawrzeć w pliku wykonywalnym EXE?". Dobre pytanie i kilka różnych odpowiedzi. DLL może być wykorzystany przez wszystkie języki programowania. Tak więc pisząc bibliotekę w C++ Builder możesz ją wykorzystać w Delphi. Jest to duża zaleta. Przykładem może być tutaj biblioteka do odtwarzania plików mp3 - jest ona napisana w C++, a możesz ją wykorzystać w Delphi.
Innym przykładem może być funkcja DrawText lub ShellExecute, z których pewnie często korzystasz. Tak naprawdę są to funkcje importowane z bibliotek DLL dostarczonych wraz z Windowsem. Tak więc funkcje DrawText i ShellExecute są funkcjami Windowsowymi.

Inny przykład. Każda aplikacja posiada błędy i nie można się ustrzec. Możesz w swoim programie część kodu umieścić w bibliotece DLL, a część w pliku EXE. Teraz jeżeli wykryjesz jakiś błąd to wystarczy, że wymienisz tylko plik DLL, a nie cały program.

Dobra przejdźmy do konkretów. Żeby stworzyć bibliotekę DLL należy z menu File wybrać New, a następnie w oknie, które się pojawi kliknąć na ikonę z napisem DLL. Otworzy się nowy projekt, a w nim:

```
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
```

```
  View-Project 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 DELPHIMM.DLL shared memory manager, which must be deployed along
  with your DLL. To avoid using DELPHIMM.DLL, pass string information
  using PChar or ShortString parameters. }

uses
  SysUtils,
  Classes;

begin
end.
```

Obszerny komentarz omówię później. Doprowadź wygląd biblioteki do takiej postaci:

```
library Project1;

uses Windows; //<-- wystarczy tylko ten moduł

procedure ShowWindow; stdcall;
begin
  MessageBox(0, 'Witaj! Jestem procedurą z biblioteki DLL!',
  'Witam!', MB_OK);
end;

exports  // eksportuj procedurę
  ShowWindow name 'ShowWindow';

begin
end.
```

Jak widzisz w bibliotekach nie kładzie się komponentów, ale jedynie pisze. Dobra, mamy jedną procedurę. Zauważ klauzurę exports u dołu biblioteki. Otóż po tym słowie wpisuje się nazwy procedur, które zostaną eksportowane "na zewnątrz" tzn., będzie możliwe ich

wykorzystanie poza biblioteką.

Eksportowanie procedur i funkcji

Tak jak mówiłem - po słowie exports wypisuje się procedury i funkcje, które będą eksportowane. Jeżeli masz więcej niż jedną procedurę do eksportu wypisujesz je po przecinku:

```
exports
  Procedura,
  JakasFunkcja;
```

Istnieje możliwość eksportowania procedur poprzez nazwę. Znaczy to, że procedura eksportowana może mieć inną nazwę - np:

```
exports
  JakasEksportowanaProcedura name 'Go';
```

W tym wypadku eksportowaliśmy procedurę JakasEksportowanaProcedura, ale eksportowaliśmy ją pod nazwą Go.

Istnieje także możliwość eksportowania procedur poprzez indeks:

```
exports
  JakasEksportowanaProcedura index 1,
  JakasEksportowanaProcedura2 index 2;
```

Importowanie procedur z bibliotek

Jeżeli już skompilujesz bibliotekę możesz ją wykorzystać w swoim programie. Oto sposób (statyczny) na zaimportowanie procedury z biblioteki. Na samym początku chciałem

wspomnieć, że jest to statyczny sposób na załadowanie biblioteki. Najlepiej w sekcji Interface umieść taki nagłówek:

```
procedure ShowWindow; stdcall external 'Example_lib.dll' name 'ShowWindow';
```

Załadowanie biblioteki następuje za pomocą słowa external - po tym nazwa biblioteki, a na samym końcu nazwa procedury do zaimportowania.

Zauważ słowo stdcall pojawiające się teraz jak i w bibliotece DLL przy nazwie procedury. Zawsze stosuje tę dyrektywę gdyż zapewnia ona kompatybilność jeżeli np. biblioteka jest napisana w Delphi, a wykorzystujesz ją w C++. Istnieją także inne dyrektywy, które możesz wykorzystać przy swoich procedurach:

safecall - Opatrując tą dyrektywą swoją procedurę masz pewność, że każdy zaistniały wyjątek zostanie przekazany do programu wykorzystującego daną bibliotekę.
register - Ta dyrektywa począwszy od Delphi 2 jest dyrektywą domyślną gdyż zapewnia największą efektywność działania. Związane jest to z czyszczeniem stosu podczas działania procedury, ale nie zamierzam się tutaj rozpisywać o tej dyrektywie.
pascal, cdecl Są to dwie konwencje - Pascalowa i C++. Rzadką są one wykorzystane gdyż mieszanką tych dwóch dyrektyw jest stdcall.

Teraz gdy chcesz uruchomić procedurę ShowWindow wystarczy, że gdzieś w programie napiszesz:

```
ShowWindow;
```

Spowoduje to załadowanie z biblioteki DLL powyższej procedury.

Umieszczanie formularzy w bibliotekach DLL

Gdy otworzysz projekt biblioteki DLL możesz w nim umieścić formularz. Wystarczy z menu File wybrać New Form. To nie wszystko musisz bowiem napisać procedurę, która tworzyć i wyświetlać będzie formularz. Oto cały kod biblioteki DLL:

```
library LibSample;

uses
  Forms,
  Main in 'Main.pas' {Form1};

procedure ShowForm;
var
  Form1 : TForm1;
begin
  Form1 := TForm1.Create(Application); // stworz formularz
  Form1.ShowModal; // wyswietl formularz
  Form1.Free; // zwolnij zmienna
end;

exports
  ShowForm index 1;

begin
end.
```

Teraz gdy skompilujesz taką bibliotekę to będzie miała rozmiar zwykłego programu - w Delphi 2 jest to 175 KB. Cóż jest to niewątpliwie wada Delphi - generuje zbyt duże pliki wykonywalne oraz biblioteki DLL.

Ok, teraz wyświetlenie formularza z biblioteki następuje za pomocą takiego kodu:

```
procedure ShowForm; stdcall external 'LibSample.dll' index 1; // to w sekcji Interface
```

No i gdzieś w programie wystarczy, że napiszesz:

```
ShowForm;
```

Ładowanie dynamiczne

Dotychczas podczas ładowania procedur z biblioteki dll posługiwaliśmy się metodą statyczną. Znaczy to, że już podczas uruchamiania programu biblioteka zostaje ładowana, a procedura ładowania i uruchamiana zostaje dopiero później - np. podczas naciśnięcia przycisku.

Ładowanie statyczne polega na wykorzystaniu biblioteki tylko wtedy gdy jest ona potrzebna - tzn., załadowanie biblioteki, procedury, a następnie zwolnienie pamięci.

Ładowanie dynamiczne jest trochę trudniejsze - oto kod:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  DLL : THandle; // uchwyt biblioteki
  ShowForm : procedure;
begin
  DLL := LoadLibrary('LibSample.dll'); // laduj biblioteke
  try
    @ShowForm := GetProcAddress(DLL, 'ShowForm'); // laduj procedure
    if @ShowForm=nil then raise Exception.Create('Błąd - nie mogę znaleźć proceudry w bibliotece!');
    ShowForm; // wywolaj procedure
  finally
    FreeLibrary(DLL); // wreszcie zwolnij pamiec
  end;
end;
```

Ładowanie biblioteki następuje za pomocą polecenia LoadLibrary. To nie wszystko bowiem dla zmiennej DLL przypisaliśmy bibliotekę. Zauważ, że umieściłem zmienną ShowForm, która nie jest żadnego konkretnego typu, ale oznacza to, że zmienna będzie procedurą. Do tej zmiennej przypisany zostaje adres proceudry znajdującej się w bibliotece (GetProcAddress). Jeżeli wartość ta równa się nil to znaczy, że proceudry lub funkcji o tej nazwie nie można odnaleźć w bibliotece. Na końcu biblioteka jest zwalniana z pamięci.

Procedura inicjująco - kończąca

Skąd tak dziwna nazwa? Biblioteki DLL nie posiadają, w przeciwieństwie do modułów, sekcji intialization oraz finalization. Co trzeba zrobić jeżeli chce się wykonać jakieś operacje po tym jak biblioteka np. zostanie zwolniona z pamięci? Z pomocą przychodzi DLLProc. Po kolei. Możesz stworzyć taką procedurę:

```
library Project1;

uses
  Windows;

procedure Inicjajcja(Reason : Integer);
begin
  if Reason=DLL_PROCESS_DETACH then
 { biblioteka jest usuwana z pamieci. Tutaj zamieszczone sa
   odpowiednie operacje. }
end;

begin
  DLLProc := @Inicjajcja; // przypisanie procedury
end.
```

W bloku begin..end dla DLLProc (procedury inicjująco - kończącej) zostanie przypisana procedura, którą wcześniej stworzyliśmy. Tak jak pokazane to zostało w przykładzie - możemy kontrolować moment, w którym biblioteka zostaje usuwana z pamięci. Zmienna Reason może przybrać taką wartość:

DLL_PROCES_DETACH - Biblioteka jest usuwana z pamieci.
DLL_THREAD_ATTACH - Tworzenie nowego wątku.
DLL_THREAD_DETACH - Niszczenie (koniec) wątku.

Oto jeszcze jeden przykład procedury inicjująco - kończącej:

```
library Project1;

uses
  Windows, SysUtils;
```

```
var
  P : Pointer;

procedure Inicjajcja(Reason : Integer);
begin
  if Reason=DLL_PROCESS_DETACH then
 { biblioteka jest usuwana z pamieci. Tutaj zamieszczone sa
   odpowiednie operacje. }
    FreeMem(P); // zwalnianie pamieci
end;

begin
  DLLProc := @Inicjajcja; // przypisanie procedury
  { dalsze operacje }
  P := AllocMem(1024); // rezerwacja kilobajtu w pamieci
end.
```

W tym przykładzie na początku działania biblioteki (gdy DLL jest ładowany do pamięci) rezerwowana zostaje pamięć w komputerze. Na końcu działania biblioteki pamięci ta jest zwalniana.

Parametry funkcji

Jak w zwykłych procedurach również z bibliotek DLL to programu możesz eksportować procedury. Niektórzy mają z tym problemy. W naszym przykładzie spróbujemy eksportować z biblioteki DLL cały rekord. Rekord, który będziemy eksportować umieść w pliku - np. records.inc:

```
{ plik ten zawiera deklaracje rekordu, z ktorego bedziemy korzystac }

type
  PSomeRec=^TSomeRec;
  TSomeRec=record
    Name: PChar; // imie i nazwisko
```

```
  City : PChar; // miasto
  Country : PChar; // kraj
  Code : Integer; // kod
 end;
```

Jak zauważyłeś eksportować będziemy wskaźnik na rekord. Jak to wygląda w całości - tzn., jak wyeksportować rekord używając procedury w bibliotece:

```
procedure CreateRecord(var SomeRec : PSomeRec); stdcall;
begin
{ przydzielenie pol do rekordu }
 with SomeRec^ do
 begin
  Name := 'Adam Boduch';
  City := 'Wrocław';
  Country := 'Polska';
  Code := 54150;
 end;
end;
```

W procedurze tej po prostu używając wskaźnika eksportujemy rekord wypełniając jego pola. Chyba nic nie wymaga tutaj komentarza. A jak zaimportować tę procedurę w programie? Oto właściwy kod:

```
{ oto procedura importowana z DLLa w sposob statyczny }
procedure CreateRecord(var SomeRec : PSomeRec); stdcall
   external 'records.dll' name 'CreateRecord';

var
 Rec : PSomeRec; // wskazanie na rekord
begin
 New(Rec); // przydzielenie pamieci
{ przydzielenie rekoru z procedury importiwanej z DLL'a do rekordu Rec }
 CreateRecord(Rec);
 MessageBox(0, PChar(  // wyswietlenie elementow rekordu
```

```
      'Oto przekazane przez bibliotekę dane: ' + #13#13 +
      Rec^.Name + #13 +
      Rec^.City + #13 +
      Rec^.Country + #13),
  'Oto dane przekazane...', MB_OK + MB_ICONINFORMATION);
  Dispose(Rec); // zwolnienie pamieci
end.
```

Podczas importu musimy najpierw przydzielić pamięć dla importowanego rekordu. Później wywołujemy procedurę, która powoduje przypisanie pól z rekordu do rekordu Rec. Później oczywiście prezentujemy poszczególne elementy rekordu. No i na końcu NALEŻY zwolnić pamięć potrzebna do wykonania tego rekordu poleceniem Dispose. Cały kod tego programu możesz ściągnąć stąd.

Komentarz biblioteki

Na początku tego artykułu mówiłem o komentarzu na początku biblioteki. O co właściwie chodzi? Mówi o tym, że jeżeli korzystasz w bibliotece z długich łańcuchów to musisz do listy modułów (na pierwszym miejscu na liście modułów) dopisać słowo ShareMem, a oprócz tego do programu dołączyć bibliotekę Borlndmm.dll (**nie** Delphimm.dll!). Żeby tego uniknąć nie stosuj długich łańcuchów:

Zamiast pisać:

```
procedure Koduj(var S : String);
begin
  { jakis kod }
end;
```

Zamieniaj String na PChar:

```
procedure Koduj(var S : PChar);
begin
```

```
  { jakis kod }
end;
```
