DLL (*динамические библиотеки*) содержат дополнительные функции или сложные библиотеки для преобразования изображений.

Также в **DLL** можно хранить не только функции и процедуры, но также и ресурсы (всевозможные изображения, иконки, строки и т.д.). А также формы, плагины.

Краткое описание функций и приёмов работы с Dll

Разберём два способа работы с Dll:

1 способ
Привязка Dll к программе

Это наиболее простой метод работы с **Dll**, но он имеет один существенный недостаток.

Если библиотека не будет найдена, то программа не запустится, просто выдавая ошибку о том, что не может найти **Dll**. Поиск ведётся, в корневом каталоге, в текущем, в папке C:WindowsSystem, по указанному адресу и т.д.

Общая форма приёма:

```
implementation ... function FunctionName(Par1: Par1Type; Par2: Par2Type; ...): ReturnType; stdcall; external 'DLLNAME.DLL' name 'FunctionName' index FuncIndex;
```

А если это не функция, а процедура?

```
procedure ProcedureName(Par1: Par1Type; Par2: Par2Type; ...); stdcall; external 'DLLNAME.DLL' name 'ProcedureName' index ProcIndex;
```

FunctionName(*ProcedureName*) - имя функции или процедуры, которая будет использоваться в программе.
Par1, Par2 ... - список параметров, которые будут использоваться в функции(процедуре).
ReturnType - возвращаемое значение, только для функции.
stdcall - директива, которая должна точно совпадать с используемой в самой **DLL.**

external - директива, указывающая на **Dll** из которой будет импортирована функция или процедура(в данном случае DllName.Dll.

Name - директива, указывающая на имя конкретной функции, которая должна быть импортирована из нужной **Dll**. Эта директива является необязательной.

index - Это ещё одна необязательная директива, которая указывает порядковый номер функции(процедуры) в **Dll.**

2 способ

Динамическая загрузка Dll

Несмотря на то, что этот метод более сложный, он является, наиболее элегантным. Хорошо то, что он лишён недостатка первого способа, но объём кода необходимый для этого варианта поражает размерами, , причем сложность в том, что функция, импортируемая из **DLL** достуна лишь тогда, когда эта **DLL** загружена и находится в памяти. Далее следует, краткое описание функций API используемых этим методом.

LoadLibrary(LibFileName: PChar) - Загрузка указанной **Dll** в память. При успешном завершении операции, возвращается дескриптор
THandle
этой библиотеки.

GetProcAddress(Module: THandle; ProcName: PChar) - считывает адрес экспортированной библиотечной функции. При успешном завершении функция возвращает дескриптор **TFarProc** функции в загруженной
DLL
.

FreeLibrary(LibModule: THandle) - делает недействительным этот модуль и освобождает связанную с ним память. После вызова этой функции, функции библиотеки недоступны.

А теперь самое главное, ПРИМЕРЫ...

Пример 1

Привязка Dll к программе

```
{ Здесь идет заголовок файла и определение формы TForm1 и ее экземпляра Form1 } implementation { Определяем внешнюю библиотечную функцию } function GetSimpleText(LangRus: Boolean): PChar; stdcall; external 'MYDLL.DLL'; procedure Button1Click(Sender: TObject); begin {И используем ее} ShowMessage(StrPas(GetSimpleText(True))); ShowMessage(StrPas(GetSimpleText(False))); { ShowMessage - показывает диалоговое окно с указанной надписью; StrPas - преобразует строку PChar в string } end;
```

Теперь то-же самое, но только вторым способом(Помните я вам говорил про массивность кода?)...

Пример 2

Динамическая загрузка Dll

```
{... Здесь идет заголовок файла и определение формы TForm1 и ее экземпляра Form1 } var Form1: TForm1; GetSimpleText: function(LangRus: Boolean): PChar; LibHandle: THandle; procedure Button1Click(Sender: TObject); begin { "Чистим" адрес функции от "грязи" } @GetSimpleText := nil; { Пытаемся загрузить библиотеку } LibHandle := LoadLibrary('MYDLL.DLL'); { Если все OK } if LibHandle >= 32 then begin { ...то пытаемся получить адрес функции в библиотеке } @GetSimpleText := GetProcAddress(LibHandle, 'GetSimpleText'); { Если и здесь все OK } if @GetSimpleText  nil then { ...то вызываем эту функцию и показываем результат } ShowMessage(StrPas(GetSimpleText(True))); end; { И не забываем освободить память и выгрузить DLL } FreeLibrary(LibHandle); end;
```

Отличия первого метода от второго видны я думаю сразу... Ну да ладно, посмотрим теперь на саму библиотеку.

Пример 3

Исходник проекта MyDll.dpr.

```
library mydll; uses SysUtils, Classes; { Определяем функцию как stdcall } function GetSimpleText(LangRus: Boolean): PChar; stdcall; begin { В зависимости от LangRus возвращаем русскую (True) либо английскую (False) фразу } if LangRus then Result := PChar('Здравствуй, мир!') else Result := PChar('Hello, world!'); end; { Директива exports указывает, какие функции будут экспортированы этой DLL } exports GetSimpleText; begin end.
```

Как уже писалось выше, в **Dll** можно размещать не только функции, но и иконки, курсоры, рисунки, меню и т.д. Приведённое ниже описание покажет вам, как это сделать. Для начала опишу всё поподробнее. Для помещения ресурсов в **Dll** нужно:

- ***.RES** файл с необходимыми ресурсами.
- Исходник **Dll**, куда добавляем.
- Скомпилировать **Dll**, включая ***.RES**.

Теперь, тоже, но поподробнее:

1. Рекумендую создать какой-нибудь каталог в папке %Delphi%Bin, например MyDllTest.
2. ***.RES** файл с необходимыми ресурсами.

а) Запускаем BorlandImageEditor. File -> New -> Resource File (*.res). После этого Resource -> New -> [тип ресурса]. Сохраняем данный файл в каталоге MyDllTest. Например *MyRes.res* .

б) Компилируем **.res* файл при помощи brcc32.exe из %Delphi%Bin: brcc32.exe mydll.txt, предварительно записав в *myres.txt* приблизительно такую информацию:

```
{ ---[ Myres.txt ]--- }  FlyFold AVI "C:WindowsFlyFolders.avi"  CoolPic BITMAP "C:MyBitmapsmyPhoto.bmp"  MyDta RCDATA "C:myfile.dta"  { ---[ MyRes.txt ]--- }
```

Здесь первое поле - название ресурса в **.res* файле(и в **Dll** соответственно), второе - его тип (**ICON>, BITMAP** и т.д.). Третье - местонахождение файла, содержащего данный ресурс.

Перемещаем полученный **.res* в каталог MyDllTest.

Исходник **Dll**, куда добавляем В каталоге MyDllTest создаём файл следующего содержания (если нет готового исходника):

```
{ ---[ MyDll.pas ]--- }  library mylib;  {$R MyRes.RES}  begin  end  { ---[ MyDll.pas ]--- }
```

Компилируем **Dll**, включая **.res*.

Находясь в каталоге %Delphi%Bin, набрать в командной строке:

dcc32.exe MyDllTestmydll.pas

Открыть **Dll** из Delphi и нажать **build**.

Размещение в **DLL** ресурсов и форм. В **DLL** можно размещать не только функции, но и курсоры, рисунки, иконки, меню, текстовые строки.

На этом мы останавливаться не будем. Замечу лишь, что для загрузки ресурса нужно загрузить **DLL**, а затем, получив ее дескриптор, - загружать сам ресурс соотвествующей функцией (**LoadIcon, LoadCursor**, и т.д.).

В этом разделе мы лишь немного затронем размещение в библиотеках **DLL** окон приложения (т.е. форм в Дельфи). Для этого нужно создать новую
DLL
и добавить в нее новую форму (File -> New -> DLL, а затем - File -> New Form).

Далее, если форма представляет собой диалоговое окно (модальную форму (*bsDialog*)), то добавляем в
DLL
следующую функцию (допустим, форма называется
Form1
, а ее класс -
TForm1
):

Пример 3

Размещение формы в Dll

function ShowMyDialog(Msg: PChar): Boolean; stdcall; exports ShowMyDialog; function ShowMyDialog(Msg: PChar): Boolean; begin { Создаем экземпляр Form1 формы TForm1 } Form1 := TForm1.Create(Application); { В Label1 выводим Msg } Form1.Label1.Caption := StrPas(Msg); { Возвращаем True только если нажата OK (ModalResult = mrOk) } Result := (Form1.ShowModal = mrOk); { Освобождаем память } Form1.Free; end;

Если же нужно разместить в **DLL** немодальную форму, то необходимо сделать две

функции - открытия и закрытия формы. При этом нужно заставить
DLL
запомнить дескриптор этой формы.

Создание плагинов Здесь мы не будем подробно рассматривать плагины, т.к. уже приведенные выше примеры помогут Вам легко разобраться в львиной части программирования **DLL**.

Напомню лишь, что плагин - дополнение к программе, расширяющее ее возможности. При этом сама программа обязательно должна предусматривать наличие таких дополнений и позволять им выполнять свое предназначение.

То есть, например, чтобы создать плагин к графическому редактору, который бы выполнял преобразование изображений, Вам нужно предусмотреть как минимум две функции в плагине (и, соответственно, вызвать эти функции в программе) - функция, которая бы возвращала имя плагина (и/или его тип), чтобы добавить этот плагин в меню (или в тулбар), плюс главная функция - передачи и приема изображения.

То есть сначала программа ищет плагины, потом для каждого найденного вызывает его опозновательную функцию со строго определенным именем (например, **GetPluginName**) и добавляет нужный пункт в меню, затем, если пользователь выбрал этот пункт - вызывает вторую функцию, которой передает входное изображение (либо имя файла, содержащего это изображение), а эта функция, в свою очередь, обрабатывает изображение и возвращает его в новом виде (или имя файла с новым изображением).

Вот и вся сущность плагина.

code