Использование DLL в Delphi

Содержание

Понятие DLL

Вспомним процесс программирования в DOS. Преобразование исходного текста программы в машинный код включал в себя два процесса - компиляцию и линковку. В процессе линковки, редактор связей, компоновавший отдельные модули программы, помещал в код программы не только объявления функций и процедур, но и их полный код. Вы готовили таким образом одну программу, другую, третью ... И везде код одних и тех же функций помещался в программу полностью (см. рис 1).

Программа1 Программа2 : : MyFunc(:) MyFunc(:) : : код функции MyFunc код функции MyFunc код других функций код других функций

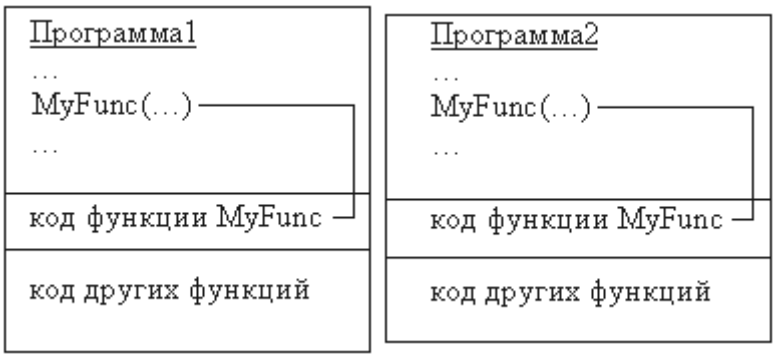

Рис.1 : Вызов функций при использовании статической компоновки

В многозадачной среде такой подход был бы по меньшей мере безрассудным, так как очевидно, что огромное количество одних и тех же функций, отвечающих за прорисовку элементов пользовательского интерфейса, за доступ к системным ресурсам и т.п. дублировались бы полностью во всех приложениях, что привело бы к быстрому истощению самого дорогого ресурса - оперативной памяти. В качестве решения возникшей проблемы, еще на UNIX-подобных платформах была предложена концепция динамической компоновки (см. рис . 2).

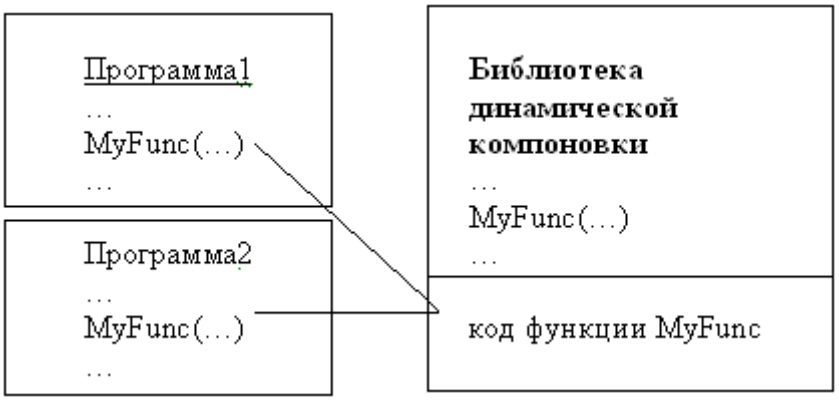

Рис.2: Вызов функций при использовании динамической компоновки

Но, чем же отличаются Dynamic Link Library (DLL) от обычных приложений? Для понимания этого требуется уточнить понятия задачи (task), экземпляра (копии) приложения (instance) и модуля (module).

При запуске нескольких экземпляров одного приложения, Windows загружает в оперативную память только одну копию кода и ресурсов - модуль приложения, создавая несколько отдельных сегментов данных, стека и очереди сообщений (см. рис. 3), каждый набор которых представляет из себя задачу, в понимании Windows. Копия приложения представляет из себя контекст, в котором выполняется модуль приложения.

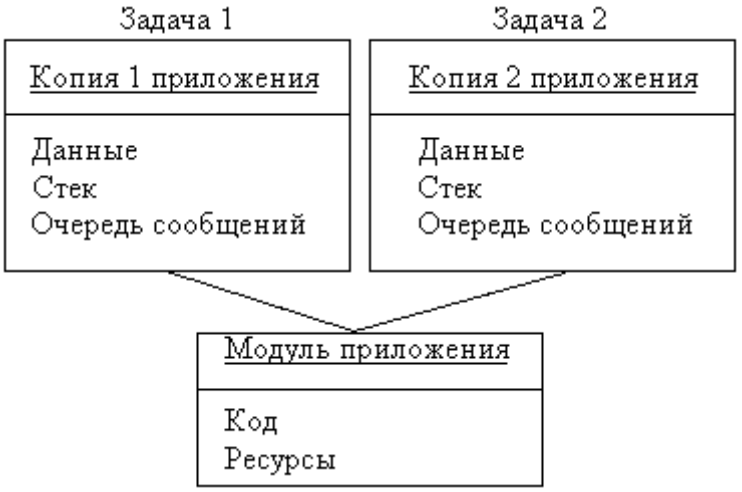

Рис.3 : Копии приложения и модуль приложения

DLL - библиотека также является модулем. Она находится в памяти в единственном экземпляре и содержит сегмент кода и ресурсы, а также сегмент данных (см. рис. 4).

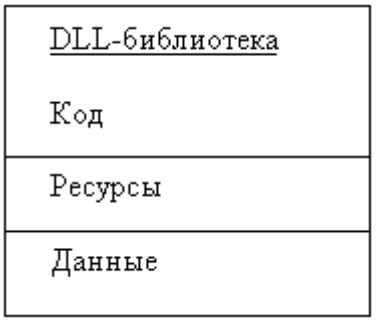

Рис.4 : Структура DLL в памяти

DLL - библиотека, в отличие от приложения не имеет ни стека, ни очереди сообщений. Функции, помещенные в DLL, выполняются в контексте вызвавшего приложения, пользуясь его стеком. Но эти же функции используют сегмент данных, принадлежащий библиотеке, а не копии приложения.

В силу такой организации DLL, экономия памяти достигается за счет того, что все запущенные приложения используют один модуль DLL, не включая те или иные стандартные функции в состав своих модулей.

Часто, в виде DLL создаются отдельные наборы функций, объединенные по тем или иным логическим признакам, аналогично тому, как концептуально происходит планирование модулей (в смысле unit) в Pascal. Отличие заключается в том, что функции из модулей Pascal компонуются статически - на этапе линковки, а функции из DLL компонуются динамически, то есть в run-time.

Создание DLL в Delphi (экспорт)

Для программирования DLL Delphi предоставляет ряд ключевых слов и правил синтаксиса. Главное - DLL в Delphi такой же проект как и программа.

Рассмотрим шаблон DLL:

```
library MyDll;
uses
   <используемые модули>;
 <объявления и описания функций>
exports
 < экспортируемые функции>
begin
   <инициализационная часть>
end.
```

Имя файла проекта для такого шаблона должно быть MYDLL.DPR.

!!!! К сожалению, в IDE Delphi автоматически генерируется только проект программы, поэтому Вам придется проект DLL готовить вручную. В Delphi 2.0 это неудобство устранено.

Как и в программе, в DLL присутствует раздел uses. Инициализационная часть необязательна. В разделе же exports перечисляются функции, доступ к которым должен производится из внешних приложений.

Экспортирование функций (и процедур) может производится несколькими способами:

- по номеру (индексу);
- по имени.

В зависимости от этого используется различный синтаксис:

```
{экспорт по индексу}
procedure ExportByOrdinal; export;
begin
  .....
end;
exports
  ExportByOrdinal index 10;
{экспорт по имени}
procedure ExportByName; export;
begin
  .....
end;
exports
  ExportByName name 'MYEXPORTPROC'; { имя для экспорта может не совпадать с име
```

Так как в Windows существует понятие "резидентных функций" DLL, то есть тех функций, которые находятся в памяти на протяжении всего времени существования DLL в памяти, в Delphi имеются средства для организации и такого рода экспорта:

```
exports
  ExportByName name 'MYEXPORTPROC' resident;
```

Стоит отметить тот факт, что поиск функций, экспортируемых по индексу, производится быстрее, чем при экспорте по имени. С другой стороны, экспорт по имени удобнее, особенно если Вы периодически дополняете и расширяете набор экспортируемых из DLL функций, при гарантии работы приложений, использующих DLL, и не хотите специально следить за соблюдением уникальности и соответствия индексов.

Если же Вы будете экспортировать функции следующим образом:

```
exports
  MyExportFunc1,
```

```
  MyExportFunc2,
  .....;
```

то индексирование экспортируемых функций будет произведено Delphi автоматически, а такой экспорт будет считаться экспортом по имени, совпадающему с именем функции. Тогда объявление импортируемой функции в приложении должно совпадать по имени с объявлением функции в DLL. Что же касается директив, накладываемых уже на импортируемые функции, то об этом мы поговорим ниже.

Использование DLL в Delphi (импорт)

Для организации импорта, т.е. доступа к функциям, экспортируемым из DLL, так же как и для их экспорта, Delphi предоставляет стандартные средства.

Для показанных выше примеров, в Вашей программе следует объявить функции, импортируемые из DLL таким образом:

```
{ импорт по специфицированному имени  }
procedure ImportByName;external 'MYDLL' name 'MYEXPORTPROC';
{ импорт по индексу  }
procedure ImportByOrdinal;  external  'MYDLL'  index  10;
{ импорт по оригинальному имени  }
procedure MyExportFunc1;  external  'MYDLL';
```

Этот способ называется статическим импортом.

Как Вы могли заметить, расширение файла, содержащего DLL, не указывается - по умолчанию подразумеваются файлы *.DLL и *.EXE. Как же тогда быть в случае, если файл имеет другое расширение (например, как COMPLIB.DCL в Delphi), или если требуется динамическое определение DLL и импортируемых функций (например, Ваша программа работает с различными графическими форматами, и для каждого из них существует отдельная DLL.)?

Для решения такого рода проблем Вы можете обратиться напрямую к API Windows, используя, так называемый, динамический импорт:

```
uses
  WinTypes,  WinProcs, ... ;
type
  TMyProc  = procedure ;
var
  Handle  :  THandle;
  MyImportProc  :  TMyProc;
begin
  Handle:=LoadLibrary('MYDLL');
  if Handle>=32 then        { if <=32 - error ! }
  begin
    @MyImportProc:=GetProcAddress(Handle,'MYEXPORTPROC');
    if MyImportProc<>nil then
      ..... {using imported procedure}
  end;
  FreeLibrary(Handle);
end;
```

!!! Синтаксические диаграммы объявлений экспорта/импорта, подмена точки выхода из DLL, и другие примеры, Вы можете найти в OnLine Help Delphi, Object Pascal Language Guide, входящему в Borland RAD Pack for Delphi, и, например, в книге "Teach Yourself Delphi in 21 Days".

Если не говорить о генерируемом компилятором коде (сейчас он более оптимизирован), то все правила синтаксиса остались те же , что и в Borland Pascal 7.0

DLL, использующие объекты VCL для работы с данными

При создании своей динамической библиотеки Вы можете использовать вызовы функций из других DLL. Пример такой DLL есть в поставке Delphi (X:\DELPHI\DEMOS\BD\BDEDLL). В эту DLL помещена форма, отображающая данные из таблицы и использующая для доступа к ней объекты VCL (TTable, TDBGrid, TSession), которые, в свою очередь, вызывают функции BDE. Как следует из комментариев к этому примеру, для такой DLL имеется ограничение: ее не могут одновременно использовать несколько задач. Это вызвано тем, что объект Session, который создается автоматически при подключении модуля DB, инициализируется для модуля, а не для задачи. Если попытаться загрузить эту DLL вторично из другого приложения, то возникнет ошибка. Для предотвращения одновременной загрузки DLL несколькими задачами нужно осуществить некоторые действия. В примере - это процедура проверки того, используется ли DLL в данный момент другой задачей.

Исключительные ситуации в DLL

Возникновение исключительной ситуации в DLL, созданной в Delphi, приведет к прекращению выполнения всего приложения, если эта ситуация не была обработана внутри DLL. Поэтому желательно предусмотреть все возможные неприятности на момент разработки DLL. Можно порекомендовать возвращать результат выполнения импортируемой функции в виде строки или числа и, при необходимости, заново вызывать исключительную ситуацию в программе.

Код в DLL:

```
function MyFunc : string;
begin
  try
  {собственно код функции}
  except
    on EResult: Exception do
      Result:=Format(DllErrorViewingTable,
 [EResult.Message]);
    else
      Result := Format(DllErrorViewingTable,
 ['Unknown error']);
  end;
end;
```

Код в программе:

```
StrResult:=MyFunc;
if StrResult<>'' then
  raise Exception.Create(StrResult);
```