ПУСТЬ ЭТА КНИГА ПРИНЕСЕТ ВАМ УДАЧУ!

Г.А. Тяпичев

«СОЛОН»

Быстрое программирование на C++

- уникальная пошаговая методика для новичков
- справочная система и инсталляция своими руками
- программирование для Интернета
- подборка примеров на все случаи жизни

Серия
Про ПК

УДК 681.3
ББК 32.97326-018.2
Т19

Г. А. Тяпичев
Т19 Быстрое программирование на C++. — М.: СОЛОН-Пресс, 2009. —384 с.: ил. — (Серия «Про ПК»).

ISBN 5-98003-162-6

Книга предназначена для широкого круга читателей, которые не знакомы с каким-либо языком программирования, но мечтают о создании компьютерных программ собственными силами. Автор предлагает читателю самостоятельно научиться программированию в среде C++ Builder по разработанной им методике «начать от нуля». В процессе работы над книгой читатель научится создавать проекты компьютерных программ и, одновременно, будет осваивать основы языка программирования C/C++, а также требования среды программирования. В книге рассмотрены вопросы создания справочных систем и инсталляции программ, описаны особенности программирования звука и принтера, вопросы программирования для Интернета.

Компакт-диск содержит множество оригинальных авторских примеров и листингов, которые сделают процесс обучения быстрым и интересным.

По вопросам приобретения обращаться:
ООО «Альянс-книга»
Тел: (095) 258-91-94, 258-91-95, **www.abook.ru**

Фирменный магазин издательства «СОЛОН-Пресс»
г. Москва, ул. Бахрушина, д. 28 (м. «Павелецкая кольцевая»)
Тел.: 959-21-03, 959-20-94

ISBN 5-98003-162-6

Глава 3. Более сложное программирование

Разработка и использование DLL

Использование модулей DLL (Dynamic Link Library — динамически связываемая библиотека) — несомненно является одним из самых распространенных в настоящее время способов создания модульных приложений. Например, сама операционная система Windows почти полностью состоит из динамических библиотек. Такие модули, как KERNEL, USER и многие другие, а также файлы шрифтов (.FOT) и разных драйверов (.DRV) являются динамически подключаемыми библиотеками.

Использование DLL-библиотек является нормой почти для всех коммерческих приложений.

DLL представляет собой особый вид выполняемого кода, который загружается в память основным приложением в процессе его выполнения и может содержать такой же код, как и обычный выполняемый файл программы. Программу, оформленную в виде DLL, нельзя запустить на выполнение в одиночку — должна существовать специальная программа, способная эту DLL загрузить и «толкнуть». Этой специальной программой может быть основной выполняемый файл приложения или другой модуль DLL.

Модуль DLL можно загружать статически или динамически. Статически загружаемые модули связываются с главной выполняемой программой при ее компоновке и остаются загруженными в память все время, пока выполняется приложение. Динамически загружаемый модуль в процессе работы приложения при необходимости можно выгрузить из памяти, освободив таким образом ресурсы. Другое существенное отличие между статически и динамически загружаемыми DLL в том, что без первых приложение нельзя запустить на выполнение, а модуль второго типа вообще может отсутствовать на компьютере, если приложение выполняет такие операции, в которых этот модуль «не участвует». Работа приложения прекратится только в том случае, если потребуется обращение к функции, размещенной в отсутствующем модуле DLL.

Но DLL отличается не только способом загрузки, но и содержимым. Распространена практика помещать в DLL таблицы строковых констант — это позволяет легко адаптировать приложение к потребностям пользователей, говорящих на разных языках.

Приступая к разработке DLL-модуля, нужно прежде всего решить, что именно в этом модуле должно быть доступно другим компонентам приложения — будут ли это отдельные функции или какая-то информация. Если предполагается экспортировать (выдавать из DLL) функции, то рекомендуется разработать экспортируемую функцию таким образом, чтобы она не обращалась к функциям из других модулей. Обычно в состав модулей DLL входят функции оболочки, функции работы с сетью, какие-либо специальные классы.

Использование мастера DLL Wizard

Работая в среде C++ Builder, проще всего создать модуль DLL с помощью мастера **DLL Wizard**. Для того чтобы вызвать мастера на экран, выбираем команду **File➔New➔Other**. На экране появляется окно выбора **New Items** (см. рис. 3.1), в котором выбираем **DLL Wizard** и щелкаем на клавише **OK**.

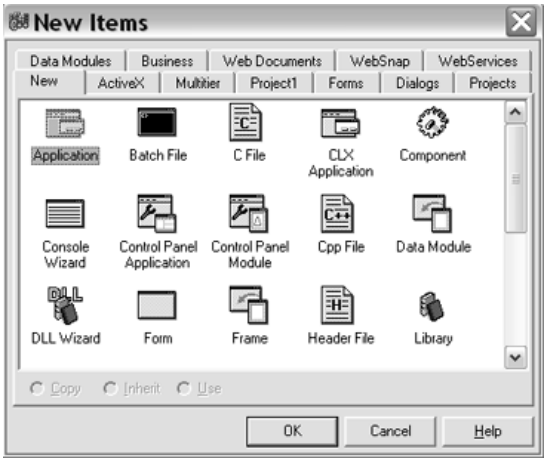

Рис. 3.1. Окно New Items

Окно **DLL Wizard** (см. рис. 3.2) предлагает сделать выбор используемого языка программирования и библиотеки компонентов.

Рис. 3.2. Окно DLL Wizard

Ниже описаны различные варианты выбора, но для нашего простого случая не следует долго задумываться, поэтому оставляйте все так, как оно есть, и нажимайте на клавишу **OK**. Варианты выбора:

- **C**. При выборе этого переключателя флажок **Use VCL** (использовать VCL) блокируется. Оно и понятно — компоненты из библиотеки VCL можно включить только в проект на языке C++. Программе на языке C незнакомы такие

понятия, как класс, объект и т. п., а потому, если в создаваемом модуле вы собираетесь использовать программные компоненты, созданные на языке C++, переключатель C устанавливать не следует.

- **C++**. Выбор этой опции позволяет использовать любые компоненты из библиотеки VCL, организовать многопотоковый режим выполнения и создать DLL в стиле Visul C++. При выборе этой опции для построения выполняемого модуля DLL будет использован компилятор языка C++, а значит, в текст программы можно включать любой программный код на этом языке.
- **Use VCL**. Если этот флажок установлен, то в состав DLL можно включать компоненты из библиотеки VCL. Выше уже отмечалось, что этот флажок блокируется, если выбран переключатель **C**. При установке флажка **Use VCL** C++ Builder включает директиву #include <VCL.h> в главный модуль программного кода DLL и соответственно настраивает код запуска и опции компоновщика. Обращаю ваше внимание на то, что при установке этого флажка соседний флажок — **Multi Threaded** — блокируется и его нельзя сбросить. Причина в том, что компоненты VCL нуждаются в поддержке многопотокового режима выполнения программы.
- **Multi Threaded**. Установка этого флажка задает поддержку многопотокового режима выполнения программы DLL. Рекомендуется, если нет особых возражений, всегда устанавливать этот флажок. Если установлен флажок **Use VCL**, то установку **Multi Threaded** среда C++ Builder выполнит автоматически, причем после этого флажок блокируется и сбросить его нельзя.
- **VC++ Style DLL**. Установка или сброс этого флажка задает тип точки запуска создаваемого модуля DLL. Если необходимо, чтобы при запуске вызывалась функция **DLLMain()**, как то предусмотрено в Visual C++, установите флажок **VC++ Style DLL**. В противном случае будет вызываться функция **DLLEntryPoint()**. Как правило, этот флажок сбрасывается, причем независимо от того, планируется ли в дальнейшем использовать этот модуль DLL совместно с приложением, разработанным в среде Visual C++.

Следует учитывать, что чаще всего при работе с мастером используется настройка, предлагаемая по умолчанию. Но иногда такой вариант по каким-либо причинам не подходит и нужно вспомнить, что влечет за собой выбор той или иной опции. Более подробную информацию об этих опциях можно почерпнуть из системы оперативной справки C++ Builder.

Создаем файл CWcore1.dll

Название создаваемого файла подключаемой библиотеки **CWcore1.dll** состоит из двух составляющих: буквосочетание **CW** обозначает телеграфную радиосвязь, а английское слово *core* можно перевести как «основа», «ядро». Примерно такими обозначениями для подобных файлов пользуются американские разработчики программ. По русски название можно перевести как «основа для телеграфного сигнала». Достаточная доля правды в этом есть, т. к. в файле будут

заложены коды для формирования звукового телеграфного сигнала. Цифра '1' обозначает вариант библиотечного файла.

Создаем файлы проекта

Для начала создадим папку, в которой предстоит хранить и создавать файл **CWcore1.dll**. Предположим, это будет **D:\CWcore1**.

Вспомним наши действия в предыдущем разделе.

- Мы выбрали в главном меню C++ Builder пункты **File➔NEW➔Other**. На экране появляется окно выбора **New Items** (см. рис. 3.1), в котором выбираем **DLL Wizard** и щелкаем на клавише **OK**.
- Окно **DLL Wizard** (см. рис. 3.2) предлагает сделать выбор используемого языка программирования и библиотеки компонентов. Ничего не изменяем и нажимаем клавишу **OK**. Этим самым нами дана команда для C++ Builder создать основу для проекта подключаемой библиотеки.

Теперь следует эту основу сохранить в заданной папке. Для этого в главном меню выбираем пункты **Files➔Save All**. Программа тут же предлагает сохранить файлы в папке **Projects** и запрашивает имя первого файла. Конечно, вы можете согласиться с предложением программы и сохранить файлы проекта в предлагаемой папке, но я обычно предпочитаю использовать свои собственные папки, чтобы файлы не потерялись при возможных перезагрузках компилятора.

Первым сохраняется файл с расширением .cpp. Дадим этому файлу название CWcore1.cpp.

Вторым сохраняется файл с расширением .bpr. Этому файлу также можно дать похожее название — CWcore1.bpr.

Следующим действием нужно создать один из самых основных файлов проекта — файл, в котором должны находиться собственно коды библиотеки. Для создания такого файла выбираем в главном меню пункты **File➔New➔Unit** и щелкаем на клавише **OK**. Новый файл сохраняем под именем CWfunc.cpp. Одновременно с этими действиями C++ Builder создает файл CWfunc.h. Об этом следует знать.

Основа для создания библиотечного файла сохранена. Давайте посмотрим, что из себя представляют сохраненные нами файлы. В папке **D:\CWcore1** будут находиться следующие файлы:

- CWcore1.bpf — файл формы, который в нашем случае никакой роли играть не будет. Файл создается автоматически. Никакое вмешательство не требуется.
- CWcore1.bpr — файл проекта, который содержит в себе всю необходимую информацию для компилятора. Файл создается автоматически и не допускает вмешательства.

- CWcore1.cpp — главный файл проекта, в котором содержится главная функция проекта — **DllEntryPoint()**, подобно тому как в обычных Windows приложениях главной функцией является **WinMain()**.
- CWcore1.res — файл с описанием ресурсов.
- CWfunc.cpp — файл, в котором содержатся все исходные коды программы.
- CWfunc.h — заголовочный файл, в котором объявляются функции, описанные в файле DLLfunc.cpp.

Просмотрим содержание на данный момент главного файла проекта — файла CWcore1.cpp, расположенного в листинге 3.1.

Листинг 3.1. Файл CWcore1.cpp

```
//---------------------------------------------------------------------
#include <vcl.h>
#include <windows.h>
#pragma hdrstop
//---------------------------------------------------------------------
//Important note about DLL memory management when your DLL uses the
//static version of the RunTime Library:
//---------------------------------------------------------------------
//If your DLL exports any functions that pass String objects (or structs/
// classes containing nested Strings) as parameter or function results,
//you will need to add the library MEMMGR.LIB to both the DLL project and
//any other projects that use the DLL.  You will also need to use
//MEMMGR.LIB
//if any other projects which use the DLL will be performing new or //dele-
te
//operations on any non-TObject-derived classes which are exported from
//the
//DLL. Adding MEMMGR.LIB to your project will change the DLL and its //cal-
ling
//EXE's to use the BORLNDMM.DLL as their memory manager.  In these cases,
//the file BORLNDMM.DLL should be deployed along with your DLL.
//
//To avoid using BORLNDMM.DLL, pass string information using "char *" or
//ShortString parameters.
//
//If your DLL uses the dynamic version of the RTL, you do not need to
//explicitly add MEMMGR.LIB as this will be done implicitly for you
//
#pragma argsused
int WINAPI DllEntryPoint(HINSTANCE hinst, unsigned long reason, void* lpRe-
served)
{
        return 1;
}
//
```

В начале файла находится пространное предупреждение, которое относится к организации управления памятью .DLL при использовании статической библиотеки. В основном здесь рекомендуется при передаче в функцию и из функции строк использовать тип строки (char*), а не AnsiString. В этих случаях не придется использовать MEMMGR.LIB и BORLNDMM.DLL совместно с .DLL.

Далее располагается главная функция проекта, в тело которой в необходимых случаях можно вводить свои коды. К нашему варианту это также не относится.

Посмотрим содержание заголовочного файла CWfunc.h, расположенное в листинге 3.2. Файл пока выглядит очень коротким.

Листинг 3.2. Файл CWfunc.h

```
#ifndef CWfuncH
#define CWfuncH
//---------------------------------------------------------------------
#endif
//---------------------------------------------------------------------
```

Как известно, этот файл создается и расширяется автоматически по мере заполнения кодами файла CWfunc.cpp и пока является практически пустым.

Также пустым является и файл CWfunc.cpp, текст которого расположен в листинге 3.3.

Листинг 3.3. Файл CWfunc.cpp

```
//---------------------------------------------------------------------
#pragma hdrstop
#include "CWfunc.h"
//---------------------------------------------------------------------
#pragma package(smart_init)

//---------------------------------------------------------------------
```

Из подключаемых файлов в этом листинге значится пока один файл CWfunc.h.

Статическое связывание DLL

Мною предлагается, на базе созданной основы, создать библиотечный файл, в котором будут находиться функции, необходимые для создания звукового воспроизведения различных знаков телеграфного сигнала («морзянки»). При этом управляющая программа должна быть постоянно связана с библиотекой .DLL. Это значит, что мы должны выбрать метод «статического связывания DLL». Рассмотрим вопрос «связывания» DLL-библиотеки и управляющего этой библиотекой приложения более подробно.

Использование DLL-библиотеки в приложении

Для того чтобы функции динамически подключаемой библиотеки стали доступны в прикладной программе, они должны быть подключены, или, говоря иначе, импортированы. Импорт функций динамической библиотеки может осуществляться двумя различными способами:

- с помощью директивы компилятора **_declspec(dllimport)** (статический импорт);
- вручную с использованием функций **LoadLibrary** и **GetProcAddress** (динамический импорт).

Ранее было решено, что мы будем использовать статический импорт, который является для нашего случая более удобным и используется намного чаще.

В случае статического импорта в исходный текст работающей с библиотекой программы необходимо поместить объявления функций динамически подключенной библиотеки, которые предполагается вызывать. В объявлениях следует использовать директиву **_declspec** с параметром **dllimport**.

```
Extern "C" __declspec (dllimport)  <имя функции 1>();
Extern "C" __declspec (dllimport)  <имя функции 2>();
```

Создание файла CWfunc.cpp

Несколько лет тому назад мною была разработана программа под названием CW_QSO, которая предназначалась для приема и передачи, с помощью любительской радиостанции, телеграфных сигналов и текстов. Программу CW_QSO можно взять в Интернете на моем сайте, расположенном по адресу: http://ra3xb.narod.ru/. Там же в разделе «Исходные коды» имеются тексты на языке C++ исходных кодов этой программы. Так что мы смело можем взять из этого материала подпрограмму передачи телеграфного сигнала и поместить ее в файл CWfunc.cpp.

В листинге 3.4 как раз и приведен текст этого файла, заполненного подпрограммой передачи сигнала из программы CW_QSO.

Листинг 3.4. Файл CWfunc.cpp

```
//---------------------------------------------------------------------
#pragma hdrstop
#include <vcl.h>
#include "CWfunc.h"
#include <time.h>
#include <stdio.h>
#include <sys/timeb.h>
#include <dos.h>
#include <mmsystem.h>
//---------------------------------------------------------------------
#pragma package(smart_init)
```

```
 UINT tabl[] =
 {0x55,0x31,0x40,0x32,0x3f,0x2f,0x27,0x23,0x21,0x20,0x30,0x38,0x3c,0x3e,
  0x78,0x36,0x2a,0x45,0x28,0x4c,0xc5,0x05,0x18,0x1a,0x0c,0x02,0x12,0x0e,
  0x10,0x04,0x17,0x0d,0x14,0x07,0x06,0x0f,0x16,0x1d,0x0a,0x08,0x03,0x09,
  0x11,0x0b,0x19,0x1b,0x1c,0x6d,0x73,0x6d,0x7d,0x80,0x13,0x05,0x18,0x1a,
  0x0c,0x02,0x12,0x0e,0x10,0x04,0x17,0x0d,0x14,0x07,0x06,0x0f,0x16,0x1d,
  0x0a,0x08,0x03,0x09,0x11,0x0b,0x19,0x1b,0x1c,0x1f,0x24,0x15,0x1e,0x80};
 UINT tab2[] = {
 'ю','а','б','ц','д','е','ф','г','х','и','й','к','л','м','н','о',
 'п','щ','р','с','т','у','ж','в','ь','ы','з','ш','э','я','ч','ъ',
 'Ю','А','Б','Ц','Д','Е','Ф','Г','Х','И','Й','К','Л','М','Н','О',
 'П','Щ','Р','С','Т','У','Ж','В','Ь','Ы','З','Ш','Э','Я','Ч','Ъ',
 '`','A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P',
 'Q','R','S','T','U','V','W','X','Y','Z','{','|','}','~',88,
 '`','a','b','c','d','e','f','g','h','i','j','k','l','m','n','o',
 'p','q','r','s','t','u','v','w','x','y','z','{','|','}','~',88};

int ton = 1000;       // объявляется переменная величина ton
int skorost=10;       // объявляется переменная величина skorost
int flag_zw = 1;      // объявляется переменная величина flag_zw
String variant = "1.01";     // объявляется переменная строка "1.01"
String FileName;

void Delay(int); // объявляется функция, часто используемая в расчетах
void Pauza(int); // объявляется функция, часто используемая в расчетах
void Zwuk(int);  // объявляется функция, часто используемая в расчетах
void Sound(int); // объявляется функция, часто используемая в расчетах
//---------------------------------------------------------------------
```

Начинается файл как обычно, с подключения заголовочных файлов. В дополнение к тем файлам, которые находились в заготовке (листинг 3.3), нужно дописать файлы, используемые добавляемыми функциями. Такими файлами являются `<vcl.h>`, `<time.h>`, `<stdio.h>` и другие файлы.

Далее, после строки `#pragma package(smart_init)`, записываются две таблицы для кодирования телеграфных сигналов, т. е. для перевода обычных буквенных символов русского или латинского алфавитов в наборы точек и тире — символы кода Морзе.

Затем следуют объявления глобальных переменных и функций.

Листинг 3.5 начинается с описания функций, которые будут экспортироваться из библиотеки. Пронумеруем, как обычно, эти функции.

Листинг 3.5. Продолжение 1 файла CWfunc.cpp

```
//---------------------------------------------------------------------
// Функция выдает контрольную фразу.
void Say(char *WhatToSay)                                  // 1
 {
```

```
 ShowMessage("Это сообщение от DLL\n"+(String)WhatToSay);
  }
//---------------------------------------------------------------------
// Функция выдает вариант данной библиотеки
String GetVariant(void)                                    // 2
 {
 return(variant);
  }
//---------------------------------------------------------------------
// Функция устанавливает величину скорости
void SetSkorost(int c)                                     // 3
 {
   skorost = c;
  }
//---------------------------------------------------------------------
// Функция устанавливает звуковую частоту сигнала
void SetTXFrequency(int c)                                 // 4
 {
  ton = c;
      switch(ton)
      {
        case 800:   FileName = "zw800.wav";  break;
        case 1000:  FileName = "zw1000.wav"; break;
        case 1200:  FileName = "zw1200.wav"; break;
        case 1400:  FileName = "zw1400.wav"; break;
        case 1600:  FileName = "zw1600.wav"; break;
          default:  FileName = "zw1000.wav"; break;
      }
  }
//---------------------------------------------------------------------
// Функция устанавливает контрольную величину - «флаг»
void SetFlagZw(int c)                                      // 5
 {
 flag_zw = c;
  }
//---------------------------------------------------------------------
```

Функции 1...5 служат для обмена данными между основным приложением и библиотекой. Функция 6, с которой начинается листинг 3.6, может считаться основной в данной DLL, т. к. именно она выполняет функции перекодирования символа в код Морзе и организует выдачу звукового сигнала.

Листинг 3.6. Продолжение 2 файла CWfunc.cpp

```
//---------------------------------------------------------------
// Функция преобразования буквенного символа в код Морзе
void TXsimwol(char simwol)                                 // 6
{
  int count;
  UINT dd,bb;
```

```
  UINT ch_z;
  int i;
  int j;
  UINT ch_i = simwol;
//  const int AA = 44;

  if(ch_i == 0x20) {Pauza(8); return;}
  if(ch_i < 44 ) return;
  if(ch_i < 127 ) {goto whod1;}
  else{
   for(j=0;j<=64;j++)
   {
   if(tab2[j] == ch_i)
      {
   ch_i = tab2[j + 64]; break;}
      }
    }
 whod1:
  i = ch_i - 44;
  ch_z = tabl[i];
       dd = 8;
  kt:  ch_z = ch_z<<1;
       dd--;
       bb = ch_z & 0x100;
       if(dd == 0) goto wyh;
       if(bb == 0) goto kt;
  g:   ch_z = ch_z<<1;
       bb = ch_z & 0x100;
       if(bb == 0) {count = 2; Zwuk(count); goto k2;}
       count = 7;
       Zwuk(count);
 k2:      count = 2;
       Pauza(count);
       dd--;
       if(dd == 0) goto wyh1;
       goto g;
 wyh1: count = 4;
       Pauza(count);
   wyh:
  return;
 }
//---------------------------------------------------------------------
// Функция создает и выдерживает паузу между посылками
 void Pauza(int count)                                    // 7
      {
    int j;
      for(j=0; j<=count;j++) {
      Sleep(skorost);
      };
      return;
```

```
      }
//---------------------------------------------------------------------
// Функция выдает телеграфную звуковую посылку
 void Zwuk(int count)                                        // 8
     {
       if(flag_zw == 1)
 PlaySound(FileName.c_str(),NULL,SND_ASYNC);
      else  PlaySound(NULL,NULL,NULL);
      for(int j=0;j<=count;j++)  {
         Sleep(skorost);
         };
  PlaySound(NULL,NULL,NULL);
       return;
          }
//-----------------------------------------------------
```

Для нормальной работы создаваемой нами библиотеки необходимы дополнительные файлы, в которых записан гармонический звук определенной частоты. Это сделано только для упрощения исходных кодов программы, т. е. для более наглядного освоения происходящих процессов создания телеграфных звуковых посылок. Файлы с записанным в них звуком определенной частоты можно сделать самому или взять в Интернете на моей WEB-странице по адресу: http://ra3xb/narod.ru/.

Следующим шагом нужно добавить в имеющийся файл CWfunc.h необходимые записи объявления каждой из примененных в DLL экспортируемых функций.

После добавления нужных записей файл будет выглядеть точно так, как на листинге 3.7.

Листинг 3.7. Файл CWfunc.h

```
//---------------------------------------------------------------------
#ifndef CWfuncH
#define CWfuncH
//---------------------------------------------------------------------
#include <windows.h>
#include <SysUtils.hpp>
//---------------------------------------------------------------------
extern "C" void __declspec(dllexport) Say(char *WhatToSay);
extern "C" void __declspec(dllexport) SetTXFrequency(int);
extern "C" void __declspec(dllexport) SetSkorost(int);
extern "C" void __declspec(dllexport) SetFlagZw(int);
extern "C" String __declspec(dllexport) GetVariant(void);
extern "C" void __declspec(dllexport) TXsimwol(char);
#endif
//---------------------------------------------------------------------
```

Таким образом, все файлы оказались подготовленными к компиляции, но нужно выполнить еще одно действие. Выбираем в главном меню пункты **Projrct→Op-**

tijns➔Lincer. На этой странице следует убедиться, что включен индикатор опции **Generate import library** (см. рис. 3.3).

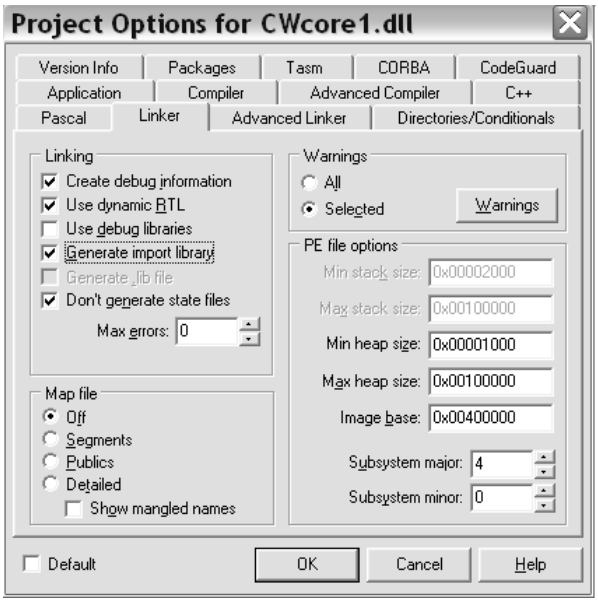

Рис. 3.3. Окно Project Options

Этот индикатор обеспечивает при создании DLL автоматическую генерацию (создание) вспомогательного файла с расширением **.lib**, который необходим для статического присоединения DLL к проектам приложений.

Последний этап: выбираем в меню **Project➔Build CWcore**. В результате получаем в заданной директории два файла: выполняемый файл **CWcore1.dll** и файл библиотеки **CWcore1.lib**.

О файле библиотеки *.lib

Файл с расширением .lib необходим для статического связывания выполняемого файла и основного приложения. Этот файл должен быть обязательно включен в состав проекта основного приложения. Делается эта процедура следующим образом:

- Файл с расширением .dll (в нашем случае это CWcore1.dll) и файл с расширением .lib (в нашем случае это файл CWcore1.lib) копируются в директорию, в которой находятся все файлы проекта основного приложения.
- В главном меню выбираются пункты **Project➔Add to Project**. В появившемся диалоговом окне **Add to project** (см. рис. 3.4) в самом нижнем поле выбора «Тип файла» выбираем **Library file [*.lib]**. В главном окне появляется название файла библиотеки (в нашем случае — CWcore1). Щелкаем на названии этого файла мышкой, затем нажимаем клавишу **Открыть**. Библиотека связана с основным приложением.

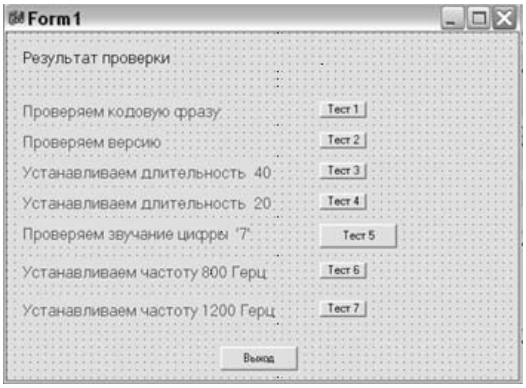

Рис. 3.4. Окно Add to project

Иногда может получиться так, что у вас имеется только файл *.dll. В таком случае создать файл *.lib можно с помощью утилиты IMPLIB, имеющейся в составе C++ Builder. В этом случае утилита запускается из командной строки следующим образом: **IMPLIB <имя файла>.lib <имя файла>.dll**.

Тестируем созданную DLL

Убедиться в работоспособности созданной нами CWcore.dll можно путем создания очень простой вспомогательной программы **TestCWcore**.

Как обычно, создаем для нового проекта директорию, например, d:\TestCWcore\, создаем описанным многократно способом в этой директории проект нового приложения **TestCWcore**. Копируем в эту директорию файлы CWcore1.dll и CWcore1.lib. Командой **Add to Project** подключаем файл CWcore.lib к проекту основного приложения.

На форму нового проекта устанавливаем компоненты в соответствии с рис. 3.5.

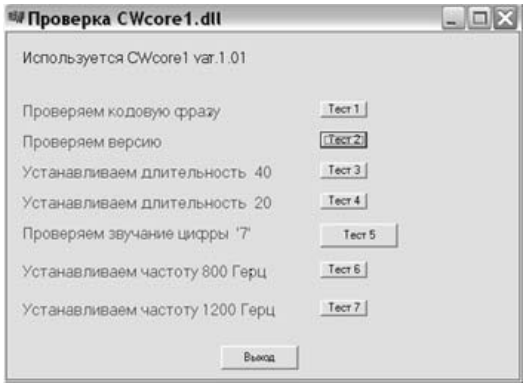

Рис. 3.5. Форма нового проекта

На форму устанавливаются простейшие компоненты типа **Label** и **Button**, так что никаких трудностей с их установкой быть не должно.

Ниже, в листинге 3.8, приведено содержание файла TestCWcore.cpp.

Листинг 3.8. Файл TestCWcore.cpp

```
 //---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include <mmsystem.h>;
#include "TestCWcore.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
// Ниже даны объявления функций для импорта из библиотеки
extern "C" __declspec(dllimport) void Say(String);
extern "C" __declspec(dllimport) String GetVariant(void);
extern "C" __declspec(dllimport) void SetSkorost(int);
extern "C" __declspec(dllimport) void SetTXFrequency(int);
extern "C" __declspec(dllimport) void SetFlagZw(int);
extern "C" __declspec(dllimport) void TXsimwol(char);

 int skorost, ton;   // Здесь и ниже даны объявления переменных величин
 int flag_zw;
 void Say(String);   // Объявление функций
 void Sound(int);
//---------------------------------------------------------------------------
// функция описания формы
__fastcall TForm1::TForm1(TComponent* Owner)         // 1
        : TForm(Owner)
{
 Caption="Проверка CWcore1.dll";
}
//---------------------------------------------------------------------------
// Функция отклика на нажатие клавиши «Выход»
void __fastcall TForm1::Button1Click(TObject *Sender)    // 2
{
 Close();
}
//---------------------------------------------------------------------------
// Функция отклика на нажатие клавиши Тест 1
void __fastcall TForm1::Button2Click(TObject *Sender)     // 3
{
String s;
 Say("Привет, здесь DLL!");
}
//---------------------------------------------------------------------------
// Функция отклика на нажатие клавиши Тест 2
```

```
void __fastcall TForm1::Button3Click(TObject *Sender)      // 4
{
String s;
 s = GetVariant();
 Label1->Caption ="CWcore1 var."+s;
}
//---------------------------------------------------------------------
// Функция отклика на нажатие клавиши Тест 3
void __fastcall TForm1::Button4Click(TObject *Sender)       // 5
{
 skorost = 40;
 Label1->Caption ="Сейчас длительность = "+IntToStr(skorost);
 SetSkorost(skorost);
}
//---------------------------------------------------------------------
// Функция отклика на нажатие клавиши Тест 5
void __fastcall TForm1::Button6Click(TObject *Sender)        // 6
{
 char sim = '7';
TXsimwol(sim);
}
//---------------------------------------------------------------------
// Функция отклика на нажатие клавиши Тест 4
void __fastcall TForm1::Button5Click(TObject *Sender)         // 7
{
 skorost = 20;
 Label1->Caption ="Сейчас длительность = "+IntToStr(skorost);
 SetSkorost(skorost);
}
//---------------------------------------------------------------------
// Функция отклика на нажатие клавиши Тест 6
void __fastcall TForm1::Button7Click(TObject *Sender)          // 8
{
ton = 800;
 SetTXFrequency(ton);
Label1->Caption ="Сейчас частота = "+IntToStr(ton)+" Герц";
}
//---------------------------------------------------------------------
// Функция отклика на нажатие клавиши Тест 7
void __fastcall TForm1::Button8Click(TObject *Sender)          // 9
{
ton = 1200;
SetTXFrequency(ton);
Label1->Caption ="Сейчас частота = "+IntToStr(ton)+" Герц";
}
//---------------------------------------------------------------------
```

Думаю, что нет необходимости более подробно расписывать входящие в файл функции.

Несколько слов о том, как следует проводить тест.

- После нажатия на клавишу **Тест 1** на экране должно появиться информационное окно. Это уже говорит о том, что библиотека подключена к программе проверки.
- Клавиша **Тест 2** вызывает на экран номер действующей версии библиотеки.
- Затем нужно клавишами **Тест 6** и **Тест 7** поочередно задавать рабочую частоту, а клавишей **Тест 5** проверять звучание телеграфной цифры '7' при той и другой частотах.
- Практически так же проверяется работа функций, регулирующих длительности звучания цифры '7'.

Надеюсь, что у вас все получится с первого же раза. Не забудьте о том, что в директории приложения должны находиться файлы с записанными в них звуковыми сигналами определенной частоты. В данном случае достаточно будет двух файлов — файла **zw800.wav** и файла **zw1200.wav**. Файлы нужно либо сделать самому, либо взять в Интернете на моей страничке. Без этих файлов телеграф звучать не будет!

Приложение использует DLL

В целях закрепления представленного в этой главе материала, в этом разделе создадим простое Windows-приложение, которое будет использовать созданную нами ранее подключаемую библиотеку **CWcore1.dll**.

Предлагаемое мною приложение представляет собой простой тренажер для самостоятельного изучения приема телеграфных сигналов кода Морзе на слух. Назовем эту программу **WinCW1**, что должно обозначать «Телеграф (CW1) для Windows».

Начинаем, как обычно, с создании директории, например, D:\WinCW1\. В этой директории создаем проект приложения **WinCW1**. Все эти процедуры вам уже хорошо знакомы, а если кто-то запамятовал, то обратитесь к моментам создания проектов в предыдущих проектах.

Не забудьте в эту директорию скопировать файлы **CWcore1.dll** и **CWcore1.lib**, а также файлы с записанными звуковыми сигналами — файлы **zw800.wav**, **zw1000.wav**, **zw1200.wav** и другие аналогичные. Вполне возможно, что вы захотите обойтись только одним или двумя звуковыми файлами.

Подключите к проекту файл **CWcore1.lib**, который связывает приложение с библиотекой, посредством выбора пунктов меню **Add to Project**.

Начнем с создания формы и ее заполнения компонентами. На рис. 3.6 представлен предлагаемый вариант формы.

Новыми компонентами, не встречавшимися в прежних проектах, являются для вас установленные три компонента выбора из списка — компоненты **ComboBox**.

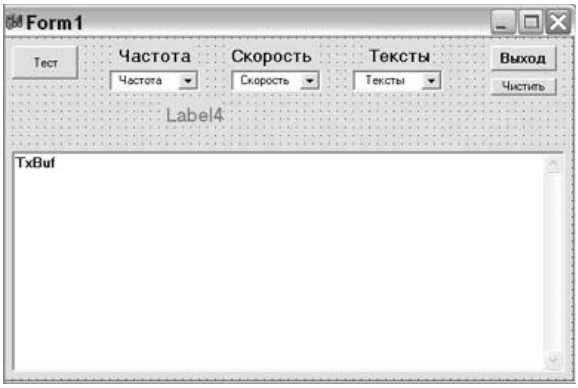

Рис. 3.6. Форма нового проекта

Один из этих компонентов служит для выбора частоты генерируемого звукового сигнала, второй для выбора скорости передачи, третий — для выбора определенного текста. Краткое описание этого компонента можете найти в *главе 1* этой книги.

После установки компонента **ComboBox** на форму необходимо в свойствах этого компонента выбрать строку **Items** и щелкнуть на расположенном справа квадратике. Появляется окно **String List Editor** — редактор перечня строк (см. рис. 3.7).

Рис. 3.7. Окно для редактирования выбираемых строк

Вначале текстовое окно пустое, курсор находится в левом верхнем углу. Печатаем текст первой строки и нажимаем клавишу **<Enter>** для перевода строки. Затем таким же образом печатаем последующие строки. В заключение щелкаем на **OK**. Свойство **ItemIndex** оставляем равным −1.

Точно таким же образом оформляем перечни выбираемых строк и в оставшихся двух компонентах **ComboBox**.

Давайте сначала посмотрим файл WinCW.h, чтобы познакомиться с названиями примененных компонентов и использованных в программе функций.

Листинг 3.9. Файл WinCW.h

```
 //---------------------------------------------------------------------------
#ifndef WinCWH
#define WinCWH
//
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Buttons.hpp>
//---------------------------------------------------------------------------
class TForm1 : public TForm
{
__published:   // IDE-managed Components
        TMemo *TxBuf;
        TButton *Button1;
        TComboBox *ComboBox1;
        TSpeedButton *SpeedButton1;
        TComboBox *ComboBox2;
        TComboBox *ComboBox3;
        TLabel *Label1;
        TLabel *Label2;
        TLabel *Label3;
        TButton *Button2;
        TLabel *Label4;
        void __fastcall SpeedButton1Click(TObject *Sender);
        void __fastcall TxBufKeyPress(TObject *Sender, char &Key);
        void __fastcall Button1Click(TObject *Sender);
        void __fastcall ComboBox2Change(TObject *Sender);
        void __fastcall ComboBox3Change(TObject *Sender);
        void __fastcall ComboBox1Change(TObject *Sender);
        void __fastcall Button2Click(TObject *Sender);
private:       // User declarations
public:
// User declarations
        __fastcall TForm1(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TForm1 *Form1;
//---------------------------------------------------------------------------
#endif
//---------------------------------------------------------------------------
```

Мне думается, что не стоит комментировать приведенный листинг. Вы уже имеете опыт рассмотрения подобных файлов из ранее рассмотренных проектов. Следует отметить только тот факт, что файл полностью сгенерирован программой и добавлять в него что-либо от себя пока не стоит.

Далее, в листинге 3.10, приведен текст файла WinCW.cpp.

Листинг 3.10. Файл WinCW.cpp

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include <mmsystem.h>
#include "WinCW.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;

extern "C" __declspec(dllimport) void Say(String);
extern "C" __declspec(dllimport) String GetVariant(void);
extern "C" __declspec(dllimport) void SetSkorost(int);
extern "C" __declspec(dllimport) void SetTXFrequency(int);
extern "C" __declspec(dllimport) void SetFlagZw(int);
extern "C" __declspec(dllimport) void TXsimwol(char);

 char chz;
 int skorost, ton;
 int flag_zw;
 void Sound(int);       // функция добавлена автором
 void SimTX(char);      // функция добавлена автором
 void ClearBuf1(void);  // функция добавлена автором

 String FileName;
 String TextName;
 char buff1[4000];
//---------------------------------------------------------------------------
```

В начале листинга 3.10 выполнено, как обычно, подключение необходимых заголовочных файлов. Строка `TForm1 *Form1;` представляет собой объявление формы **Form1**. Далее идут строки, в которых выполняется объявление функций, импортируемых из подключаемой библиотеки. Строка объявления выглядит так:

```
extern "C" __declspec(dllimport) String GetVariant(void);
```

В этой строке объявляется функция **GetVariant()**, предназначенная для получения от библиотеки сведений об варианте исполнения данной версии библиотеки. Если вы помните, то в самой подключаемой библиотеке эта функция объявлялась как экспортируемая и имела вид:

```
extern "C" __declspec(dllexport) String GetVariant(void);
```

Подобные объявления дают возможность четкого выполнения функции.

Далее, до конца листинга, идут обычные объявления глобальных переменных величин, а также функций, разработанных автором и вставленных в этот файл.

Листинг 3.11 начинается с функции описания формы. В тело этой функции добавлена строка, устанавливающая заголовок для главного окна программы, и

Содержание