ЛАБОРАТОРНАЯ РАБОТА №2

БИБЛИОТЕКИ ДИНАМИЧЕСКОЙ КОМПОНОВКИ (DLL)

ОГЛАВЛЕНИЕ

1. ПРИМЕНЕНИЕ БИБЛИОТЕКИ DLL

Применение библиотек динамической компоновки на практике подразумевает наличие в них специальных функций, которые могут быть вызваны из других приложений или DLL. Такие функции называются **экспортируемыми**. Вызывающее приложение определенным образом получает информацию об адресе и характеристиках экспортируемой функции (импортирует функцию), а затем осуществляет собственно ее вызов с передачей соответствующих параметров. Таким образом, приложение получает возможность задействовать все преимущества, которые дает применение библиотек динамической компоновки при разработке программного обеспечения.

Как правило, типичная DLL содержит некоторое количество внутренних функций, выполняющих реальную работу, а также несколько экспортируемых функций, которые играют роль интерфейса между DLL и приложениями. При этом экспортируемые функции могут вызывать внутренние функции DLL, что, вообще говоря, является хорошей практикой, так как обеспечивает инкапсуляцию и высокую степень внутренней связности в DLL.

Стандартной точкой входа для DLL, т.е. аналогом WinMain для приложений является функция DllEntryPoint() (еще она может называться DllMain()). Она применяется достаточно редко – как правило, в случаях, когда требуется выполнять какие-то инициализационные действия при загрузке DLL, поскольку в этом случае DllEntryPoint() вызывается автоматически. Тем не менее, DllEntryPoint() является обязательным элементом библиотеки DLL, хотя и содержит обычно только оператор выхода.

Рассмотрим код простой библиотеки DLL, экспортирующей только одну функцию:

```
//---------------------------------------------------
#include <windows.h>

LPCWSTR GetSomeString ()
{
   return L"Hello from DLL!\n";
}

LPCWSTR  __declspec (dllexport) Test (void)
{
   return GetSomeString ();
}

BOOL WINAPI DllEntryPoint (HINSTANCE, DWORD, DWORD)
{
    return 1;
}
//---------------------------------------------------
```

Функция GetSomeString() является внутренней функцией DLL и не экспортируется. Таким образом, другие приложения не имеют прямого доступа к ее коду. В приведенном примере функция GetSomeString() выполняет некоторую полезную работу, т.е. возвращает определенную строку, которая используется вызывающим приложением.

Функция Test() является экспортируемой, т.е. она доступна другим приложениям. Для этого она должна быть описана со специальным модификатором, который определяется используемой версией компилятора (в приведенном примере **__declspec (dllexport)**). При использовании этого модификатора информация о функции будет помещена в таблицу экспорта DLL и станет доступна внешним приложениям.

Таким образом, после успешной сборки DLL и заполнения таблицы экспорта внешние приложения смогут вызывать функцию Test() и, соответственно, получать в свое распоряжение строку, возвращаемую функцией GetSomeString(), т.е. задействовать функционал, предоставляемый библиотекой DLL.

Вызов функций из DLL существенно отличается в зависимости от того, применяется неявное или явное связывание библиотеки DLL с приложением. При **неявном связывании** информация о вызываемой функции и библиотеке DLL, в которой она реализована, помещается в таблицу импорта вызывающего приложения, и оно получает всю необходимую информацию для вызова функции. Это действие выполняется на этапе компоновки приложения с помощью т.н. библиотеки импорта. Такая библиотека создается с помощью специальной утилиты LIB. Полученная библиотека импорта содержит сведения о функциях, экспортируемых из DLL. Таким образом, она является посредником между DLL, которая экспортирует функции, и EXE-файлом, который эти функции использует.

Поскольку вызываемая из DLL функция является внешней по отношению к вызывающему приложению, она должна обязательно иметь прототип в вызывающей программе, определенный в точном соответствии с его описанием в DLL. При этом прототип обязательно должен иметь модификатор **extern**, показывающий, что функция является внешней по отношению к данному модулю, и ее код следует искать где-то во внешних модулях (в данном случае в DLL). Ниже приведен код простого приложения, использующего неявное связывание для вызова функции из DLL:

```
//---------------------------------------------------
#include <windows.h>

extern LPCWSTR Test (void);
```

```
int WINAPI WinMain (HINSTANCE, HINSTANCE, LPSTR, int)
{
    LPCWSTR str = Test ();
    return MessageBox (NULL, str, L"Вызов из DLL!",
                       MB_OK | MB_ICONEXCLAMATION);
}
//-----------------------------------------------------
```

Как видно, приложение содержит прототип функции Test(), совпадающий с ее описанием в коде библиотеки DLL (за исключением модификатора __declspec (dllexport)), и имеющий модификатор **extern.** Это дает возможность при компоновке с использованием соответствующей библиотеки импорта напрямую осуществлять вызов функции DLL, неявно связанной с приложением, как показано в коде функции WinMain(). Данная схема вызова удобна тем, что не требует от программиста никаких дополнительных усилий для связи с DLL и вызова нужной функции.

При **явном связывании** вызов функции осуществляется иначе и требует от программиста следующих явных действий:

- загрузка DLL (через LoadLibrary())
- получение адреса нужной функции (через GetProcAddress())
- выгрузка DLL (через FreeLibrary())

Ниже приведен пример приложения, демонстрирующего вызов функции при явном связывании с DLL:

```
//-----------------------------------------------------
#include <windows.h>

typedef  LPCWSTR  (*pfn) (void);

int WINAPI WinMain (HINSTANCE, HINSTANCE, LPSTR, int)
```

```
{
   HMODULE hMod = LoadLibrary (L"testdll.dll");
   if (!hMod)
      return MessageBox (NULL, L"Ошибка загрузки
                          testdll.dll!\n", L"ОШИБКА!",
                          MB_OK | MB_ICONEXCLAMATION);

   pfn addr = (pfn)GetProcAddress (hMod, "Test");

   if (!addr)
      return MessageBox (NULL, L"Ошибка получения
                         адреса функции!\n", L"ОШИБКА!",
                          MB_OK | MB_ICONEXCLAMATION);

   LPCWSTR str = (*addr)();

   if (!str)
      return MessageBox (NULL, L"Получена пустая
                          строка!\n", L"ОШИБКА!",
                          MB_OK | MB_ICONEXCLAMATION);

   MessageBox (NULL, str, L"Вызов из DLL!",
               MB_OK | MB_ICONEXCLAMATION);

   FreeLibrary (hMod);

   return 0;
}
//-----------------------------------------------------
```

Как видно, вызов функции при явном связывании осуществляется значительно сложнее и требует, в частности, иного описания функции в самой библиотеке DLL.

Для корректного применения при явном связывании функция Test() в коде библиотеки DLL должна быть описана следующим образом:

```
extern "C" LPCWSTR  __declspec (dllexport) Test (void)
```

Наличие в описании модификатора **extern “C”** продиктовано особенностями явного связывания, согласно которым вызывающей программе должно быть известно ***внутреннее*** имя функции в DLL. Дело в том, что компиляторы C++ используют т.н. расширение имен (name mangling), предназначенное для поддержки перегруженных функций. При расширении имен внутреннее имя функции отличается от имени в тексте программы, поскольку к нему дописываются специальные символы расширения. Таким образом, попытка вызвать функцию из DLL по имени будет неудачной (имя функции, переданное вызывающим приложением, и внутреннее имя функции будут отличаться). Один из способов обхода данной проблемы – это отключение расширения имен, которое достигается с использованием специального модификатора **extern “C”**. В этом случае внутреннее имя функции будет совпадать с именем функции в коде вызывающего приложения.

Вызывающее приложение определяет указатель на функцию, имеющую тот же тип, что и функция Test(), которая в этом примере вызывается из DLL. Это необходимо, так как вызов функции при явном связывании происходит по ее указателю. Для удобства дальнейшего использования создается тип такого указателя, имеющий имя PFN.

Для вызова функции Test() из DLL вызывающее приложение вначале должно загрузить библиотеку DLL, содержащую эту функцию, в адресное пространство процесса, что осуществляется посредством функции WinAPI LoadLibrary() как показано в приведенном примере. При этом в переменной hMod сохраняется дескриптор модуля DLL, загруженной в адресное пространство процесса.

Далее посредством вызова функции WinAPI GetProcAddress() выполняется получение адреса функции Test() внутри модуля DLL, ранее

загруженной в процесс вызывающего приложения. Полученный адрес приводится к ранее определенному типу PFN. Так как теперь известен адрес функции Test(), то выполняется ее фактический вызов, в результате которого в переменную str возвращается строка, представляющая собой результат выполнения функций из библиотеки DLL. Данная строка выводится в MessageBox, на чем использование функционала DLL приложением и завершается. Так как теперь ранее загруженная DLL более не требуется, то происходит ее выгрузка из адресного пространства процесса вызывающего приложения посредством функции WinAPI FreeLibrary(), после чего вызывающее приложение завершает работу.

Следует особо отметить, что явное связывание при больших усилиях, связанных с вызовом функций из DLL, обеспечивает значительную гибкость и лучший контроль со стороны программиста, чем неявное связывание.

2. СБОРКА ДЕМОНСТРАЦИОННОГО ПРИЛОЖЕНИЯ DLL

2.1 Сборка приложения явного связывания

Для создания демонстрационного приложения, использующего динамическую библиотеку DLL, в среде разработки Microsoft Visual Studio 2010 необходимо создать соответствующее решение (solution), включающее в себя два проекта: проект собственно DLL-библиотеки и проект вызывающего приложения, как показано на приведенных ниже снимках экранов. Проекты, составляющие решение, могут быть откомпилированы и собраны в соответствующие исполняемые файлы средствами Visual Studio.

Для создания решения необходимо выбрать соответствующий пункт в меню «Файл» (или нажать комбинацию клавиш Ctrl+Shift+N), в появившемся диалоге выбрать тип проекта «Проект Win32» как показано на рис. 1. Кроме того, необходимо также указать название проекта библиотеки DLL, место его

расположения и наименование решения, в которое входит проект, в соответствующих полях.

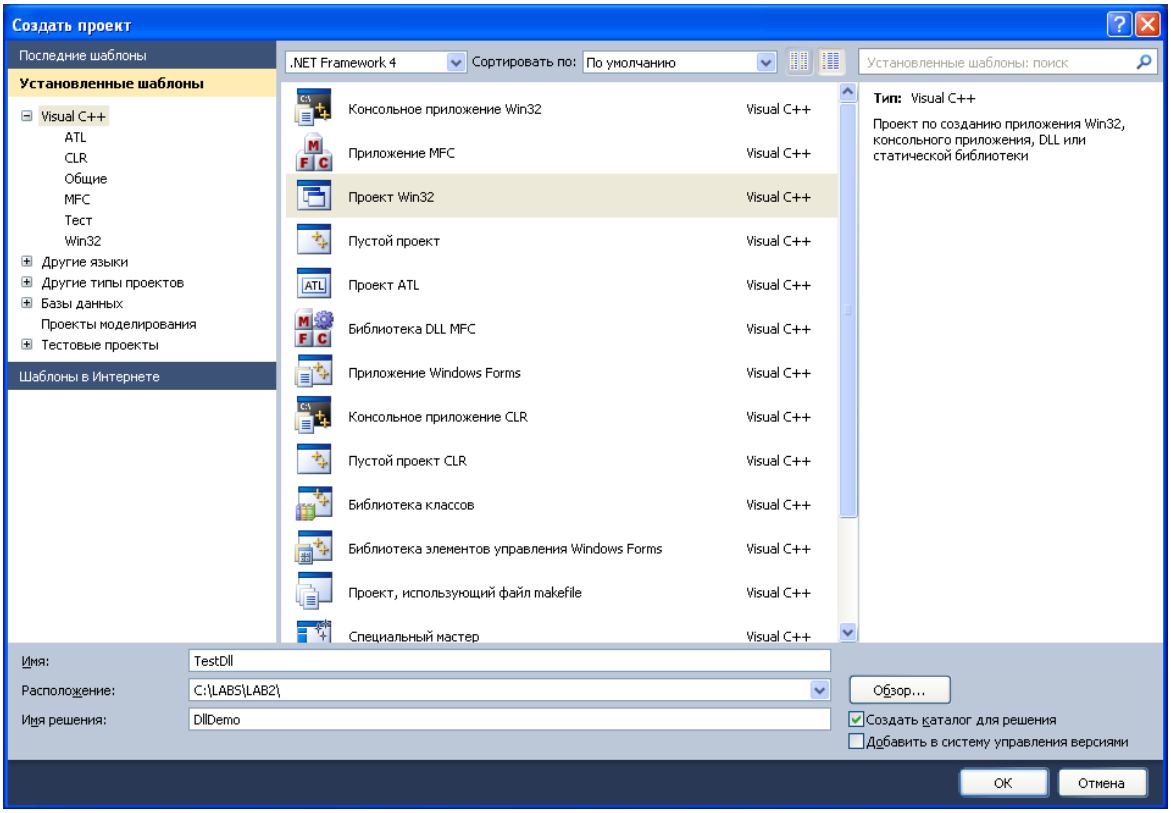

Рис. 1 – Выбор типа проекта для библиотеки DLL

В появившемся окне Мастера приложений Win32 необходимо установить параметры, как показано на рис. 2.

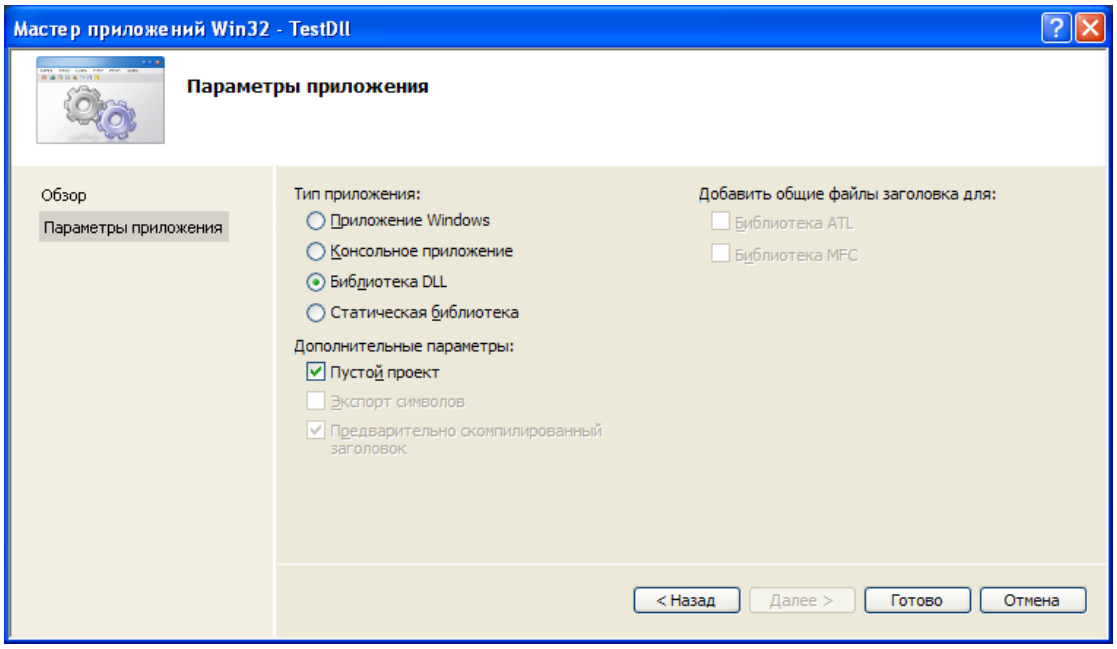

Рис. 2 – Установка параметров проекта для библиотеки DLL

После выполнения этих действий будет создан пустой проект, в который следует добавить файл, содержащий исходный код демонстрационной библиотеки DLL, либо создать такой файл непосредственно в проекте и написать там исходный код библиотеки DLL.

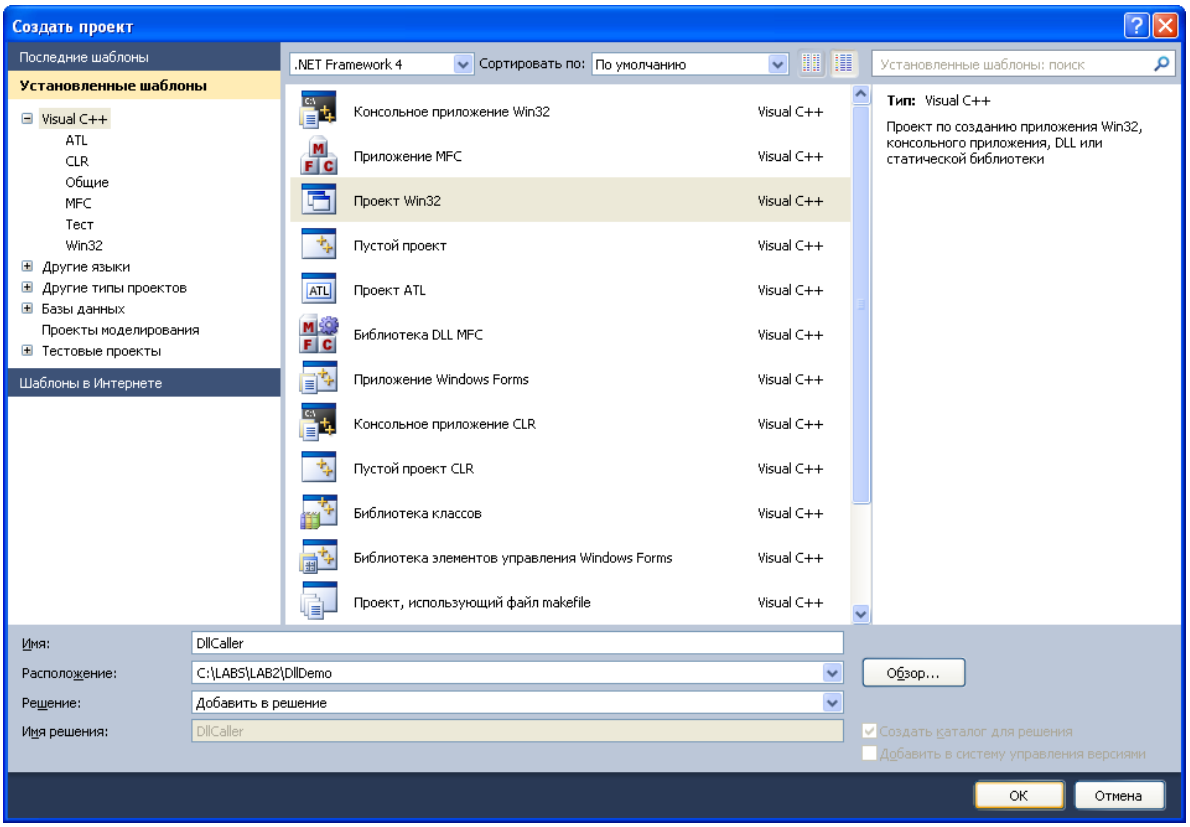

Рис. 3 – Выбор типа проекта для приложения, вызывающего DLL

Для создания проекта демонстрационного приложения, использующего демонстрационную библиотеку DLL, и добавления его в существующее решение следует выбрать соответствующий пункт в меню «Файл» (или нажать комбинацию клавиш Ctrl+Shift+N). В появившемся диалоге создания проекта следует указать имя проекта и его принадлежность к ранее созданному решению, как показано на рис. 3.

В появившемся окне Мастера приложений Win32 необходимо установить параметры, как показано на рис. 4. Далее необходимо добавить в созданный проект файл, содержащий исходный код приложения, либо создать такой файл непосредственно в проекте.

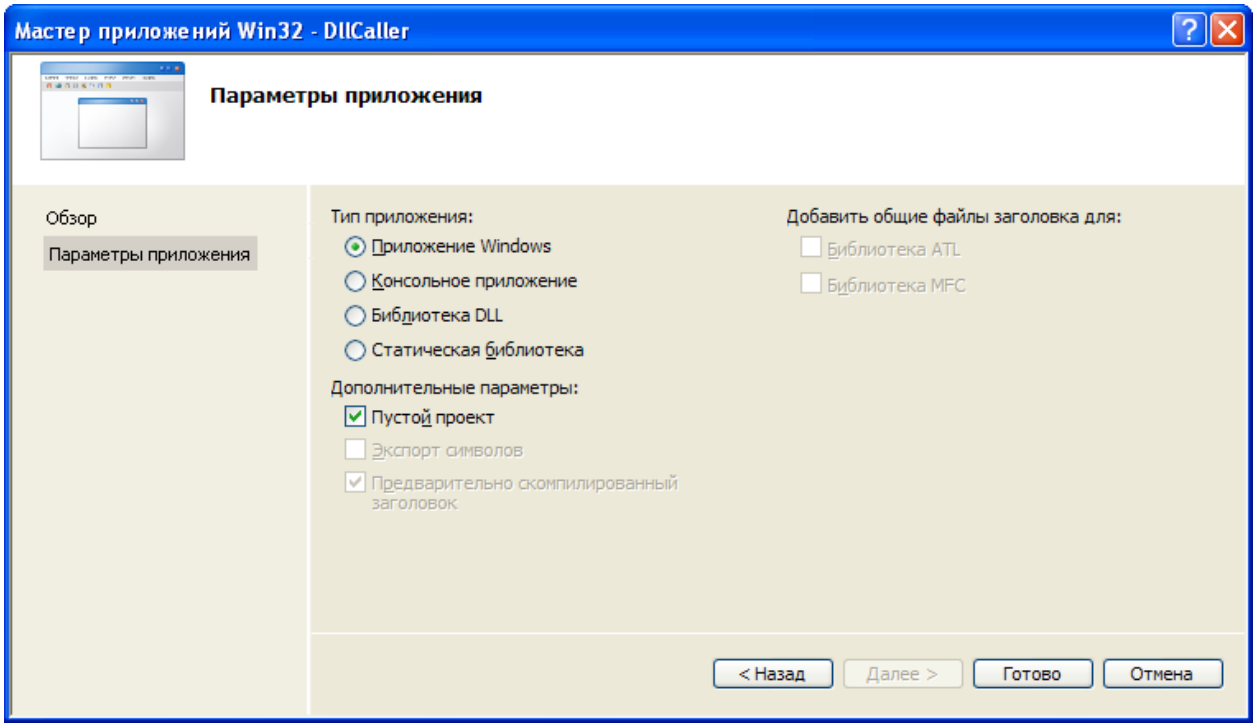

Рис. 4 – Установка параметров проекта для приложения, вызывающего DLL

После выполнения всех указанных действий будет создано решение (solution), содержащее два проекта – собственно DLL и вызывающего приложения. Далее необходимо установить порядок сборки проектов в решении и указать проект, который будет запускаться на выполнение после сборки (очевидно, что это должен быть проект вызывающего приложения). Для этого следует щелкнуть правой кнопкой на имени проекта вызывающего приложения и выбрать в появившемся меню пункт «Назначить запускаемым проектом». Также в пункте меню «Порядок построения объектов» можно указать порядок, в котором будет выполняться сборка проектов. Рекомендуется вначале выполнять сборку проекта DLL, а затем сборку проекта вызывающего приложения. Типичный вид готового к сборке решения показан на рис. 5.

После настройки можно выполнить собственно сборку и запуск демонстрационного приложения, нажав клавишу F5 или выбрав соответствующий пункт в меню.

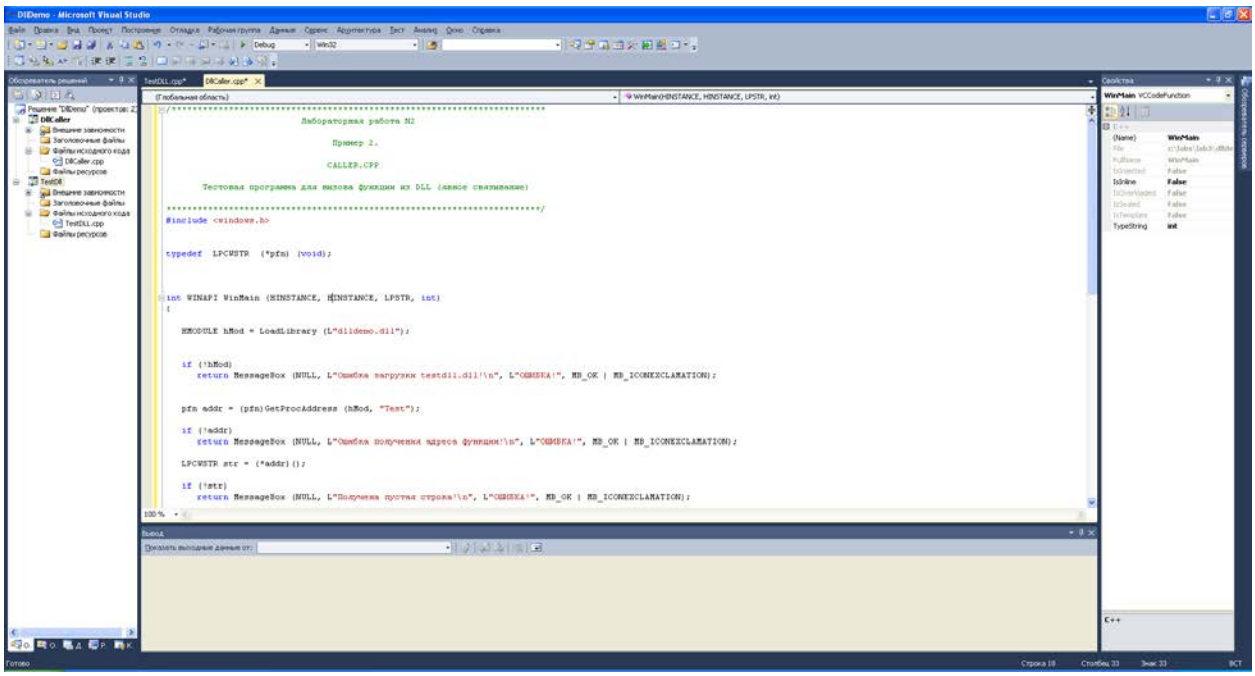

Рис. 5 – Решение настроено для сборки

2.2 Сборка приложения неявного связывания

Сборка приложения и DLL, демонстрирующих неявное связывание, производится аналогично сборке демонстрационных приложения и DLL для явного связывания за исключением важного нюанса. Так как загрузка библиотеки DLL должна производиться автоматически при запуске приложения (в отличие от явного связывания), то вызывающее приложение должно иметь информацию о функциях, экспортируемых из DLL. Для этого необходимо в проекте вызывающего приложения поставить ссылку на проект библиотеки DLL как показано на рис. 6, что позволит использовать информацию об экспортируемых из DLL функциях при сборке вызывающего приложения и корректно разрешить внешние ссылки на функции DLL.

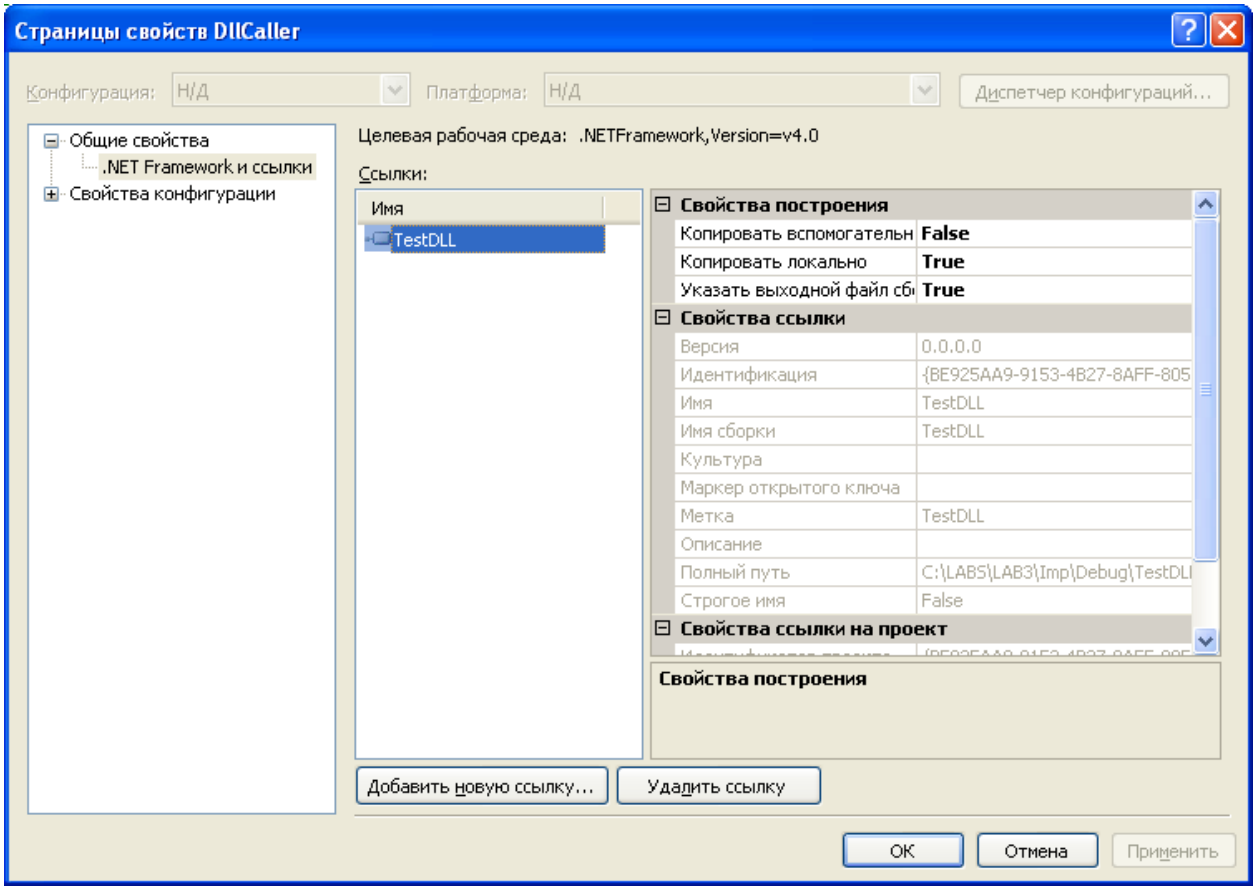

Рис. 6 – Ссылка на DLL в приложении с явным связыванием