DR. BOB DAVIDOV

DLL Lcard USB интерфейс для CPP процесса

Цель работы: освоение правил построения динамической библиотеки (DLL) и связи CPP процесса с USB устройством через библиотечные функции.

Задача работы: построение быстродействующего канала связи CPP процесса с USB устройством: L card E14-440.

Приборы и принадлежности: персональный компьютер, ОС Windows XP (Windows 2000) среда программирования Microsoft Visual C++, многоканальная измерительная USB система E14-440 с драйвером Ldevusbu.sys файла Lusbapi.inf, динамической библиотекой Lusbapi.dll, статической библиотекой Lusbapi.lib (2178 байт) для Microsoft VC, и заголовочными файлами Lusbapi.h и LusbapiTypes.h.

ОБЩИЕ СВЕДЕНИЯ

Темы:

- Создание собственной dll (visual c++ 6.0, visual c++ 9.0)
- Отображение списка экспортируемых функций dll
- Неявная загрузка dll файла
- Явная загрузка dll файла
- Отображение списка импортируемых функций DLL
- Подключение модуля E14-440 к компьютеру

С точки зрения программиста - DLL представляет собой библиотеку функций (ресурсов), которыми может пользоваться любой процесс, загрузивший эту библиотеку.

DLL широко используются в технологии COM - в качестве основы при построении так называемых inproc-серверов (внутрипроцессных серверов)

DLL не налагает ограничений на используемый язык, как правило, DLL разрабатывается на другом языке программирования, нежели тот, который используется при ее загрузке.

Необходимо помнить, что загрузка библиотеки, занимает время и увеличивает расход памяти; поэтому бездумное дробление одного приложения на множество DLL ничего хорошего не принесет. Другое дело - если какие-то функции используются несколькими приложениями. Тогда, поместив их в одну DLL, можно избавиться от дублирования кода и сократить общий объем приложений - и на диске, и в оперативной памяти. Можно выносить в DLL и редко используемые функции отдельного приложения.

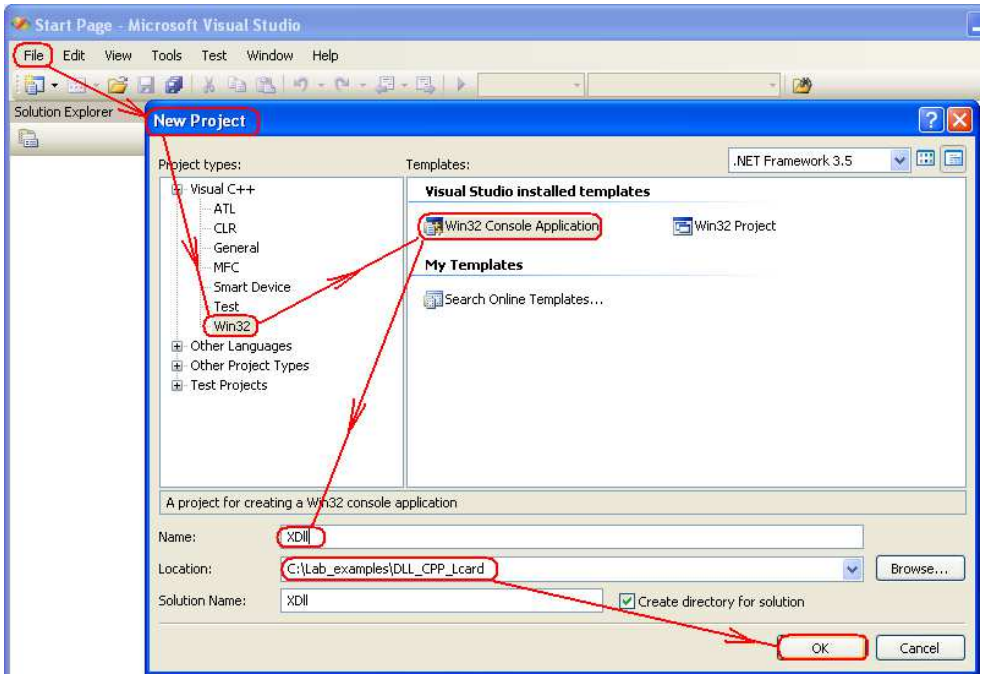
Загрузившему DLL процессу доступны не все ее функции, а лишь явно предоставляемые самой DLL для "внешнего мира" - т. н. экспортируемые. Чем больше функций экспортирует DLL - тем медленнее она загружается.

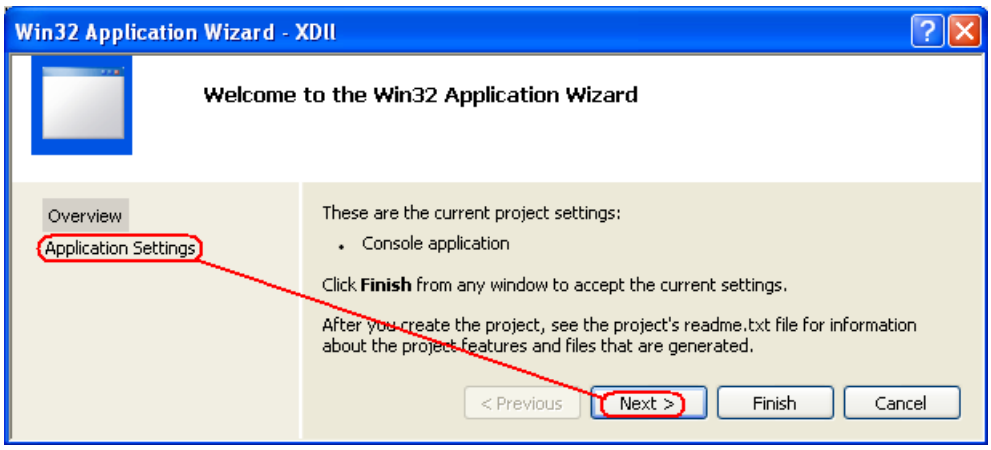
Далее здесь представлена последовательность построения DLL с экспортируемой функцией и CPP процесса который вызывает DLL и передает ей два параметра, DLL складывает параметры и возвращает результат.

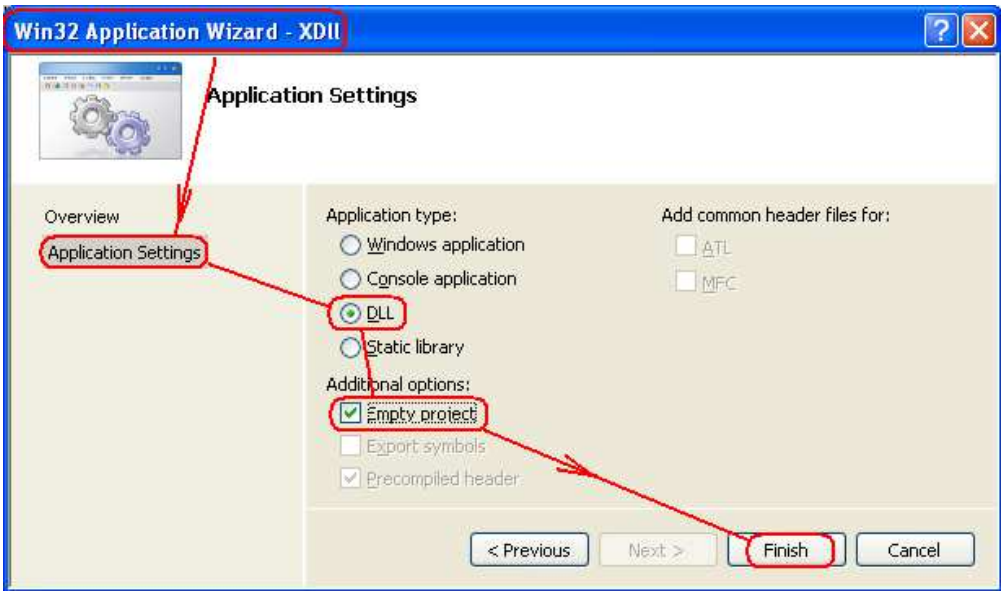
ТЕМА: СОЗДАНИЕ СОБСТВЕННОЙ DLL (Visual C++ 6.0, Visual C++ 9.0)

Для экспортирования функции из DLL - перед ее описанием следует указать ключевое слово __declspec(dllexport), как показано в следующем примере:

1. Создание пустого проекта, например, XDll (примечание: создаваемая DLL получает имя проекта) типа "Win32 Dynamic-Link Library". Для этого необходимо выбрать File -> New -> Project -> Visual C++ -> Win32 -> Win32 Console Application, затем в секции “Application type” выбрать DLL, а в секции “Additional options” отметить чекбокс Empty Project. (добавить картинку)

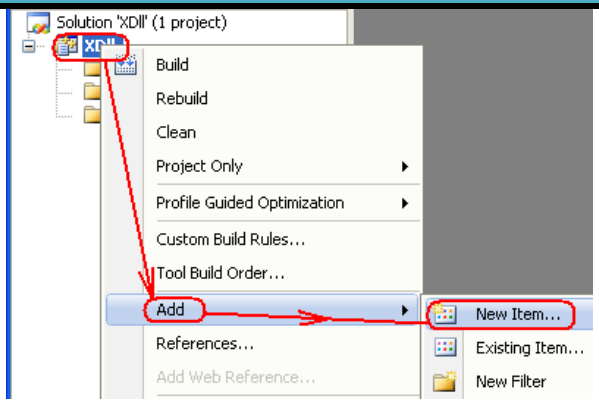


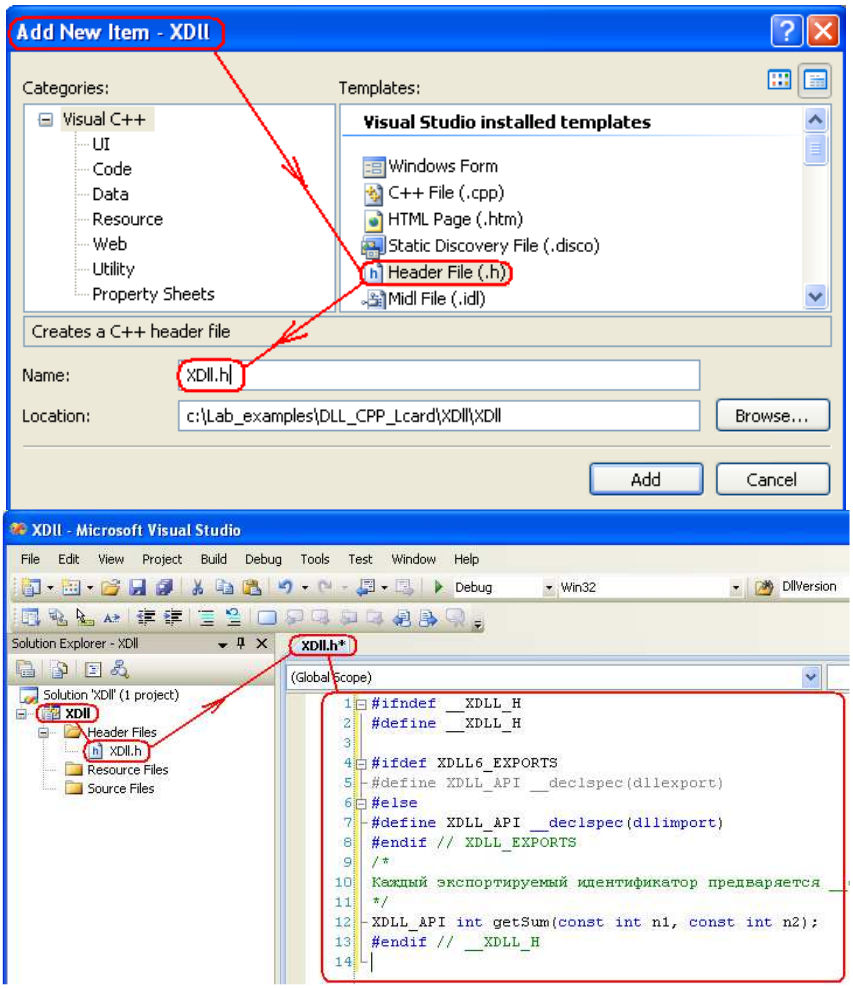




2. Добавление в проект следующих файлов.
Файл XDll.h.

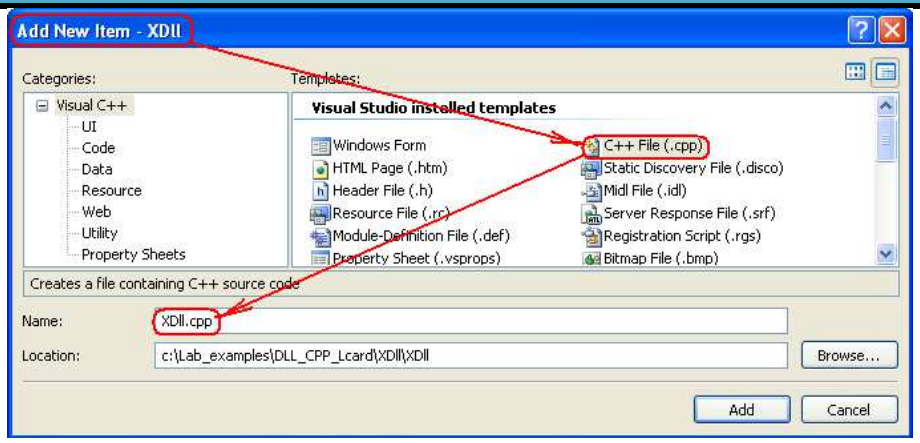
```
#ifndef __XDLL_H
#define __XDLL_H

#ifdef XDLL6_EXPORTS
#define XDLL_API __declspec(dllexport)
#else
#define XDLL_API __declspec(dllimport)
#endif // XDLL_EXPORTS
/*
Каждый экспортируемый идентификатор предваряется __declspec(dllexport). Эта директива позволяет линкеру определить, что данный идентификатор следует экспортировать из DLL. При этом создается специальный lib-файл, который содержит все экспортируемые идентификаторы из модуля. Также экспортируемые объекты заносятся в раздел экспорта DLL - это можно проверить при помощи утилиты dumpbin.exe
*/
XDLL_API int getSum(const int n1, const int n2);
#endif // __XDLL_H
```

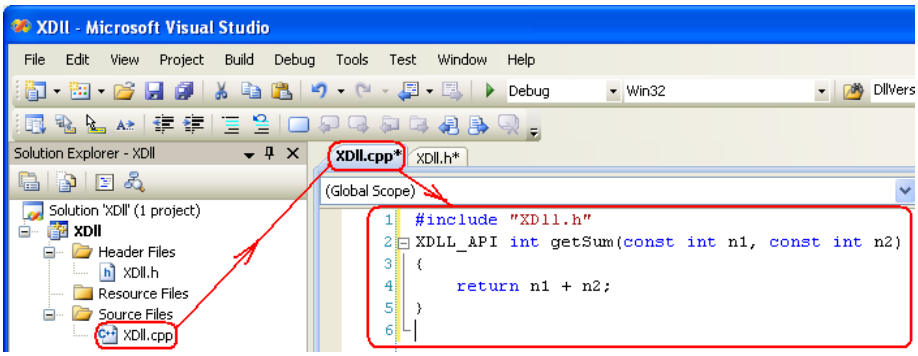




Файл XDll.cpp.

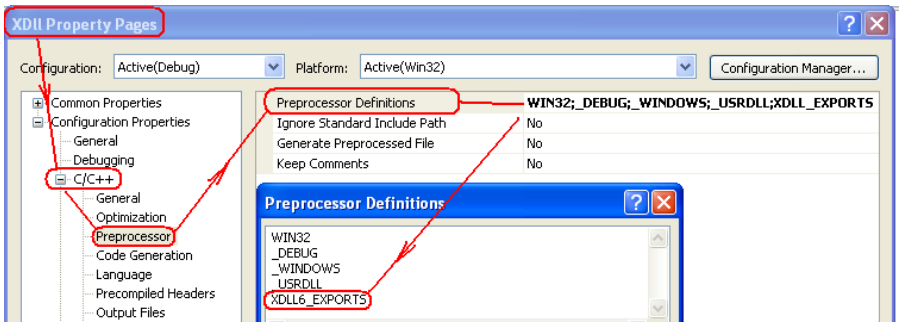
```
#include "XDll.h"
XDLL_API int getSum(const int n1, const int n2)
{
   return n1 + n2;
}
```



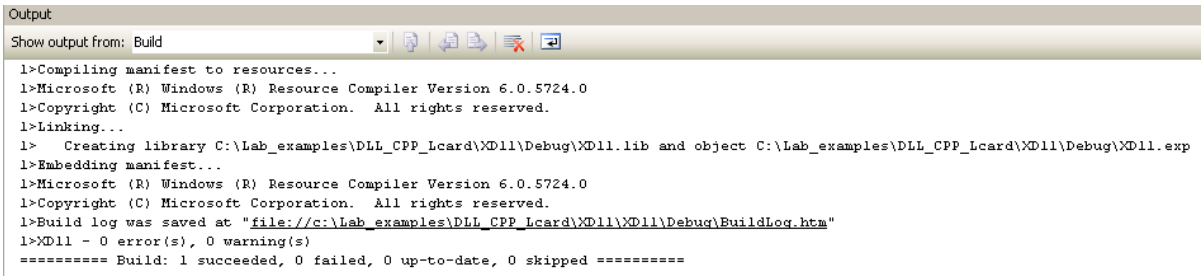


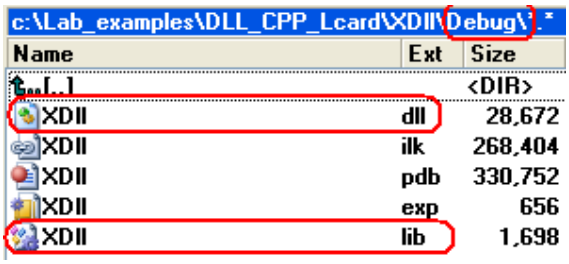
3. Определение идентификатора XDLL6_EXPORTS в настройках проекта (см. "Projects settings->C++->General->Preprocessor definitions".).

   При этом все экспортируемые идентификаторы предваряются символом XDLL_API. Что это дает? В случае определения XDLL6_EXPORTS в проекте XDLL_API определяется как экспортируемый объект; в случае же отсутствия такого определения получаем импортируемый объект.Таким образом, один и тот же заголовочный файл может быть использован и в DLL-проекте, и в проекте, который будет использовать данную DLL!



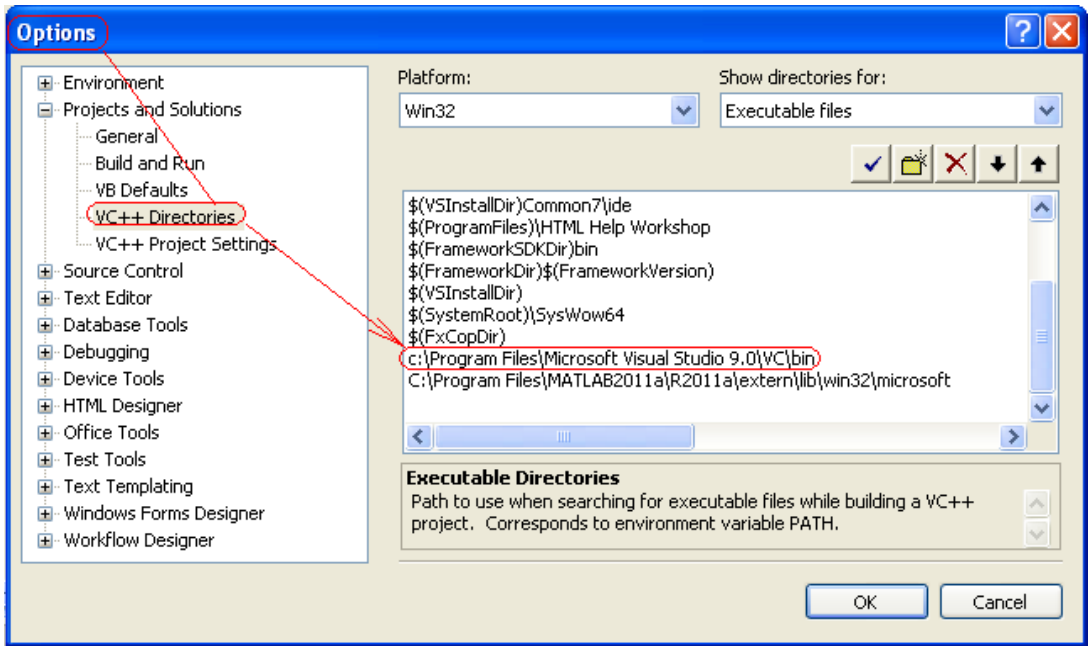
4. Компиляция проекта (Shift+Ctrl+B или F7) файлом ..VC\bin\cl.exe. После успешной компиляции в директории вывода объектных файлов появятся два файла - XDll6.dll и XDll6.lib.



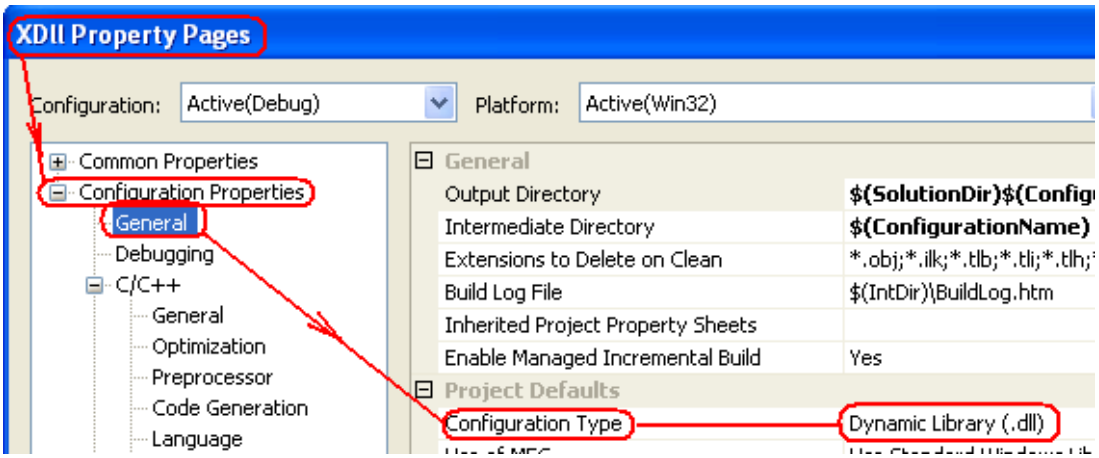


Примечание:

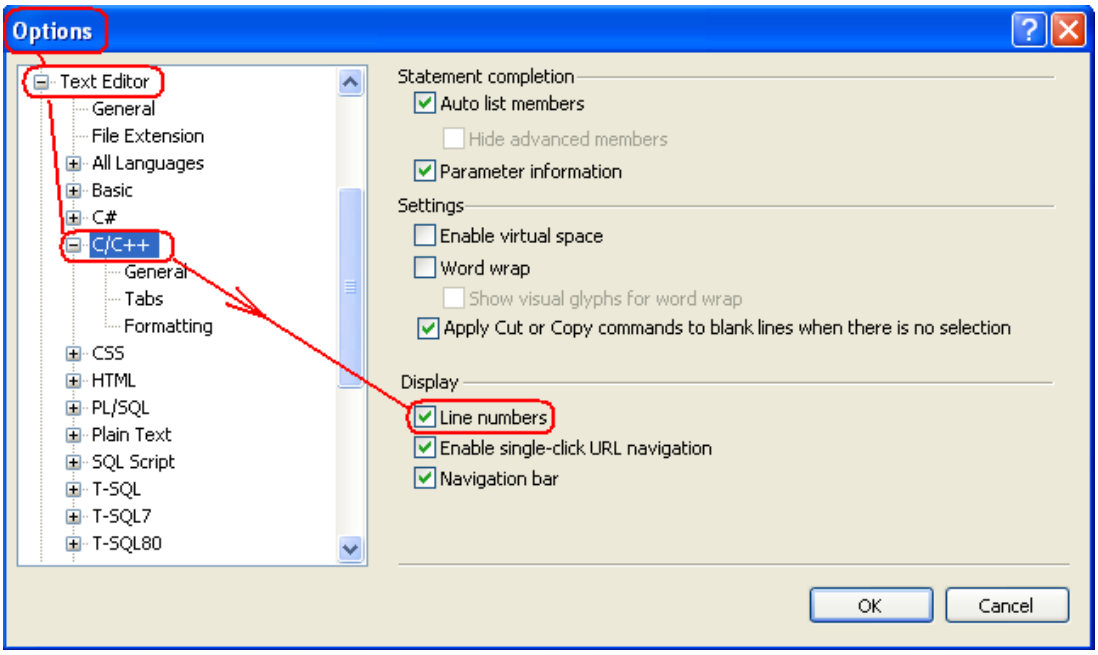
- Настройка среды программирования (Menu > Tools > Options >) на директорию компилятора выполняется как показано на рисунке:



- В список директорий должны быть включены и пути
  $(SystemRoot)
  $(SystemRoot)\System32
  $(SystemRoot)\System32\wbem
- Для исправления ошибки "fatal error C1902: Program database manager mismatch" необходимо удалить mspdb80.dll из ..Program Files\Microsoft Visual Studio 9.0\VC\bin
- Настройка компилятора на создание DLL проверяется в "Configuration Type" как показано на рисунке



- Для отображения номеров строк кода установите следующий параметр в Menu > Tools > Options >



ТЕМА: ОТОБРАЖЕНИЕ СПИСКА ЭКСПОРТИРУЕМЫХ ФУНКЦИЙ DLL

Увидеть список функций экспортируемых из DLL помогает утилита ..\Microsoft Visual Studio 8.0\VC\bin\DUMPBIN.exe с ключом /EXPORTS которая входит в штатную поставку Microsoft Visual Studio.

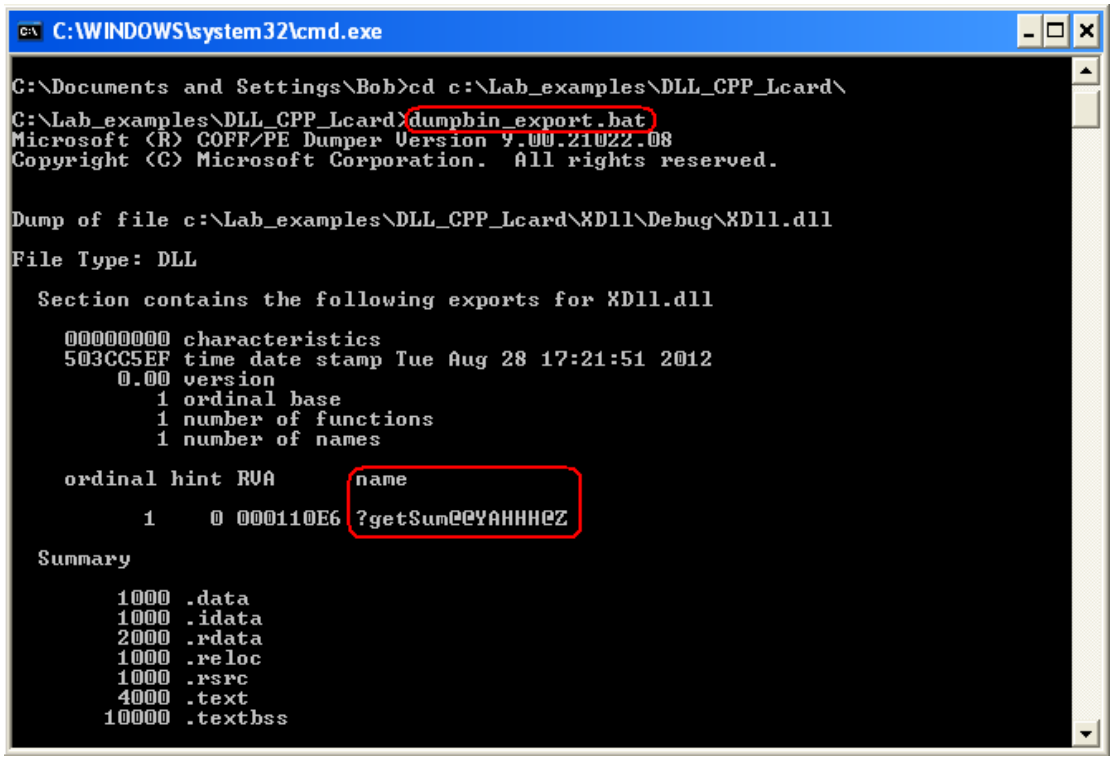
Если для выполнения DUMPBIN требуется mspd80.dll, то ее необходимо найти (скачать, например с http://www.dll-files.com/dllindex/dll-files.shtml?mspdb80) и скопировать в папку с компилятором cl.exe.

ВНИМАНИЕ: для успешной компиляции файл mspd80.dll необходимо убрать из папки компилятора cl.exe.

Пример запуска DUMPBIN.exe через интерпретатор командной строки CMD показан в командном файле и dumpbin_export.bat и последующей картинке.

Файл dumpbin_export.bat(ПРОВЕРИТЬ)

```
@echo off
goto start
-----------------------------------------------
командный файл для просмотра dll функции
-----------------------------------------------
командный файл для просмотра экспортируемых DLL функций через CMD интерпретатор командной строки
имя исходного файла *.dll указывается в последней строке этого файла
-----------------------------------------------
Замечание: комментарий в командном файле использует goto, начинается "rem", или "::"
:start

"c:\Program Files\Microsoft Visual Studio 9.0\VC\bin\"dumpbin.exe /EXPORTS
c:\Lab_examples\DLL_CPP_Lcard\XDll\Debug\XDll.dll
```



Полное имя функции **getSum** со смещением в DLL: **?getSum@@YAHHH@Z**

ТЕМА: НЕЯВНАЯ ЗАГРУЗКА DLL ФАЙЛА

Файл XDll.lib представляет собой список экспортируемых идентификаторов. Наличие такого файла позволяет проводить так называемую неявную загрузку DLL - в этом случае компилятор и линкер по содержимому файла XDll6.lib могут автоматически получить всю информацию, необходимую для правильного разрешения адресов при вызове функции.

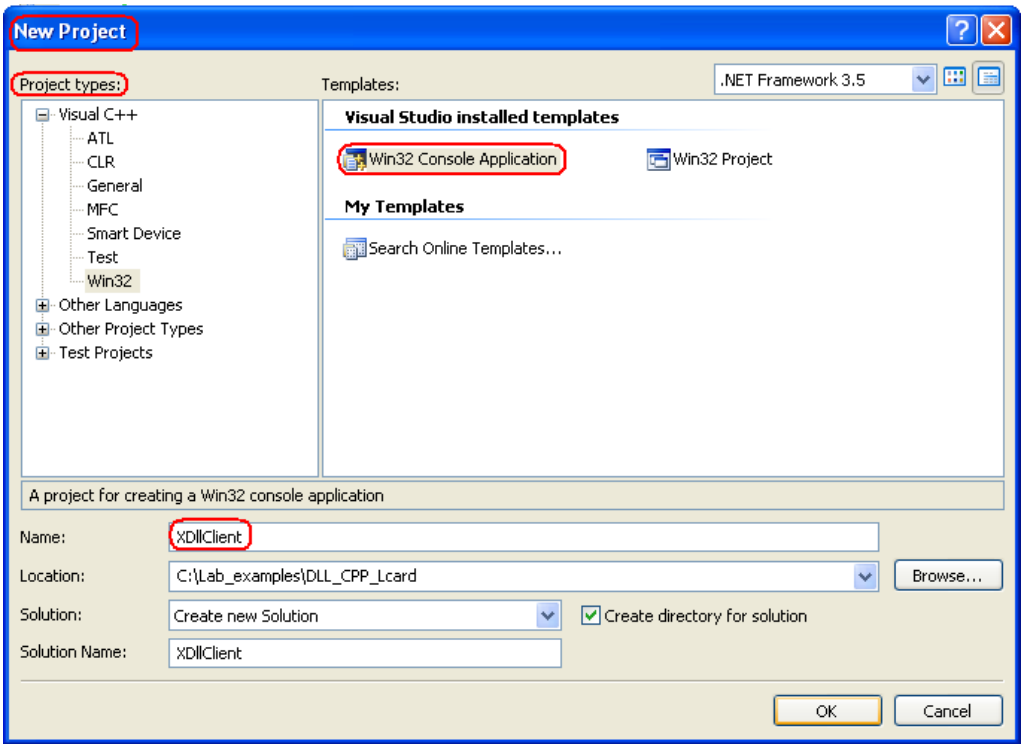
При неявной загрузке DLL проецируется на адресное пространство вызывающего процесса (загружается) при его создании. Если по какой-либо причине неявная загрузка DLL завершается неудачно, загрузчик операционной системы немедленно прерывает процедуру создания процесса, выводит диалоговое окно для уведомления пользователя о возникшей проблеме и "прибивает" процесс.

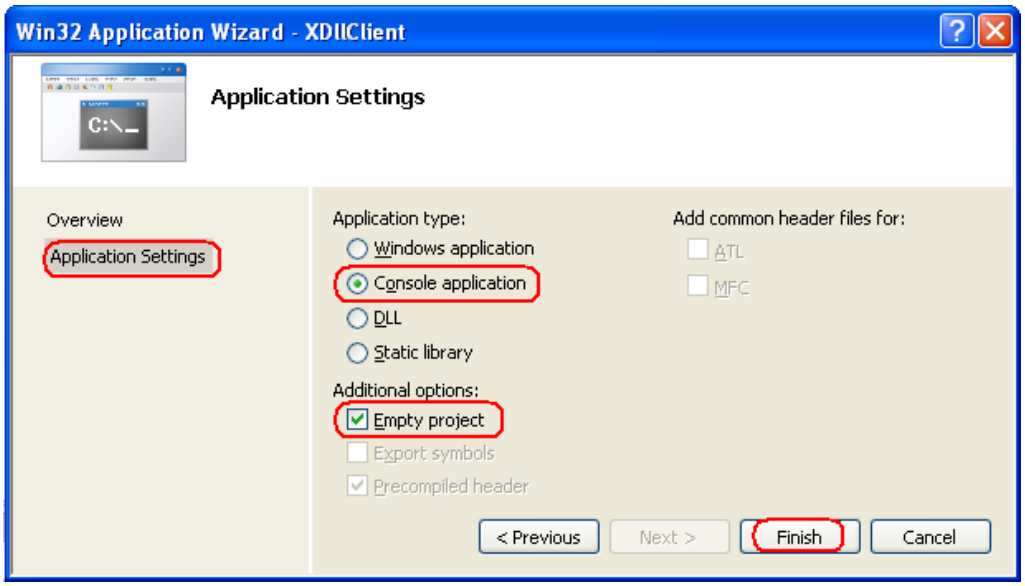
В случае неявной загрузки приложению требуются:

- h-файл (header - заголовочный) с прототипами функций, описаниями классов и типов, которые используются в приложении;
- lib-файл (library - библиотечный), в котором описывается список экспортируемых из DLL функций (переменных), и их смещения, необходимые для правильной настройки вызовов функций.

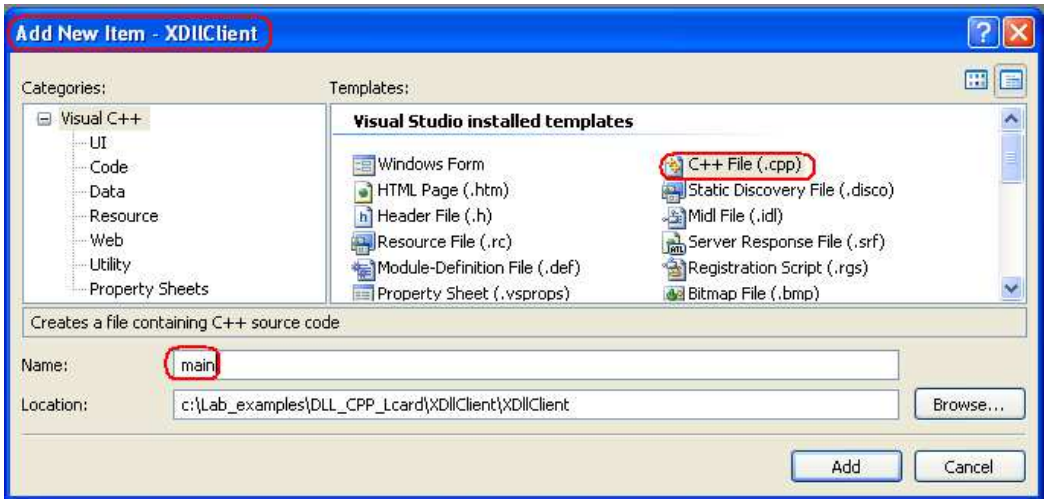
Файл процесса для неявной загрузки DLL создается в следующей последовательности.

1. Создание нового проекта XdllClient. Для этого необходимо выбрать File -> New -> Project -> Visual C++ -> Win32 -> Win32 Console Application, затем в секции “Application type” выбрать consol application, а в секции “Additional options” отметить чекбокс Empty Project





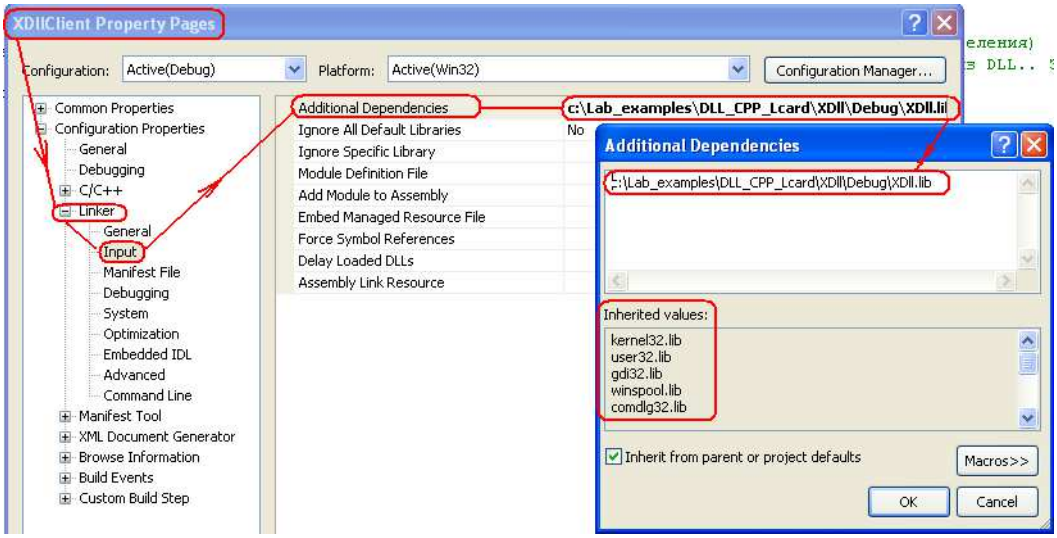
2. Создание и добавление в проект файла main.cpp. Текст его приведен ниже.

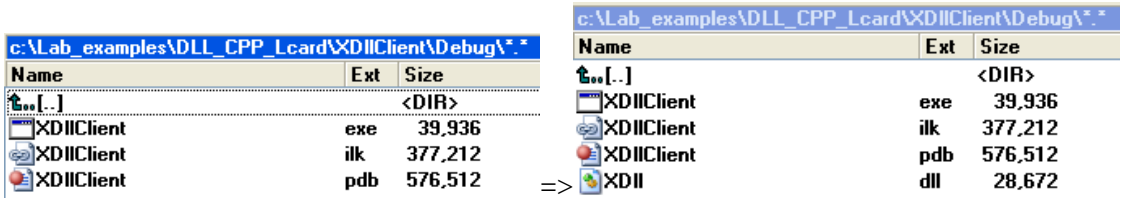


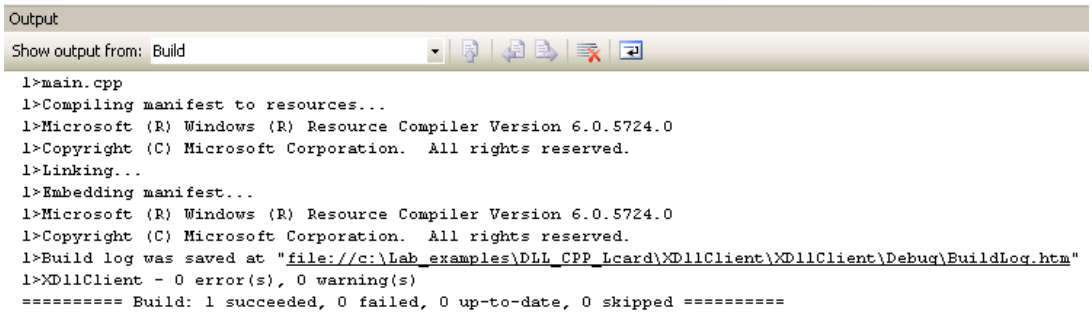
main.cpp

```
#include "../../XDLL/XDLL/XDll.h"
/* Этот файл (XDll.h) необходим, для того чтобы компилятор мог:
а) проверить корректность синтаксиса вызова функции (на основе ее определения)
б) узнать, что данные функции (в нашем случае - getSum) ИМПОРТИРУЮТСЯ из DLL.. Это  осуществляется при помощи различного определения символа XDLL_API (в случае включения его в файлы проекта DLL он определяется как __declspec(dllexport); если же мы включаем его в файл приложения, то он определяется как __declspec(dllimport) - это регулируется наличием/отсутствием идентификатора XDLL_EXPORTS в настройках проекта DLL - см. "ProjectsSettings->C++->General->Preprocessor definitions"; в клиентском приложении этот идентификатор не должен определяться!)
*/

#include <iostream> // std::cout is in the library

void main()
{
    /* используем функцию так, словно мы сами ее написали */
    const int res = getSum(10, 20);

    std::cout << "getSum(10, 20): " << res << std::endl;
}
```

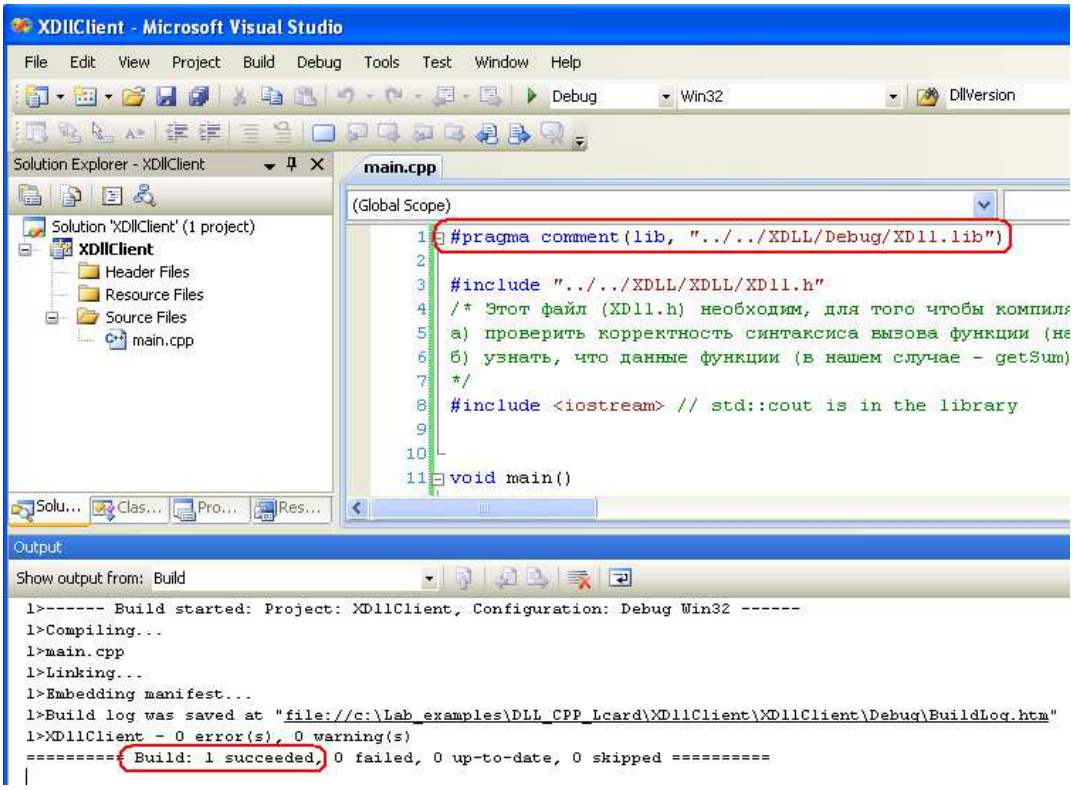
3. Компиляция проекта. Укажите путь к библиотеке импорта XDll.lib:



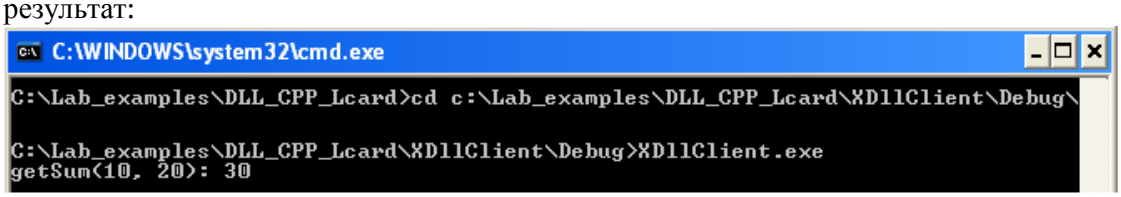




ЗАМЕЧАНИЕ:Альтернативный вариант подключения библиотеки импорта заключается в использовании директивы #pragma. В этом случае необходимо добавить примерно такой текст директивы в cpp-файл: #pragma comment(lib, "xdll.lib")



4. Запуск загрузочного файла на выполнение. Файл XDllClient.exe выдает следующий результат:



ТЕМА: ЯВНАЯ ЗАГРУЗКА DLL ФАЙЛА

Этот способ связан с явным использованием основных функций Windows API. Ниже перечислен наиболее часто используемый набор предоставляемого WinApi для работы с DLL явной загрузки:

- DisableThreadLibraryCalls - функция, "запрещающая" получать DLL уведомления DLL_THREAD_ATTACH и DLL_THREAD_DETACH Это бывает полезно в многопоточных приложениях, когда постоянно создаются и уничтожаются рабочие потоки, а DLL не требует получения подобных уведомлений. В целях оптимизации исполняемого кода обычно вызывается в ответ на сообщение DLL_PROCESS_ATTACH.
- FreeLibrary - функция, используемая для явной выгрузки DLL из ОП. Используется для DLL, которая была перед этим загружена при помощи вызова LoadLibrary.
- FreeLibraryAndExitThread - функция, позволяющая потоку, созданному в коде DLL, быть безопасно уничтоженным (с последующей выгрузкой DLL).
- GetModuleFileName[Ex] - позволяют получить полный путь к конкретному модулю с идентификатором HMODULE.

- GetModuleHandle[Ex] - позволяют получить идентификатор HMODULE по имени модуля. Функция возвращает корректное значение HMODULE только для тех модулей, которые были спроецированы на адресное пространство вызывающего процесса.
- GetProcAddress - функция, позволяющая получить виртуальный адрес экспортируемой из DLL функции (или переменной) для ее последующего вызова.
- LoadLibrary[Ex] - позволяют спроецировать DLL на адресное пространство вызывающего процесса.

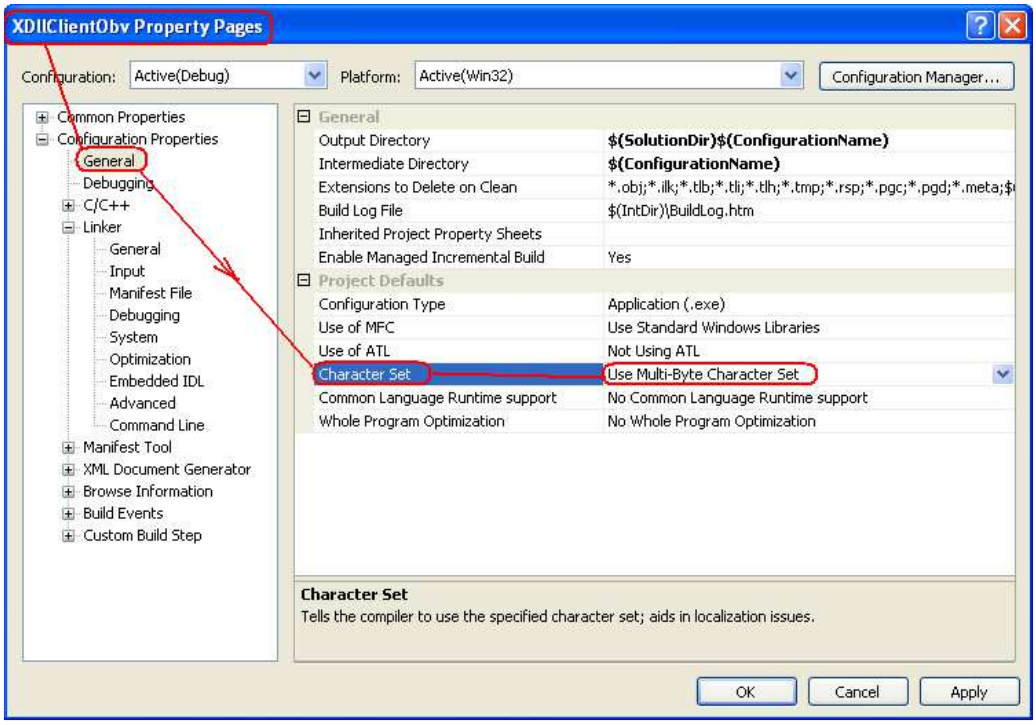
Основная нагрузка в этом случае ложится на функции LoadLibrary, LoadLibraryEx, FreeLibrary и GetProcAddress.

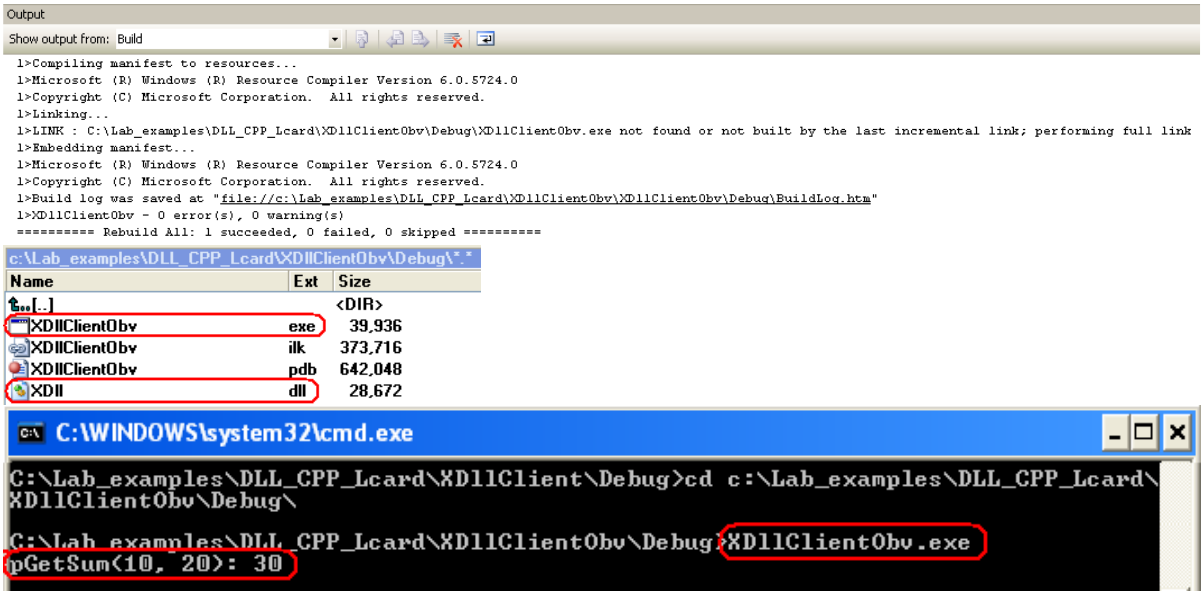
Рассмотрим назначение основных функций LoadLibrary и LoadLibraryEx
Теперь файл main.cpp должен иметь примерно следующий вид:

Файл: main.cpp

```
/*
Теперь файл (XDll.h) не требуется - нам достаточно знать сигнатуру функции для последующего ее правильного определения при помощи typedef. Ведь если неправильно определить указатель на функцию, то последующий вызов по этому указателю приведет к краху приложения в связи с нарушением доступа.
*/
//#include "../XDll6/XDll.h"
#include <iostream>
#include <windows.h>

int main()
{
    /* явным образом проецируем DLL на адресное пространство нашего процесса */
    HMODULE hModule = LoadLibrary("XDll.dll");
      /* функция выше дает ошибку если не изменить установки проекта
      “по умолчанию” с “unicode character set” на “multi-byte, */

    /* проверяем успешность загрузки */
    _ASSERT(hModule != NULL);

    /*
     определяем при помощи typedef новый тип - указатель на вызываемую функцию.
      Очень важно знать типы и количество аргументов, а также тип возвращаемого
      результата
       */

    typedef int (*PGetSum)(const int, const int);
    /*  Первый вариант: пытаемся получить адрес функции getSum по имени, полученному утилитой dumpbin.exe*/
//    PGetSum pGetSum = (PGetSum)GetProcAddress(hModule, "?getSum@@YAHHH@Z");

     /* Второй вариант: пытаемся получить адрес функции getSum. Так делать ни в коем случае не рекомендуется - изменение порядкового номера (например, в случае добавления в DLL новых экспортируемых идентификаторов) приведет к немедленному краху приложения. */
     PGetSum pGetSum = (PGetSum)GetProcAddress(hModule, MAKEINTRESOURCE(1));

    /* проверяем успешность получения адреса */
    _ASSERT(pGetSum != NULL);

    /* используем функцию так, словно мы сами ее написали */
    const int res = pGetSum(10, 20);

    std::cout << "pGetSum(10, 20): " << res << std::endl;

    /* выгружаем библиотеку из памяти */
```

```
    BOOL b = FreeLibrary(hModule);
    /* проверяем корректность выгрузки */
    _ASSERT(b);

    return 0;
}
```





Замечание: Количество вызовов функции FreeLibrary должно точно соответствовать количеству вызовов LoadLibrary!

ТЕМА: ОТОБРАЖЕНИЕ СПИСКА ИМПОРТИРУЕМЫХ ФУНКЦИЙ DLL

Увидеть список DLL функций импортируемых процессом можно при помощи утилиты ..\Microsoft Visual Studio 8.0\VC\bin\DUMPBIN.exe с ключом /IMPORTS которая входит в штатную поставку Microsoft Visual Studio.

Пример запуска DUMPBIN.exe для импортирования DLL функций через интерпретатор командной строки CMD показан в командном файле и dumpbin_import.bat и в последующей картинке.

Пример

dumpbin_import.bat (ПРОВЕРИТЬ)

```
@echo off
"c:\Program Files\Microsoft Visual Studio 9.0\VC\bin\"dumpbin.exe /IMPORTS
c:\Lab_examples\DLL_CPP_Lcard\XDllClientObv\Debug\XDllClientObv.exe
```

ТЕМА: ПОДКЛЮЧЕНИЕ МОДУЛЯ E14-440 К КОМПЬЮТЕРУ

- При самом *первом* подсоединении модуля *E14-440* к компьютеру операционная система должна запросить файлы драйвера. Тогда ей необходимо указать *inf*-файл от библиотеки *Lusbapi* ..\DRV\Lusbapi.inf. При этом операционная система сама скопирует все требуемые ей файлы в нужные места и сделает все необходимые записи в своём реестре.

- Для обеспечения надлежащей работы Ваших приложений с модулем *E14-440* поставщик модуля рекомендует скопировать бинарный файл библиотеки \DLL\Bin\Lusbapi.dll в директорию %SystemRoot%\system32 (хотя Microsoft настоятельно рекомендует хранить все используемые DLL в рабочем каталоге программы и лишь в случае острой необходимости пользоваться системными директориями).

Примечание: Пример запуска *.exe файла в MatLAB: >> open('XDllClient.exe')

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

Для выполнения заданий

- подключите модуль E14-440 к компьютеру
- установите драйвер модуля
- проверьте работоспособность модуля используя программу LGraph2

Пример рабочего кода программы для чтения АЦП и отображения данных **ReadData.cpp**:

```
//****************************************************************************
***
// Модуль E14-440.
// Консольная программа с организацией ввода данных с АЦП
// Ввод осуществляется с первых четырёх каналов АЦП на частоте 100 кГц.
// Данные двух каналов отображаются на мониторе компьютера
//****************************************************************************
***
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include "Lusbapi.h" // заголовочный файл библиотеки Lusbapi

#define CHANNELS_QUANTITY               (0x4)

// аварийный выход из программы
void AbortProgram(char *ErrorString, bool AbortionFlag = true);

DWORD DllVersion;    // версия библиотеки
ILE440 *pModule;     // указатель на интерфейс модуля
MODULE_DESCRIPTION_E440 md; // структура с информацией о модуле
HANDLE ModuleHandle; // дескриптор устройства
char ModuleName[7];  // название модуля
BYTE UsbSpeed;       // скорость работы шины USB
MODULE_DESCRIPTION_E440 ModuleDescription; // структура с полной информацией
о модуле
ADC_PARS_E440 ap;    // структура параметров работы АЦП модуля

bool IsReadThreadComplete;  // флажок завершения работы потока сбора данных
WORD ReadThreadErrorNumber; // номер ошибки при выполнении сбора данных

// отсчёты АЦП
SHORT AdcSample1, AdcSample2;
// индекс входного диапазона напряжения
const WORD InputRangeIndex = ADC_INPUT_RANGE_2500mV_E440;

int main(void)
{
```

```
      WORD i;

      // проверим версию используемой библиотеки Lusbapi.dll
      if((DllVersion = GetDllVersion()) != CURRENT_VERSION_LUSBAPI)
      {
            char String[128];
            printf(String,  "  Lusbapi.dll  Version  Error!!!\n      Current:
%1u.%1u. Required: %1u.%1u",
                  DllVersion >> 0x10, DllVersion & 0xFFFF,
                  CURRENT_VERSION_LUSBAPI  >>  0x10,  CURRENT_VERSION_LUSBAPI  &
0xFFFF);

             return 1;
      }
      else printf(" Lusbapi.dll Version --> OK\n");

      // получим указатель на интерфейс модуля
      pModule = static_cast<ILE440 *>(CreateLInstance("e440"));
      if(!pModule) {printf(" Module Interface --> Bad\n"); return 1;}
      else printf(" Module Interface --> OK\n");

      //     попробуем     обнаружить     модуль     E14-440     в     первых
MAX_VIRTUAL_SLOTS_QUANTITY_LUSBAPI виртуальных слотах
      for(i  =  0x0;  i  <  MAX_VIRTUAL_SLOTS_QUANTITY_LUSBAPI;  i++)  if(pModule-
>OpenLDevice(i)) break;
      // что-нибудь обнаружили?
      if(i  ==  MAX_VIRTUAL_SLOTS_QUANTITY_LUSBAPI)  return  1;  //AbortProgram("
Can't find any module E14-440 in first 127 virtual slots!\n");
      else printf(" OpenLDevice(%u) --> OK\n", i);

      // попробуем прочитать дескриптор устройства
      ModuleHandle = pModule->GetModuleHandle();
      if(ModuleHandle == INVALID_HANDLE_VALUE)
      {printf(" GetModuleHandle() --> Bad\n"); return 1;}
      else printf(" GetModuleHandle() --> OK\n");

      // прочитаем название модуля в обнаруженном виртуальном слоте
      if(!pModule->GetModuleName(ModuleName))
            {printf(" GetModuleName() --> Bad\n"); return 1;}
      else printf(" GetModuleName() --> OK\n");

      // проверим, что это 'E14-440'
      if(strcmp(ModuleName, "E440"))
            {printf(" The module is not 'E14-440'\n"); return 1;}
      else printf(" The module is 'E14-440'\n");

      // попробуем получить скорость работы шины USB
      if(!pModule->GetUsbSpeed(&UsbSpeed))    {printf("    GetUsbSpeed()    -->
Bad\n"); return 1;}
      else printf(" GetUsbSpeed() --> OK\n");
      // теперь отобразим скорость работы шины USB
      printf("    USB is in %s\n", UsbSpeed ? "High-Speed Mode (480 Mbit/s)" :
"Full-Speed Mode (12 Mbit/s)");

      // теперь можно попробовать загрузить из соответствующего ресурса
      // библиотеки Lusbapi код драйвера LBIOS
      if(!pModule->LOAD_MODULE())
            {printf(" LOAD_MODULE() --> Bad\n"); return 1;}
      else printf(" LOAD_MODULE() --> OK\n");

      // проверим загрузку модуля
      if(!pModule->TEST_MODULE())
```

```
            {printf(" TEST_MODULE() --> Bad\n"); return 1;}
      else printf(" TEST_MODULE() --> OK\n");

      // получим информацию из ППЗУ модуля
      if(!pModule->GET_MODULE_DESCRIPTION(&ModuleDescription))
            {printf(" GET_MODULE_DESCRIPTION() --> Bad\n"); return 1;}
      else    printf("    E14-440    (s/n    %s)    is    READY    TO    WORK\n",
md.Module.SerialNumber);

      ////////////////////////////////////////////////////////////////////////
///
      // далее можно располагать функции для непосредственного
      // управления модулем
      ////////////////////////////////////////////////////////////////////////
///

      // получим текущие параметры работы АЦП
      if(!pModule->GET_ADC_PARS(&ap))  {printf("  GET_ADC_PARS()  -->  Bad\n");
return 1;}
      else printf(" GET_ADC_PARS() --> OK\n");

      // установим желаемые параметры работы АЦП
      ap.IsCorrectionEnabled = true;                  //               разрешим
корректировку данных на уровне драйвера DSP

      ap.InputMode = NO_SYNC_E440;              //  обычный  сбор  данных  безо
всякой синхронизации ввода
      //ap.InputMode = TTL_START_SYNC_E440;           //               цифровая
синхронизация начала ввода на входе TRIG аналогового разъёма
      //ap.InputMode = TTL_KADR_SYNC_E440;            //  цифровая  покадровая
синхронизация ввода.
      //ap.InputMode = ANALOG_SYNC_E440;        //   аналоговая   синхронизация
начала ввода.

      ap.ChannelsQuantity = CHANNELS_QUANTITY;        //     четыре     активных
канала

      // формируем управляющую таблицу
      for(i = 0x0; i < ap.ChannelsQuantity; i++)
            ap.ControlTable[i] = (WORD)(i | (InputRangeIndex << 0x6));

      ap.AdcRate = 100.0;                                             //    частота
работы АЦП в кГц (max 400.0)
      ap.InterKadrDelay = 0.0;                                  //      межкадровая
задержка в мс
      ap.AdcFifoBaseAddress = 0x0;                    //  базовый  адрес  FIFO
буфера АЦП в DSP модуля
      ap.AdcFifoLength = MAX_ADC_FIFO_SIZE_E440;      // длина FIFO буфера АЦП
в DSP модуля

      //  будем  использовать  фирменные  калибровочные  коэффициенты,  которые
хранятся в ППЗУ модуля
      for(i = 0x0; i < ADC_CALIBR_COEFS_QUANTITY_E440; i++)
      {
            ap.AdcOffsetCoefs[i]                                                =
ModuleDescription.Adc.OffsetCalibration[i];
            ap.AdcScaleCoefs[i] =  ModuleDescription.Adc.ScaleCalibration[i];
      }

      // передадим требуемые параметры работы АЦП в модуль
      if(!pModule->SET_ADC_PARS(&ap))  {printf("  SET_ADC_PARS()  -->  Bad\n");
return 1;}
      else printf(" SET_ADC_PARS() --> OK\n");
```

```
      // отобразим параметры сбора данных модуля на экране монитора
      printf(" \n");
      printf("    Module    E14-440    (S/N    %s)    is    ready    ...    \n",
ModuleDescription.Module.SerialNumber);
      printf("   Module Info:\n");
      printf("                  Module       Revision           is    '%c'\n",
ModuleDescription.Module.Revision);
      printf("                 MCU    Driver    Version    is    %s    (%s)\n",
ModuleDescription.Mcu.Version.Version, ModuleDescription.Mcu.Version.Date);
      printf("              LBIOS        Version           is   %s   (%s)\n",
ModuleDescription.Dsp.Version.Version, ModuleDescription.Dsp.Version.Date);
      printf("   Adc parameters:\n");
      printf("         Data  Correction  is  %s\n",  ap.IsCorrectionEnabled  ?
"enabled" : "disabled");
      printf("     ChannelsQuantity = %2d\n", ap.ChannelsQuantity);
      printf("     AdcRate = %8.3f kHz\n", ap.AdcRate);
      printf("     InterKadrDelay = %2.4f ms\n", ap.InterKadrDelay);
      printf("     KadrRate = %8.3f kHz\n", ap.KadrRate);
      printf("                     Input    Range       =    %6.2f    Volt\n",
ADC_INPUT_RANGES_E440[InputRangeIndex]);

      printf("\n Press any key if you want to terminate this program...\n");
      // цикл перманентного выполнения функции ADC_SAMPLE и
      // отображения полученных данных на экране дисплея
      printf("\n\n");
      printf(" ADC Channel:                    1         2\n");
      while(!_kbhit())
      {
            if(!pModule->ADC_SAMPLE(&AdcSample1,         (WORD)(0x00         |
(InputRangeIndex << 6)))) { printf("\n\n  ADC_SAMPLE(, 0) --> Bad\n"); break;
}
            else     if(!pModule->ADC_SAMPLE(&AdcSample2,     (WORD)(0x01     |
(InputRangeIndex << 6)))) { printf("\n\n  ADC_SAMPLE(, 1) --> Bad\n"); break;
}

            printf("  AdcSample  Data  (ADC  code):         %5d         %5d\r",
AdcSample1, AdcSample2);
      }

      // освободим интерфейс модуля
      printf("\n\n");
      AbortProgram(" The program was completed successfully!!!\n", false);

      ////////////////////////////////////////////////////////////////////////
///
      // завершим работу с модулем, освободим интерфейс модуля
      // AbortProgram is used instead of
      ////////////////////////////////////////////////////////////////////////
///
      if(!pModule->ReleaseLInstance())
      {
            printf(" ReleaseLInstance() --> Bad\n");
            return 1;
      }
      else
      {
            printf(" ReleaseLInstance() --> OK\n");
      //    обнулим указатель на интерфейс модуля
            pModule = NULL;
            return 0;
      }
}
```

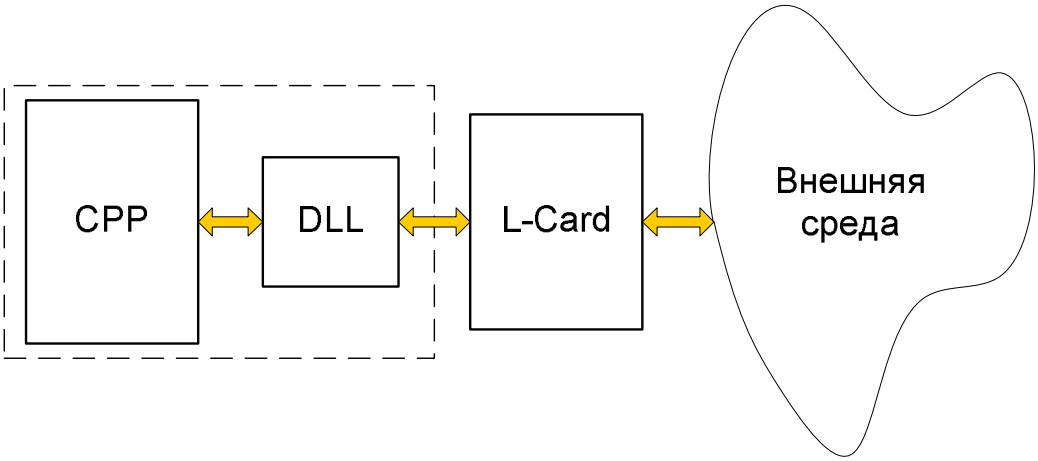
```
//---------------------------------------------------------------------------
// аварийное завершение программы
//---------------------------------------------------------------------------
void AbortProgram(char *ErrorString, bool AbortionFlag)
{
      // подчищаем интерфейс модуля
      if(pModule)
      {
            // освободим интерфейс модуля
            if(!pModule->ReleaseLInstance())  printf("  ReleaseLInstance()  -->
Bad\n");
            else printf(" ReleaseLInstance() --> OK\n");
            // обнулим указатель на интерфейс модуля
            pModule = NULL;
      }

      // выводим текст сообщения
      if(ErrorString) printf(ErrorString);

      // прочистим очередь клавиатуры
      if(_kbhit()) { while(_kbhit()) getch(); }

      // если нужно - аварийно завершаем программу
      if(AbortionFlag) exit(0x1);
      // или спокойно выходим из функции
      else return;
}
```

Перевод файла **ReadData.cpp** (показанного выше) в исполняемый файл **ReadData.exe** можно выполнить следующим командным файлом.

Файл **dll_builder.bat**:

```
@echo off

goto start
-----------------------------------------------
командный файл для компиляции dll и lib файлов
исходный файл *.C вводится первым параметром
-----------------------------------------------
Замечание: комментарий в командном файле
использует goto, или начинается "rem", или "::"

:start

Set PATH=c:\Program Files\Microsoft Visual Studio 9.0\VC\bin\;%PATH%
Set INCLUDE=c:\Program Files\Microsoft Visual Studio 9.0\VC\include\;%INCLUDE%
Set LIB=c:\Program Files\Microsoft Visual Studio 9.0\VC\lib\;%LIB%

rem для компиляции необходима библиотека "Kernel32.Lib" которая находится
:: в пакете SDK

Set LIB=c:\Program Files\Microsoft SDKs\Windows\v6.0A\Lib\;%LIB%
"c:\Program Files\Microsoft Visual Studio 9.0\VC\bin\"cl.exe ReadData.cpp /LD
```

Задание 1. Построение библиотеки функций для чтения АЦП модуля E14-440.



Примечание. Для выполнения задания используйте ТЕМУ: "СОЗДАНИЕ СОБСТВЕННОЙ DLL (Visual C++ 6.0, Visual C++ 8.0)"

1. В Microsoft Visual C++ создайте пустой проект, например, LcardDll типа "Win32 Dynamic-Link Library".
2. Добавьте в проект LcardDll.cpp файл представленный выше.
3. Разбейте main модуль CPP файла на три экспортируемых функции в соответствии с данными таблицы

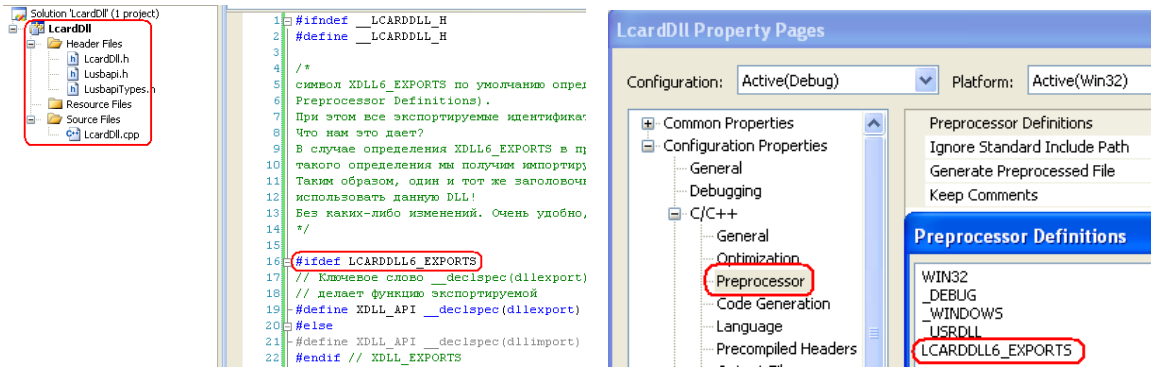
Экспортируемая функция	Назначение
XDLL_API int Connect(int stage)	Установка связи с модулем, настройка режимов работы модуля
XDLL_API void ReadCard(void)	Чтение АЦП и отображение данных
XDLL_API int Disconnect(void)	Окончание сеанса связи с модулем

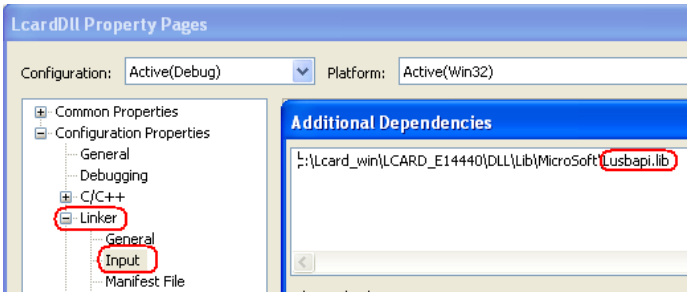
4. Доработайте CPP файл в соответствии с правилами примера построения XDll.cpp
5. Добавьте в проект новый пустой файл LcardDll.h
6. Доработайте h файл в соответствии с правилами примера построения XDll.h
7. Определите идентификатора XDLL6_EXPORTS в настройках проекта (см. "Projects settings->C++->General->Preprocessor definitions".
8. Добавьте в проект файлы Lusbapi.h и LusbapiTypes.h
9. Подключите к проекту Lusbapi.lib (для среды Microsoft). Размер файла 2178 байт.
10. Откомпилируйте проект. Получите DLL библиотеку проекта.
11. Программой DUMPBIN.exe через интерпретатор командной строки CMD получите список имен внешних функций DLL библиотеки (см. **ТЕМА:** ОТОБРАЖЕНИЕ СПИСКА ЭКСПОРТИРУЕМЫХ ФУНКЦИЙ DLL)

Информация для контроля:

Файл: LcardDll.h

```
#ifndef __LcardDll_H
#define __LcardDll_H

#ifdef LCARDDLL6_EXPORTS
#define XDLL_API __declspec(dllexport)
#else
#define XDLL_API __declspec(dllimport)
#endif // XDLL_EXPORTS


XDLL_API int Connect(int stage);

XDLL_API void ReadCard(void);

XDLL_API int Disconnect(void);

#endif // __XDLL_H
```





Задание 2. Чтение АЦП модуля E14-440 при неявной загрузки DLL функций.

Для выполнения задания используйте пример **ТЕМЫ:** НЕЯВНАЯ ЗАГРУЗКА DLL ФАЙЛА.

1. Создайте новый пустой проект LcardDllClient для Win32 Console Application.
2. Создайте и добавьте в проект файла main.cpp. Текст файла приведен ниже.

```
#include "../../LcardDll/LcardDll/LcardDll.h"
#include <iostream> // std::cout is in the library
```

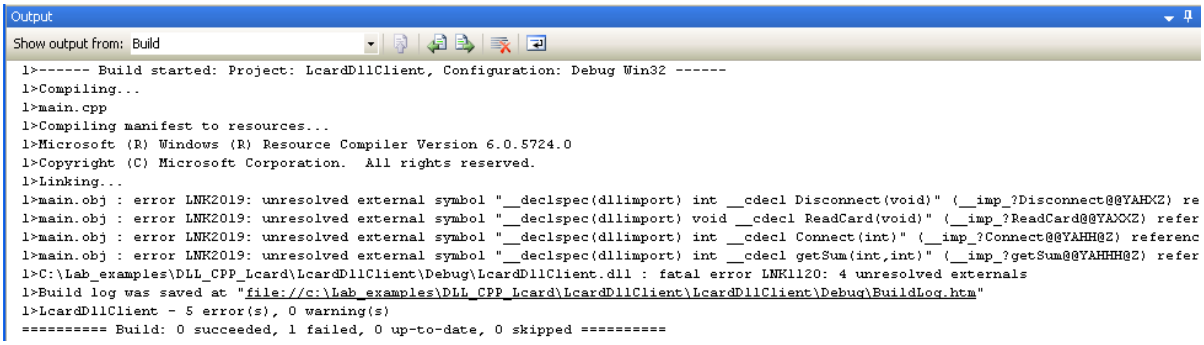
```
void main()
{
    /* используем функцию так, словно мы сами ее написали */
    const int res = getSum(10, 20);

    std::cout << "getSum(10, 20): " << res << std::endl;

      int return_value;
      int stage;

      stage = 123; // величина stage функцией Connect не используется
      return_value = Connect (stage); // соединение с модулем E14-440
      ReadCard();  // чтение и отображение данных АЦП
      return_value = Disconnect (); // отключение модуля
}
```

3. Откомпилируйте проект. Укажите путь к библиотеке импорта LcardDLL.lib
4. Запустите загрузочный файл на выполнение. Проверьте работоспособность связи с модулем.

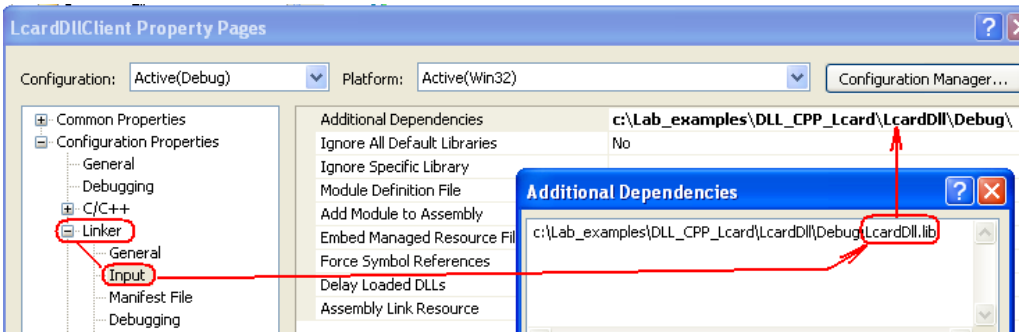
Информация для контроля:

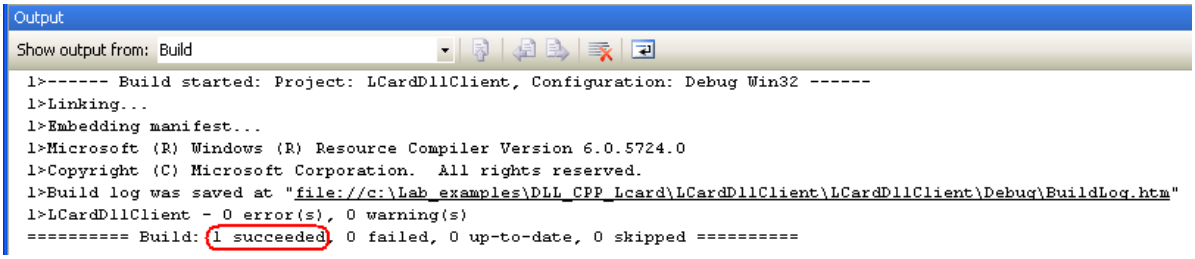
Файл **main.cpp**:

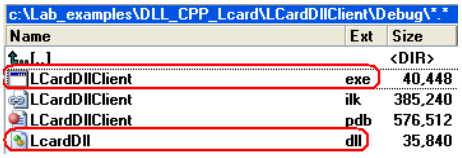
```
#include "../../LcardDll/LcardDll/LcardDll.h"
#include <iostream> // std::cout is in the library

void main()
{
      int return_value;
      int stage;

      stage = 123; // величина stage функцией Connect не используется
      return_value = Connect (stage); // соединение с модулем E14-440
      ReadCard();  // чтение и отображение данных АЦП
      return_value = Disconnect (); // отключение модуля
}
```

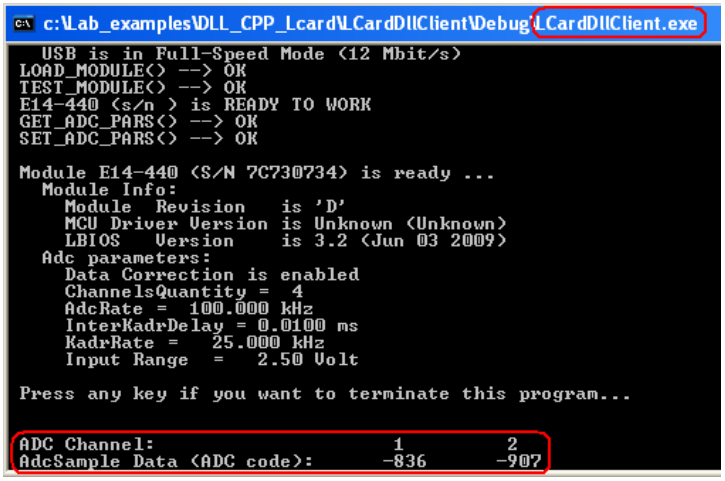


Выше показанный результат компиляции “1failed” означает, что для использования оригинальных (без смещения) имен функций библиотеки необходимо подключить lib файл к проекту:









КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что будет, если все-таки не вызывать FreeLibrary при явной загрузке DLL?
   Ответ: Ничего плохого не случится. Система при завершении приложения сама освободит все используемые ресурсы (в т.ч. выгрузит из памяти все неиспользуемые DLL - конечно, в том случае, если счетчик ссылок на DLL со стороны внешних пользователей достигнет своего нулевого значения).
2. Что будет, если вызвать LoadLibrary два и более раз при явной загрузке?
   Ответ: В этом случае счетчик ссылок увеличится на такое же число. Для завершения (выгрузки) DLL потребуется аналогичное число раз вызвать FreeLibrary.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. DLL - динамически загружаемые библиотеки (Часть 1). Как создать собственную DLL http://4synapse.com/ru/article/dll---dinamicheski-zagrujaemyie-biblioteki-chast-1_5.html
2. DLL - динамически загружаемые библиотеки (Часть 2) http://4synapse.com/ru/article/dll---dinamicheski-zagrujaemyie-biblioteki-chast-2_6.html
3. Dr. Bob Davidov. Компьютерные технологии управления в технических системах http://portalnp.ru/author/bobdavidov.