## Шаг 297 - Венгерская нотация и MFC

В **MFC** используется венгерская нотация. То есть правила написания имен переменных. Смысл этих правил в том, чтобы по имени переменной можно было определить ее тип. Например, **chName** - 8 битная строка. Почему называется она венгерской ??? Существует такая история, что был в **Microsoft** такой программист из венгрии по имени Чарльз Симони, программировал он на **С++** . Его код смотрелся очень необычно. Когда его спрашивали на чем это написано тот отвечал на венгерском. Говорят по этому поводу им была написана диссертация.

Главные соглашения **MFC**.

Префикс тип пример

С класс или структура CPoint,CPrintInfo

m\_ переменная член класса m\_pDoc

Итак, все объявления классов и структур начинаются с **С**, а если переменная является членом класса, то с **m\_**. Дальше таблица соответствий префиксов типам:

Префикс Тип Пример

ch char chGrade

ch TCHAR chName

b BOOL bEnabled

n int nLength

n UINT nLength

w WORD wPos

l LONG lOffset

dw DWORD dwRange

p \* pDoc

lp FAR\* lpDoc

lpsz LPSTR lpszName

lpsz LPCSTR lpszName

lpsz LPCTSTR lpszName

h handle hWnd

lpfn callback lpfnAbort

Так же рекомендованы префикс для идентификаторов и выделены соответствующие диапазоны:

Префикс Тип Пример Диапазон

IDR\_ Идентификатор ресурса нескольких типов IDR\_MAINFRAME 1 to 0x6FFF

IDD\_ Диалоговый ресурс IDD\_SPELL\_CHECK 1 to 0x6FFF

HIDD\_ Диалоговый ресурс для помощи HIDD\_SPELL\_CHECK 0x20001 to 0x26FF

IDB\_ Изображение IDB\_COMPANY\_LOGO 1 to 0x6FFF

IDC\_ Курсор IDC\_PENCIL 1 to 0x6FFF

IDI\_ Икона IDI\_NOTEPAD 1 to 0x6FFF

ID\_ \_ Команда меню или панели инструментов ID\_TOOLS\_SPELLING 0x8000 to 0xDFFF

HID\_ Команда помощи HID\_TOOLS\_SPELLING 0x18000 to 0x1DFFF

IDP\_ Окно сообщения строка IDP\_INVALID\_PARTNO 8 to 0xDFFF

HIDP\_ Окно сообщения помошь HIDP\_INVALID\_PARTNO 0x30008 to 0x3DFFF

IDS\_ Элемент управления на диалоговом окне IDS\_COPYRIGHT 1 to 0x7FFF

IDC\_ Control within dialog box IDC\_RECALC 8 to 0xDFFF

Давайте разберем несколько примеров:

hwndMain

h дискриптор

wnd окна

Main имя Main

lpsztext

lp дальний указатель

sz на строку

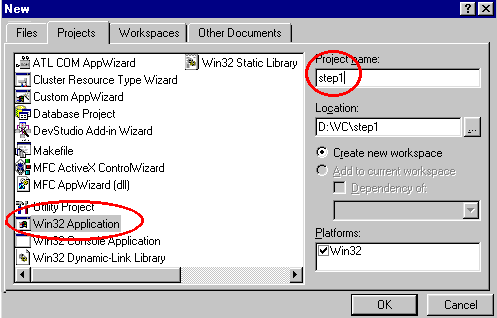
Text имя Text

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?296) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?298) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc6.html)  
Автор **Каев Артем**.

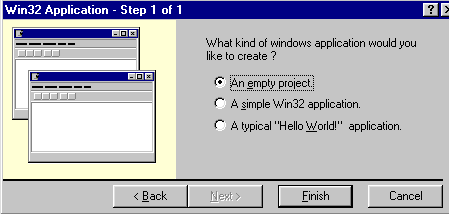
**Шаг 1 - Минимальный проект**

**Создание проекта**

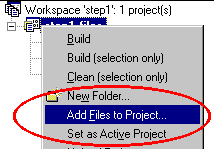
Создадим проект. Здесь все просто - выберем **Win32 Application**, а **AppWizard** пока отложим. Если Вы раньше программировали на **OWL**(я так и делал) или вообще не программировали под **Windows**(так я тоже делал), результаты работы этого шамана приведут в уныние даже настоящего мужчину. Все работает, но как ? Выбираем **File New** и переключаемся на вкладку **Project**. Введя имя проекта нажимаем **OK**.



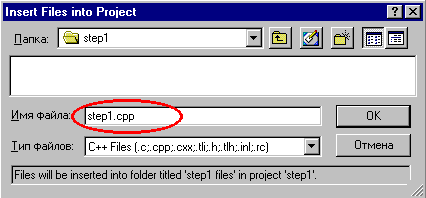
Появится окошко мастера проектов. В нем мы выберем пустой проект - **An Empty Project**. И нажмем кнопочку **Finish** для подтверждения своего выбора. **IDE VC++** начнет генерацию серии файлов необходимых для работы с проектом и его сборки.



Добавим в наш проект файл, в котором будет исходный код. Мы же с вами выбрали пустой проект и нам нужен **cpp** файл в этом проекте. Вам нужно перейти на вкладку **FileView** и щелкнуть правой кнопкой мыши на **step1 files**. Выбрать пункт меню **Add File to Project**.



Появится диалоговое окно добавления файла в проект **Insert Files Into Project**. У нас файла нет. Поэтому надо просто написать его имя и нажать **OK**.



Теперь нужно настроить проект так, чтобы он использовал **MFC**. Это очень важно. **MFC** находится в **DLL** файлах. И если наш проект будет использовать **MFC**, мы должны это указать для того, чтобы компилятор мог подключить необходимые **LIB** файлы со ссылками на функции в **DLL**. Если этого не сделать, то проект не соберется. В окне **Build** будут такие ошибки.

Linking...

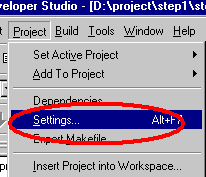
nafxcw.lib(thrdcore.obj) : error LNK2001: unresolved external symbol \_\_endthreadex

nafxcw.lib(thrdcore.obj) : error LNK2001: unresolved external symbol \_\_beginthreadex

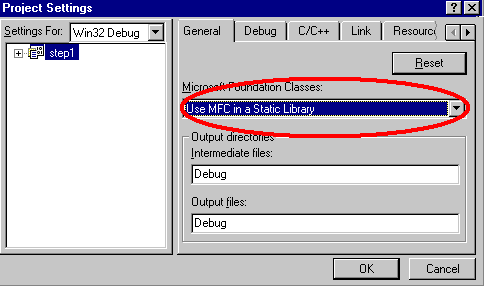
Release/step1.exe : fatal error LNK1120: 2 unresolved externals

Error executing link.exe.

Нам нужно пойти в меню **Project**, а потом **Setting**.



Вот здесь и вот так. Не забудем проверить поддержку **MFC** для всех типов (**Setting For**).



**Создаем код**

В нашем файле запишем код.

// Step1.cpp

// Сайт Первые шаги (http://www.firststeps.ru/)

// Каев Артем (1999-2001)

// создано 1999 г.

// обновление 24.06.2001

#include "afxwin.h" // MFC Основные и стандартные компоненты

class CMainWnd : public [CFrameWnd](http://www.firststeps.ru/mfc/steps/help/r.php?409)

{

public:

CMainWnd(); // Конструктор по умолчанию

};

CMainWnd::CMainWnd()

{

[Create](http://www.firststeps.ru/mfc/steps/help/r.php?79)(NULL,"Step1",WS\_OVERLAPPEDWINDOW,rectDefault,

NULL,NULL); // Создать окно программы

}

class CMyApp : public [CWinApp](http://www.firststeps.ru/mfc/steps/help/r.php?1)

{

public:

CMyApp(); //конструктор по умолчанию

virtual BOOL [InitInstance()](http://www.firststeps.ru/mfc/steps/help/r.php?41);//стандартная инициализация

};

CMyApp::CMyApp() // конструктор главного класса приложения

{}

BOOL CMyApp::[InitInstance()](http://www.firststeps.ru/mfc/steps/help/r.php?41) // стандартная инициализация

{

m\_pMainWnd=new CMainWnd(); // создать класс окна

ASSERT(m\_pMainWnd); // проверить его правильность

m\_pMainWnd->[ShowWindow](http://www.firststeps.ru/mfc/steps/help/r.php?174)(SW\_SHOW);// Показать окно

m\_pMainWnd->[UpdateWindow()](http://www.firststeps.ru/mfc/steps/help/r.php?165); // Обновить окно

return TRUE; // Вернуть что все нормально

};

CMyApp theApp; // запуск приложения

Теперь попробуем во всем этом разобраться:

Первая строка включает необходимый файл заголовка с описанием классов, функций и переменных, он ссылается ещё на кучу других, а они на другие и так до **windows.h**. Вот почему **windows.h** нам и не понадобился. В каждой программе на **С++** есть главная функция программы, в **Dos** это **main()**, в **Windows** - [WinMain()](http://www.firststeps.ru/mfc/winapi/win/r.php?103). Эта функция обеспечивает запуск программы. Она проверяет операционную среду, выполняет некоторые настройки и, если все нормально, непосредственно передает управление вашему коду. Но у нас-то её нет ! В **MFC** есть класс [CWinApp](http://www.firststeps.ru/mfc/steps/help/r.php?1), который и включает главную функцию программы. Естественно, что в приложении она может быть только одна.

Мы с вами создали **CMyApp**, как производный от [CWinApp](http://www.firststeps.ru/mfc/steps/help/r.php?1) и унаследовали все его свойства, методы и т.д. В своем классе мы объявили конструктор по умолчанию (без параметров). Он необходим, иначе вы не скомпилируете программу. Здесь все логично мы в коде объявляем статический класс производный от нашего **CMyApp** в строке **CMyApp theApp;**. Как видите он тоже без параметров.

[CWinApp](http://www.firststeps.ru/mfc/steps/help/r.php?1) имеет виртуальный метод [InitInstance()](http://www.firststeps.ru/mfc/steps/help/r.php?41). Виртуальный он потому, что это только заглушка. Мы с вами написали свой метод в виде функции. Этот метод должен возвращать ненулевое значение, если инициализация прошла нормально, а иначе 0. Он предназначен, чтобы вы могли описать класс окна программы и отобразить окно на экране.

Дальше мы должны знать, что в [CWinApp](http://www.firststeps.ru/mfc/steps/help/r.php?1) есть указатель на класс окна - **m\_pMainWnd**. Этот указатель общедоступный и вы могли догадаться, так как он нигде не объявлен. Ситуация с ним такая - Если по окончанию [InitInstance()](http://www.firststeps.ru/mfc/steps/help/r.php?41) он будет равен нулю, то приложение завершится. И естественно, когда приложение завершается он обнуляется. Строкой **m\_pMainWnd=new CMainWnd();** мы присваиваем этому указателю адрес класса **CMainWnd**, который порожден от [CFrameWnd](http://www.firststeps.ru/mfc/steps/help/r.php?409). Последний из них отвечает за работу окна программы в стиле одного документа (**SDI - single document interface**). Он создает окно, управляет сообщениями и т.д. В конструкторе этого класса мы вызвали функцию создания окна, в которой указали кучу параметров.

Первый параметр указывает на класс **Windows**, он нам не нужен, пока не нужен и поэтому **NULL**, дальше указали имя окна программы, **WS\_OVERLAPPEDWINDOW** указывает, что окно имеет заголовок и рамку, **rectDefault** говорит о том, что размер окна присвоит **Windows**, и еще некоторые параметры, ими всеми мы займемся позже. Далее следует **ASSERT(m\_pMainWnd);**. Очень хорошая и простая функция. Если создание объекта произойдет, то **m\_pMainWnd** будет отличаться от нуля и всё будет нормально, а иначе вы получите сообщение о том, что в таком файле на такой-то строке ошибка. Вообще это намного лучше, чем смотреть дизассемблированный код или оказаться в исходном коде библиотеки **MFC**, если не повесим **Windows** конечно.

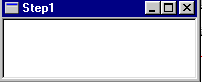
Так как **m\_pMaiWnd** это сейчас тоже что и **CMainWnd**, то мы используем это переменную, чтобы вывести на экран окно:

m\_pMainWnd->[ShowWindow](http://www.firststeps.ru/mfc/steps/help/r.php?174)(SW\_SHOW);

И показать окно.

m\_pMainWnd->[UpdateWindow()](http://www.firststeps.ru/mfc/steps/help/r.php?165);

Вот и все ! При запуске приложения вот такое окно мы получим.



**Шпаргалка**

1. Включить описание классов **MFC**:

#include <afxwin.h>

1. Объявить класс производный от [CWinApp](http://www.firststeps.ru/mfc/steps/help/r.php?1):
2. class CMyApp : public [CWinApp](http://www.firststeps.ru/mfc/steps/help/r.php?1)

.......

1. Объявить переменную этого класса:

CMyApp theApp;

1. Объявить класс производный от [CFrameWnd](http://www.firststeps.ru/mfc/steps/help/r.php?409):
2. class CMainWnd : public [CFrameWnd](http://www.firststeps.ru/mfc/steps/help/r.php?409)

.......

1. В конструкторе класса производного от **CFrameWnd** вызвать функцию **Create()** и выбрать параметры в ней на свой вкус:

[Create](http://www.firststeps.ru/mfc/steps/help/r.php?79)(NULL,"Step1",WS\_OVERLAPPEDWINDOW,rectDefault,NULL,NULL);

1. Переопределить функцию [InitInstance()](http://www.firststeps.ru/mfc/steps/help/r.php?41) класса производного от [CWinApp](http://www.firststeps.ru/mfc/steps/help/r.php?1).
2. В функции [InitInstance()](http://www.firststeps.ru/mfc/steps/help/r.php?41) создать объект производный от [CFrameWnd](http://www.firststeps.ru/mfc/steps/help/r.php?409) и присвоить указателю **m\_pMainWnd** адрес этого объекта.

m\_pMainWnd=new CMainWnd();

1. Проверить указатель на корректность:

ASSERT(m\_pMainWnd);

1. Вызвать функции класса производного от [CFrameWnd](http://www.firststeps.ru/mfc/steps/help/r.php?409) для прорисовки на экране:
2. m\_pMainWnd->[ShowWindow](http://www.firststeps.ru/mfc/steps/help/r.php?174)(SW\_SHOW);// Показать окно
3. m\_pMainWnd->[UpdateWindow()](http://www.firststeps.ru/mfc/steps/help/r.php?165); // Обновить окно

1. Закончить выполнение функции, убедив **MFC**, что все нормально:

return TRUE;

1. Откомпилировать, собрать и запустить программу, насладившись пустым окном и именем **Step1**.

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/1.zip) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?2) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc1.html)  
Автор **Каев Артем**.

## Шаг 2 - Вставка элемента управления в окно (Controls)

**Создание проекта**

Для работы возьмем наш первый проект и внесем изменения (вообще я так и задумал). Пока у нас все в одном **СРР** файле (это не хорошо, исправимся). Самый простой способ - это создать новый проект **step2**. Пустой естественно. Скопировать в него **step1.cpp**. Переименовать его в **step2.cpp**. И вставить в проект **Add File to Project**. В результате весь код с прошлого проекта Вы перенесете в новый.

**Создаем код**

Это нужно поместить где-нибудь после всех **#include**:

#define IDC\_MYBUTTON 100 // Идентификатор кнопки

#define IDC\_MYEDIT 102 // Идентификатор поля редактирования

Необходимо изменить описания конструктора класса окна.

class CMainWnd : public [CFrameWnd](http://www.firststeps.ru/mfc/steps/help/r.php?409)

{

public:

CMainWnd(); // Конструктор по умолчанию

~CMainWnd(); // Деструктор

private:

[CStatic](http://www.firststeps.ru/mfc/steps/help/r.php?452)\* MyStatic; // Указатель на объект надпись

[CButton](http://www.firststeps.ru/mfc/steps/help/r.php?467)\* MyButton; // Указатель на объект кнопка

[CEdit](http://www.firststeps.ru/mfc/steps/help/r.php?488)\* MyEdit; // Указатель на объект поле редактирования

};

Изменения в конструкторе окна. В конструкторе результат создания объектов оператором **new** проверяется через проверку на **NULL**. Если ошибка, то все действия с этим элементом отменятся. На будущее это не самый лучший выход. Вообще-то, наверно, невозможность создания одного элемента должна приводить к завершению программы. Хотя это выбор программиста. В сложной программе, если произойдет ошибка создания объекта нужно выбирать, а на будущее у нас выбора не много. Либо перенести создание объектов в отдельную функцию, где можно проверить результат и закрыть программу, либо использовать исключения (**try,throw,catch**), чтобы отслеживать ошибки в конструкторах. Вы позже всему научитесь :-).

CMainWnd::CMainWnd()

{

[Create](http://www.firststeps.ru/mfc/steps/help/r.php?417)(NULL,"Step2",WS\_OVERLAPPEDWINDOW,rectDefault,

NULL,NULL);// Создать окно программы

// оператор new по умолчанию в случае ошибки вернет NULL

// проверка указателя на NULL дает возможность избавиться от дальнейших ошибок

MyStatic = new [CStatic](http://www.firststeps.ru/mfc/steps/help/r.php?452)();

if (MyStatic!=NULL) MyStatic->[Create](http://www.firststeps.ru/mfc/steps/help/r.php?457)("MyStatic",WS\_CHILD|WS\_VISIBLE|SS\_CENTER,

CRect(10,10,100,50),this); // создали

MyButton = new [CButton](http://www.firststeps.ru/mfc/steps/help/r.php?467)();

if (MyButton!=NULL) MyButton->[Create](http://www.firststeps.ru/mfc/steps/help/r.php?472)("MyButton",WS\_CHILD|WS\_VISIBLE|SS\_CENTER,

CRect(120,10,220,50),this,IDC\_MYBUTTON);

MyEdit = new [CEdit](http://www.firststeps.ru/mfc/steps/help/r.php?488)();

if (MyEdit!=NULL) MyEdit->[Create](http://www.firststeps.ru/mfc/steps/help/r.php?492)(WS\_CHILD|WS\_VISIBLE|WS\_BORDER,

CRect(240,10,340,50),this,IDC\_MYEDIT);

}

Изменения в деструкторе. Когда класс окна в виде объекта будет уже не нужен, автоматически вызывается деструктор. Раз удаляется окно, значит элементы управления внутри него уже не нужны. Самое время их удалить:

CMainWnd::~CMainWnd()

{

if (MyStatic!=NULL) delete MyStatic; // удалить динамический объект

if (MyButton!=NULL) delete MyButton; // удалить динамический объект

if (MyEdit!=NULL) delete MyEdit; // удалить динамический объект

}

**Описание**

**Windows**, в отличие от **DOS**, содержит в себе много элементов управления. Оказалось, что кнопки, поля с надписями, списки и т.д. используются в каждой программе. Сам программный код реализации этих элементов находится в **DLL Windows**. То есть в вашем компиляторе нет реализации этих функций в библиотеках (**lib**), там только ссылки. Вот мы и используем их в своей программе. У всех элементов управления есть одно свойство. Они должны все иметь идентификатор - число, которое определяет этот элемент управления. Правило простое - хочешь элемент управления - определи идентификатор.

Первыми строками используя **#define** мы объявляем идентификаторы. Смысл этих команд в том, что набору символов присваивается число. И при компиляции вместо набора символов подставляется число, что очень удобно.

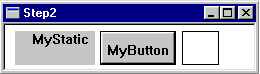
Дальше в класс рамки окна вставляются указатели на объекты элементов управления. Это только указатели, т.е. числа. Объекта самого нет. Каждый объект использует память и до того, как он не понадобится, хранится только указатель.

В конструкторе окна мы и создадим объекты **new**, а после этого вызовем функцию **Create()**, которая из объекта создаст элемент управления. Если вы обратили внимание, то у **MyStatic** нет идентификатора. Это обман, он просто задан по умолчанию, как **0xffff**. Это подтверждает правило кесарю - кесарево, а элементу управления - идентификатор. Функция **Create()** используется очень часто и есть у многих объектов. Обычно она требует:

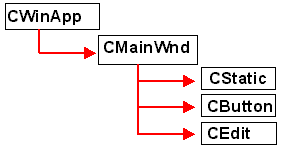
1. Строку для имени или надписи
2. Стиль
3. Размеры и положение
4. Куда вставлять
5. Идентификатор

**this** означает, что "вставлять туда где находишься" (мы в классе рамки окна).

Вот так должна выглядеть программа после запуска.



Обратите внимание на схему ниже. На ней показана вложенность объектов. То есть кому, какой объект принадлежит. Создаются объекты в порядке от [**CWinApp**](http://www.firststeps.ru/mfc/steps/help/r.php?1) и дальше, а разрушаются в обратном.



**Шпаргалка**

1. В конструкторе окна надо объявить указатели на элементы управления:
2. [CStatic](http://www.firststeps.ru/mfc/steps/help/r.php?452)\* MyStatic;

......

1. Указатели превратить в объекты и вызвать функцию **Create()**:
2. MyButton = new [CButton](http://www.firststeps.ru/mfc/steps/help/r.php?467)();
3. MyButton->[Create](http://www.firststeps.ru/mfc/steps/help/r.php?472)("MyButton",WS\_CHILD|WS\_VISIBLE|SS\_CENTER,
4. CRect(120,10,220,50),this,IDC\_MYBUTTON);

......

1. Обеспечить удаление динамических объектов при завершении программы:
2. if (MyStatic!=NULL) delete MyStatic;
3. if (MyButton!=NULL) delete MyButton;
4. ......
5. Откомпилировать, собрать и запустить.

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/2.zip) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?1) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?3) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc1.html)  
Автор **Каев Артем**.

## Шаг 3 - События рамки окна

**Создание проекта**

Продолжаем развитие нашего проекта дальше. Можете опять скопировать файл, как мы это делали в [предыдущем шаге](http://www.firststeps.ru/mfc/steps/r.php?2).

**Создаем код**

В описание рамки окна необходимо внести следующие изменения:

class CMainWnd : public [CFrameWnd](http://www.firststeps.ru/mfc/steps/help/r.php?409)

{

public:

CMainWnd();

afx\_msg void [OnLButtonDblClk](http://www.firststeps.ru/mfc/steps/help/r.php?359)(UINT, CPoint); // виртуальная процедура ответа на левую кнопку

afx\_msg void [OnRButtonDblClk](http://www.firststeps.ru/mfc/steps/help/r.php?369)(UINT, CPoint); // виртуальная процедура ответа на правую кнопку

afx\_msg void [OnKeyDown](http://www.firststeps.ru/mfc/steps/help/r.php?357)(UINT, UINT, UINT); // виртуальная процедура ответа на клавишу

~CMainWnd(); // деструктор

private:

[CStatic](http://www.firststeps.ru/mfc/steps/help/r.php?452)\* MyStatic;

[CButton](http://www.firststeps.ru/mfc/steps/help/r.php?467)\* MyButton;

[CEdit](http://www.firststeps.ru/mfc/steps/help/r.php?488)\* MyEdit;

DECLARE\_MESSAGE\_MAP(); // таблица откликов

};

И вписать следующие процедуры после описания класса.

BEGIN\_MESSAGE\_MAP(CMainWnd, [CFrameWnd](http://www.firststeps.ru/mfc/steps/help/r.php?409)) // таблица откликов на сообщения

ON\_WM\_LBUTTONDBLCLK() // реакция на нажатие левой кнопки мыши

ON\_WM\_RBUTTONDBLCLK() // реакция на нажатие правой кнопки мышки

ON\_WM\_KEYDOWN() // реакция на нажатие клавиши

END\_MESSAGE\_MAP()

void CMainWnd::[OnKeyDown](http://www.firststeps.ru/mfc/steps/help/r.php?357)(UINT, UINT, UINT)

{

AfxMessageBox(" Key Button Down ");

}

void CMainWnd::[OnRButtonDblClk](http://www.firststeps.ru/mfc/steps/help/r.php?369)(UINT, CPoint)

{

AfxMessageBox(" Rigth Button Click ");

}

void CMainWnd::[OnLButtonDblClk](http://www.firststeps.ru/mfc/steps/help/r.php?359)(UINT, CPoint)

{

AfxMessageBox(" Left Button Click ");

}

**Описание**

Помимо наличия окна и элементов управления неплохо, если программа в виде класса окна сможет реагировать на события. **Windows** порождает кучу событий, если вы двигаете мышку, нажимаете клавишу или пьете кофе, операционная среда знает все. Свои знания о ситуации она передает с помощью сообщений. На программу сообщения льются непрекращаемым потоком. Нам из этого потока необходимо выбрать нужные и обработать.

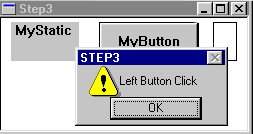
Добавление в класс макроса **DECLARE\_MESSAGE\_MAP()** говорит о том, что данный класс будет реагировать на события.

Если класс реагирует на события, то необходимо описать таблицу откликов на них. Таблица откликов находится между определениями **BEGIN\_MESSAGE\_MAP(Класс реагирующий, Класс родитель)** и **END\_MESSAGE\_MAP()**. Зачем при определении таблицы откликов указывать класс, который будет производить реакцию? Но таблиц откликов может быть много для разных классов и как интересно компилятор разберется, кому она принадлежит? Класс родителя необходим для обработки стандартных сообщений, даже если вы их явно не обрабатываете, например, закрыть приложение.

Между этими определениями вы пишете реакцию. В **MFC** предусмотрена обработка кучи событий, и их описание начинается с **ON\_WM\_message**. В нашем случае это стандартные события **WINDOWS** по реакции на нажатие клавиш и мыши.

Для каждого стандартного события описана виртуальная функция, которая будет вызываться при его возникновении. Имя и параметры этой функции строго определены и ошибаться здесь нельзя. Правило выглядит так: хочешь реакцию на стандартное событие - опиши его в таблице откликов и замени виртуальную процедуру. **Afxmsg** воспринимайте как аналог **virtual**.

После описания всех функций необходимо их реально создать, что и делается далее. **AfxMessageBox()** - это вызов простого диалогового окна с надписью. Результат двойного щелчка мышкой смотрите на картинке ниже.



**Шпаргалка**

1. Указать классу, что он реагирует на события:

DECLARE\_MESSAGE\_MAP();

1. Создать таблицу откликов на события:
2. BEGIN\_MESSAGE\_MAP(CMainWnd, [CFrameWnd](http://www.firststeps.ru/mfc/steps/help/r.php?409))
3. ......

END\_MESSAGE\_MAP()

1. Описать процедуры, соответствующие таблице, в классе, реагирующем на события:
2. afx\_msg void [OnLButtonDblClk](http://www.firststeps.ru/mfc/steps/help/r.php?359)(UINT, CPoint);

......

1. Создать реализацию событий:
2. void CMainWnd::[OnKeyDown](http://www.firststeps.ru/mfc/steps/help/r.php?357)(UINT, UINT, UINT)
3. {
4. AfxMessageBox(" Key Button Down ");
5. }
6. Дальше Вам известно. Результат превосходит все ожидания. Только не забудьте, что реакция мышки на двойной щелчок.

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/3.zip) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?2) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?4) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc1.html)  
Автор **Каев Артем**.

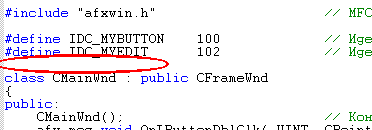
## Шаг 4 - События элемента управления

**Создание проекта**

Продолжаем развитие нашего проекта дальше.

**Создаем код**

Это вписывается после последнего **#define**. Вот сюда:



Мы записываем объявление класса в самое начало, чтобы его было видно во всем файле.

class CMyButton: public [CButton](http://www.firststeps.ru/mfc/steps/help/r.php?467)

{

public:

afx\_msg void [OnLButtonDblClk](http://www.firststeps.ru/mfc/steps/help/r.php?359)(UINT, CPoint);

afx\_msg void [OnRButtonDblClk](http://www.firststeps.ru/mfc/steps/help/r.php?369)(UINT, CPoint);

private:

DECLARE\_MESSAGE\_MAP(); // таблица откликов кнопки

};

void CMyButton::[OnLButtonDblClk](http://www.firststeps.ru/mfc/steps/help/r.php?359)(UINT, CPoint)

{

[MoveWindow](http://www.firststeps.ru/mfc/steps/help/r.php?120)(CRect(120,100,220,150),TRUE);

}

void CMyButton::[OnRButtonDblClk](http://www.firststeps.ru/mfc/steps/help/r.php?369)(UINT, CPoint)

{

[MoveWindow](http://www.firststeps.ru/mfc/steps/help/r.php?120)(CRect(120,10,220,50),TRUE);

}

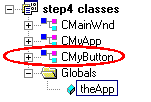
BEGIN\_MESSAGE\_MAP(CMyButton, [CButton](http://www.firststeps.ru/mfc/steps/help/r.php?467)) // таблица откликов на сообщения

ON\_WM\_LBUTTONDBLCLK()

ON\_WM\_RBUTTONDBLCLK()

END\_MESSAGE\_MAP()

После добавления описания класса в окне **ClassView** мы сможем увидеть, что добавился новый класс:



И изменяем класс кнопки в рамке окна:

class CMainWnd : public [CFrameWnd](http://www.firststeps.ru/mfc/steps/help/r.php?409)

{

public:

CMainWnd();

afx\_msg void [OnLButtonDblClk](http://www.firststeps.ru/mfc/steps/help/r.php?359)(UINT, CPoint);

afx\_msg void [OnRButtonDblClk](http://www.firststeps.ru/mfc/steps/help/r.php?369)(UINT, CPoint);

afx\_msg void [OnKeyDown](http://www.firststeps.ru/mfc/steps/help/r.php?357)(UINT, UINT, UINT);

private:

[CStatic](http://www.firststeps.ru/mfc/steps/help/r.php?452)\* MyStatic; // Элемент управления для вывода текста

**CMyButton\* MyButton;// Элемент управления кнопка**

[CEdit](http://www.firststeps.ru/mfc/steps/help/r.php?488)\* MyEdit; // Элемент редактирования

DECLARE\_MESSAGE\_MAP(); // таблица откликов

};

Меняем часть конструктора:

CMainWnd::CMainWnd()

{

Create(NULL,"Step4",WS\_OVERLAPPEDWINDOW,rectDefault,

NULL,NULL); // Создать окно программы

// оператор new по умолчанию в случае ошибки вернет NULL

// проверка указателя на NULL дает возможность избавиться от дальнейших ошибок

MyStatic = new [CStatic](http://www.firststeps.ru/mfc/steps/help/r.php?452)();

if (MyStatic!=NULL) MyStatic->[Create](http://www.firststeps.ru/mfc/steps/help/r.php?457)("MyStatic",WS\_CHILD|WS\_VISIBLE|SS\_CENTER,

CRect(10,10,100,50),this); // создали

**MyButton = new CMyButton(); // Меняем класс, на основе которого создается объект**

**if (MyButton!=NULL) MyButton->**[**Create**](http://www.firststeps.ru/mfc/steps/help/r.php?472)**("MyButton",WS\_CHILD|WS\_VISIBLE|SS\_CENTER,**

**CRect(120,10,220,50),this,IDC\_MYBUTTON);**

MyEdit = new [CEdit](http://www.firststeps.ru/mfc/steps/help/r.php?488)();

if (MyEdit!=NULL) MyEdit->[Create](http://www.firststeps.ru/mfc/steps/help/r.php?492)(WS\_CHILD|WS\_VISIBLE|WS\_BORDER,

CRect(240,10,340,50),this,IDC\_MYEDIT);

}

**Описание**

Главная идея этого шага понять, каким методом можно изменить поведение элемента управления, и не только его. С помощью таблиц откликов можно поменять поведение практически любого объекта **MFC**.

Первым делом, мы создадим свой класс кнопки. Чтобы не делать все сначала, мы возьмем готовый [CButton](http://www.firststeps.ru/mfc/steps/help/r.php?467) и породим от него свой **CMyButton**. Так же, как и вы унаследовали от родителей глаза, уши и плохие привычки, так и **CMyButton** точная копия [CButton](http://www.firststeps.ru/mfc/steps/help/r.php?467). Правило такое - сын точная копия папы. **Public** означает, что папа родной, не отчим. Получение точной копии выглядит так:

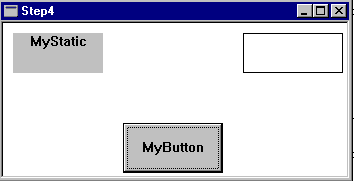
class Сын : public Папа

Пора перевоспитать наш [CButton](http://www.firststeps.ru/mfc/steps/help/r.php?467) под новые требования. Добавили таблицу откликов и описали функции реакции - это знакомо по [предыдущему шагу](http://www.firststeps.ru/mfc/steps/r.php?3). Правило: детей надо воспитывать, иначе толку никакого.

Внутри функции отклика на кнопку есть функция [MoveWindow](http://www.firststeps.ru/mfc/steps/help/r.php?120) - эта функция перемещает окно в заданное место. В описании класса [CButton](http://www.firststeps.ru/mfc/steps/help/r.php?467) этого свойства нет. Но класс [CButton](http://www.firststeps.ru/mfc/steps/help/r.php?467) тоже сын от [CWnd](http://www.firststeps.ru/mfc/steps/help/r.php?71), а в нем это свойство есть. Очередное правило: нет метода у сына - посмотри у папы.

И последнее. В классе главного окна изменить ссылку с [CButton](http://www.firststeps.ru/mfc/steps/help/r.php?467) на **CMyButton**. Здесь логично. Выйти замуж за папу или сына разница большая. Естественно и процесс создания и вызова методов нужно изменять.

Все по старому: компиляция, сбор, запуск. По двойному щелчку левой кнопкой объект кнопка переместится вниз, а по щелчку правой вверх. Вот это да!



**Шпаргалка**

1. Создать сына от готового класса:
2. class CMyButton: public [CButton](http://www.firststeps.ru/mfc/steps/help/r.php?467)
3. Воспитать сына:
4. class CMyButton: public [CButton](http://www.firststeps.ru/mfc/steps/help/r.php?467)
5. {
6. public:
7. afx\_msg void [OnLButtonDblClk](http://www.firststeps.ru/mfc/steps/help/r.php?359)(UINT, CPoint);
8. ......
9. Переопределить ссылку:

CMyButton\* MyButton;

1. Изменить процесс создания и вызова методов сына, заменив папу (через ссылку):
2. MyButton = new CMyButton();
3. if (MyButton!=NULL) MyButton-> [Create](http://www.firststeps.ru/mfc/steps/help/r.php?472)("MyButton",WS\_CHILD|WS\_VISIBLE|SS\_CENTER,
4. CRect(120,10,220,50),this,IDC\_MYBUTTON);

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/4.zip) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?3) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?5) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc1.html)  
Автор **Каев Артем**.

## Шаг 5 - Создание панели состояния

### Создание проекта

Продолжаем развитие проекта. В предыдущих шагах все понятно, сосредоточимся на главном. Сейчас мы будем создавать панель состояния, и результат станет очень похожим на результаты использования волшебника!

### Создаем код

В описания включаемых файлов добавим:

#include "afxext.h" // MFC Расширения

В описание класса рамки окна:

class CMainWnd : public [CFrameWnd](http://www.firststeps.ru/mfc/steps/help/r.php?409)

{

public:

CMainWnd(); // Конструктор по умолчанию

afx\_msg void [OnLButtonDblClk](http://www.firststeps.ru/mfc/steps/help/r.php?359)( UINT, CPoint ); // виртуальная процедура ответа на левую кнопку

afx\_msg void [OnRButtonDblClk](http://www.firststeps.ru/mfc/steps/help/r.php?369)( UINT, CPoint ); // виртуальная процедура ответа на правую кнопку

afx\_msg void [OnKeyDown](http://www.firststeps.ru/mfc/steps/help/r.php?357)( UINT, UINT, UINT ); // виртуальная процедура ответа на клавишу

**int** [**OnCreate**](http://www.firststeps.ru/mfc/steps/help/r.php?311)**(LPCREATESTRUCT lpCreateStruct); // эта функция вызывается при создании окна**

~CMainWnd(); // Деструктор

private:

[CStatic](http://www.firststeps.ru/mfc/steps/help/r.php?452)\* MyStatic; // Указатель на объект надпись

CMyButton\* MyButton; // Элемент управления кнопка

[CEdit](http://www.firststeps.ru/mfc/steps/help/r.php?488)\* MyEdit; // Указатель на объект поле редактирования

[**CStatusBar**](http://www.firststeps.ru/mfc/steps/help/r.php?532) **m\_wndStatusBar; // класс панели состояния**

DECLARE\_MESSAGE\_MAP(); // таблица откликов

};

В таблице откликов:

BEGIN\_MESSAGE\_MAP(CMainWnd, CFrameWnd) // таблица откликов на сообщения

ON\_WM\_LBUTTONDBLCLK() // реакция на нажатие левой кнопки мыши

ON\_WM\_RBUTTONDBLCLK() // реакция на нажатие правой кнопки мышки

ON\_WM\_KEYDOWN() // реакция на нажатие клавиши

**ON\_WM\_CREATE() // событие создания окна**

END\_MESSAGE\_MAP()

Реализация объявленной процедуры:

int CMainWnd::[OnCreate](http://www.firststeps.ru/mfc/steps/help/r.php?311)(LPCREATESTRUCT lpCreateStruct)

{

if (CFrameWnd::[OnCr eate](http://www.firststeps.ru/mfc/steps/help/r.php?311)(lpCreateStruct) == -1)

return -1;

m\_wndStatusBar.[Create](http://www.firststeps.ru/mfc/steps/help/r.php?536)(this);

return 0;

}

### Описание

Для создания панели состояния нам естественно нужен **MFC** класс. В файле **afxwin.h** его нет. Но зато он есть в **afxext.h**. Его и надо подключить.

Следующим шагом этот класс мы и включаем в класс окна. Заметьте, он включен статически. То есть при создании объекта окна, автоматически создается класс панели состояния, и он будет жить, пока живет объект рамки окна. Это и нужно. Хотя можно и динамически.

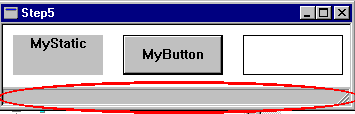
Определили функцию **OnCreate()**. Эта функция при соблюдении условия (смотри ниже) вызывается при создании окна. Но она не виртуальная и код этой функции нам нужен. Вывод: вы можете переопределить любой метод (функция) класса. Вот!

Определяем реакцию на событие создания окна. Это нужно делать обязательно во многих случаях. Если этого не сделать, то функция не выполнится никогда. Это очень важно. Если вы переопределяете метод (функцию) класса и ничего не происходит, проверьте, а нет ли необходимого сообщения для работы этой функции. Например, [OnPaint()](http://www.firststeps.ru/mfc/steps/help/r.php?328) требует событие **ON\_WM\_PAINT**, иначе на экране ничего не будет. Как правило, многие методы требуют определения реакции на соответствующее сообщение.

И, наконец, пишем саму функцию. Результат работы этой функции число. Если -1, то окно не создано. И код реализации окна нам нужен, поэтому и был вызван метод папы в этой функции - [CFrameWnd](http://www.firststeps.ru/mfc/steps/help/r.php?409)::[OnCreate](http://www.firststeps.ru/mfc/steps/help/r.php?311). На параметры не обращайте пока внимание, не в этом суть. Правило - переопределил метод, вызови метод папы **Папа::Метод()** и делай дальше, что хочешь.

Код дальше вызывает метод класса "Панель состояния" для её создания. С параметром куда вставлять - сюда, в рамку окна.

Как обычно и запуск. Панель есть внизу, как настоящая, но пока пустая. Всё, пора пить кофе.



### Шпаргалка

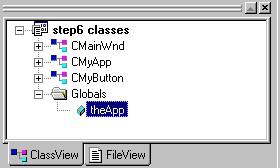
1. Включаем библиотеку расширений **MFC**:
2. #include "afxext.h"
3. В класс рамки окна добавляем процедуры создания и описание класса панели состояния.
4. int [OnCreate](http://www.firststeps.ru/mfc/steps/help/r.php?311)(LPCREATESTRUCT lpCreateStruct);
5. [CStatusBar](http://www.firststeps.ru/mfc/steps/help/r.php?532) m\_wndStatusBar;
6. Добавить в таблицу откликов реакцию на создание окна:
7. ON\_WM\_CREATE()
8. Описать процедуру создания окна и добавить создание панели состояния:
9. m\_wndStatusBar.[Create](http://www.firststeps.ru/mfc/steps/help/r.php?536)(this);

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/5.zip) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?4) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?6) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc1.html)  
Автор **Каев Артем**.

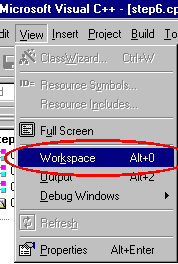
## Шаг 6 - Создание меню

### Создание проекта

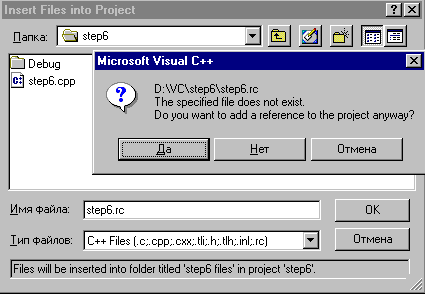
Будем развивать наш проект. Но в нем нам нужно создать файл ресурсов. Тут вот придется потрудиться. У вас в левом углу должно быть окно **WorkSpace**. Вот такое:



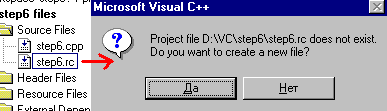
Если его нет, включите через **View - WorkSpace**:



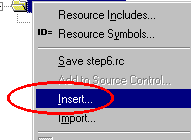
В этом окне есть вкладка **File View**. Нам туда. Там надо создать файл **step6.rc**, как в [первом шаге](http://www.firststeps.ru/mfc/steps/r.php?1) создавали **step1.cpp**. У вас спросят нужно ли его создавать на диске, ответьте положительно, пожалуйста.



Но это еще не все. В той же вкладке **WorkSpace** нужно два раза щелкнуть по файлу, чтобы он все-таки создался. Вот такая табличка будет при двойном щелчке. Ответив **ДА** вы создадите файл физически на диске.



Далее нам на вкладку **Resource**. Там пусто. Мышку на папку, правую кнопку вниз, пункт меню **Insert**:



В окне выбираем **Menu** и **New**. В результате в ресурсах появится меню. Создайте пункты меню. Пункт **File** и ниже **Exit**. Обязательно поменяйте идентификатор меню на **IDR\_MENU**.



В списке файлов (**WorkSpace -> FileView**) вы не найдете **resource.h**, если необходимо добавьте его (**Add File to Project**). В этом файле будут находиться идентификаторы ресурсов.

### Создаем код

Добавляем в раздел описаний ссылку на файл с идентификаторами ресурсов.

......

#include "afxext.h" // MFC Расширения

**#include "resource.h" // Идентификаторы ресурсов**

#define IDC\_MYBUTTON 100 // Идентификатор кнопки

......

Добавляем описание класса меню в класс рамки окна. Замечаете, на всё есть класс, это класс !!!

class CMainWnd : public CFrameWnd

{

public:

CMainWnd(); // Конструктор по умолчанию

afx\_msg void OnLButtonDblClk(UINT, CPoint); // виртуальная процедура ответа на левую кнопку

afx\_msg void OnRButtonDblClk(UINT, CPoint); // виртуальная процедура ответа на правую кнопку

afx\_msg void OnKeyDown(UINT, UINT, UINT); // виртуальная процедура ответа на клавишу

int OnCreate(LPCREATESTRUCT lpCreateStruct); // эта функция вызывается при создании окна

~CMainWnd(); // Деструктор

private:

CStatic\* MyStatic; // Указатель на объект надпись

CMyButton\* MyButton; // Элемент управления кнопка

CEdit\* MyEdit; // Указатель на объект поле редактирования

CStatusBar m\_wndStatusBar; // класс панели состояния

**CMenu m\_wndMenu; // Это наш класс Меню**

DECLARE\_MESSAGE\_MAP();// таблица откликов

};

Добавляем в процедуру создания окна команды создания меню:

int CMainWnd::OnCreate(LPCREATESTRUCT lpCreateStruct)

{

if (CFrameWnd::OnCreate(lpCreateStruct) == -1) return -1;

m\_wndStatusBar.Create(this);

m\_wndMenu.LoadMenu(IDR\_MENU); // Загрузить меню из файла ресурса

SetMenu(&m\_wndMenu); // Установить меню

return 0;

}

Собирайте программу и запускайте. Обратите внимание на то, что при компиляции программы производится и компиляция ресурсов.

--------------------Configuration: step6 - Win32 Debug--------------------

Compiling resources...

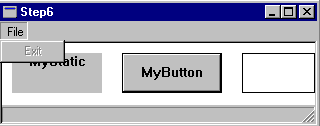
Compiling...

step6.cpp

Linking...

step6.exe - 0 error(s), 0 warning(s)

Вы увидите, что у нас появилось меню.



### Описание

Использование ресурсов является таким же приемом программирования в **Windows**, как прямое обращение к видеопамяти в **DOS**. Использовать ресурсы необязательно, всё можно сделать динамически! Но в ресурсах есть и преимущества. Вы сталкивались с тем, что Ваши специалисты теряются в английском? У Вас было желание перевести программу на русский? Любую программу с ресурсами можно изменить без перекомпиляции **EXE** файла. Меню, диалоговые панели, строки подсказки, иконы, картинки все это чаше всего находится в ресурсах. В **Borland C++ 4.5** была очень хорошая программа **Resource WorkShop**. Она позволяла извлечь ресурсы из файла, редактировать и сохранять их на место, самое главное это отдельная программа и очень удобная.

Как и с элементами управления, каждый элемент ресурсов должен иметь идентификатор. В **Windows** идентификаторы на каждом шагу, как в известном мультфильме Билл всех подсчитал. Правило очередное - каждый ресурс должен быть пронумерован.

Идентификаторы ресурсов хранятся в **\*.h** файлах. При создании файла ресурсов **VC** автоматически создает файл с описанием идентификатора ресурсов - **"resource.h"**, его то мы и подключили первой командой. Как бы иначе программа догадалась, что они существуют?

Далее мы указали классу окна, что у нас есть класс меню. Он статический, это мы уже делали ранее.

После этого для создания и отображения меню необходимо выбрать место для загрузки меню из ресурсов. **OnCreate()** на данный момент самое подходящее, но не обязательное.

**LoadMenu()** функция загрузки меню из ресурсов. Она возвращает значение типа **BOOL** указывающее на успех операции. Я это упустил сознательно. Если есть желание можете написать что-то типа **if (m\_wndMenu.LoadMenu(IDC\_MENU)) .... else ....** . Параметр только один - идентификатор меню в ресурсах.

**SetMenu()** устанавливает меню в окне. Она требует адрес класса типа **CMenu**. Имея статический класс **CMenu** адрес можно получить применив операцию взятия адреса **&CMenu**. Её мы и применили. Правило для статических классов **Адрес Класса = &Класс**.

### Шпаргалка

1. Объявить класс меню:
2. ......
3. CMenu m\_wndMenu;
4. ......
5. Загрузить и установить меню:
7. ......
8. m\_wndMenu.LoadMenu(IDR\_MENU);
9. SetMenu(&m\_wndMenu);
10. ......

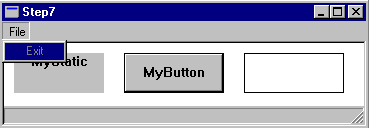
[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/6.zip) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?5) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?7) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc1.html)  
Автор **Каев Артем**.

## Шаг 7 - Обработка команд меню

### Создание проекта

Мучаем дальше наш проект. Если Вы захотите перенести файлы из предыдущего проекта, то учтите, что нужно будет перенести еще и **resource.h** и **step6.rc**. Эти файлы нужно добавить в проект, перед добавлением **step6.rc** нужно переименовать в **step7.rc**.

К моменту начала ваш проект должен иметь меню, и пункт меню **Exit** должен быть серым. Есть песня со словами "был-бы серым, но кто-то взял и покрыл меня сажей". Или в этом роде, не в этом суть, суть в процессе. Меню серое, а пока серое стучи не стучи толку нет. Вот мы и будем делать его черным.



### Создаем код

Добавляем функцию к классу рамки окна для обработки команд меню. Эта функция будет использоваться в обработчике событий. Она должна быть типа **void** и без параметров.

class CMainWnd : public CFrameWnd

{

public:

CMainWnd(); // Конструктор по умолчанию

afx\_msg void OnLButtonDblClk( UINT, CPoint ); // виртуальная процедура ответа на левую кнопку

afx\_msg void OnRButtonDblClk( UINT, CPoint ); // виртуальная процедура ответа на правую кнопку

afx\_msg void OnKeyDown( UINT, UINT, UINT ); // виртуальная процедура ответа на клавишу

int OnCreate(LPCREATESTRUCT lpCreateStruct); // эта функция вызывается при создании окна

**void MenuExit(); // Процедура реакции на выбор пункта меню**

~CMainWnd(); // Деструктор

private:

CStatic\* MyStatic; // Указатель на объект надпись

CMyButton\* MyButton; // Элемент управления кнопка

CEdit\* MyEdit; // Указатель на объект поле редактирования

CStatusBar m\_wndStatusBar; // класс панели состояния

CMenu m\_wndMenu; // Это наш класс Меню

DECLARE\_MESSAGE\_MAP(); // таблица откликов

};

А теперь в обработчике событий нужно добавить реакцию на выбор пункта меню:

BEGIN\_MESSAGE\_MAP(CMainWnd, CFrameWnd) // таблица откликов на сообщения

ON\_WM\_LBUTTONDBLCLK() // реакция на нажатие левой кнопки мыши

ON\_WM\_RBUTTONDBLCLK() // реакция на нажатие правой кнопки мышки

ON\_WM\_KEYDOWN() // реакция на нажатие клавиши

ON\_WM\_CREATE() // событие создания окна

**ON\_COMMAND(ID\_FILE\_EXIT,MenuExit) // Вот она , обработка реакции на выбор меню**

END\_MESSAGE\_MAP()

Обратите внимание на идентификатор пункта меню. Откуда я его взял? При создании пункта меню ему задается идентификатор, который записывается в **resource.h**.

// Used by step7.rc

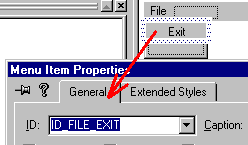
//

#define IDR\_MENU 101

#define ID\_FILE\_EXIT 40001

// Next default values for new objects

При желании его можно поменять в режиме редактирования ресурсов щелкнув правой кнопкой мыши по пункту меню и вызвав свойства (**Properties**).



И сама функция обработки (вообще это процедура, так как не возвращает значение, но вроде в **С++** всё функции ?). Её можно поместить сразу после обработки сообщений.

void CMainWnd::MenuExit()

{

DestroyWindow(); // Уничтожить окно

}

Собирайте проект и запускайте программу. Пункт меню будет черный. Это говорит о том, что его можно использовать. При выборе этого пункта меню программа прекратит свою работу.



### Описание

В классе необходимо описать функцию реакции. Просмотрите предыдущие обработки. Не перепутайте. Объявляете функцию в классе, в котором и будут обрабатывать, и в которой есть обработчик.

Макрос **ON\_COMMAND** предназначен для обработки команд вашего меню. В параметры ставите идентификатор и имя функции. Функция должна иметь тип **void**, что значит пусто, и она не вернет параметры. После Паскаля это удивительно. Что это за функция, которая не возвращает параметры. Это процедура !! Ну это на совести **С++**. То, что она тип **void** логично - куда возвращать результат, в меню что-ли ? И без параметров она, откуда им взяться ? Правило - функция реакции на меню с виду пустая, как бубен.

Функция **DestroyWindow()** разрушает окно. При этом тип у неё **BOOL**, окно и не всегда можно уничтожить, помните это и обрабатывайте, если необходимо. Правило: внутри пустой функции важно содержание!

Программа реагирует на нажатие меню, как я на запах кофе!

### Шпаргалка

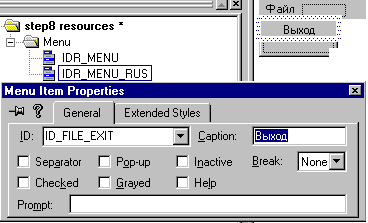
1. Добавляем функцию, которая обработает вызов меню, в класс рамки окна:
2. class CMainWnd : public CFrameWnd
3. {
4. public:
5. ......
6. void MenuExit();
7. ......
8. };
9. Создаем её обработку в обработчике событий класса рамки окна.
10. BEGIN\_MESSAGE\_MAP(CMainWnd, CFrameWnd)
11. ......
12. ON\_COMMAND(ID\_FILE\_EXIT,MenuExit)
13. ......
14. END\_MESSAGE\_MAP()
15. Пишем, что делать:
16. void CMainWnd::MenuExit()
17. {
18. ......
19. }

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/7.zip) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?6) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?8) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc1.html)  
Автор **Каев Артем**.

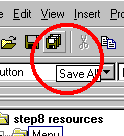
## Шаг 8 - Подмена меню

### Создание прое кта

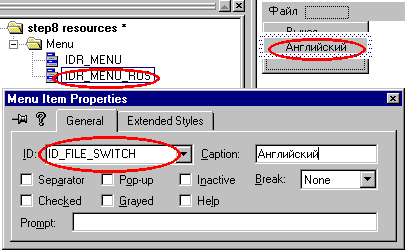
Продолжаем развитие нашего проекта. Нам в проекте нужно создать второе меню, также как и первое, но с идентификатором **IDR\_MENU\_RUS**. И создать пункты **Файл** и **Закрыть**. У пункта меню "закрыть" идентификатор должен совпадать с предыдущим меню. Это необходимо, чтобы не создавать несколько макросов обработки.

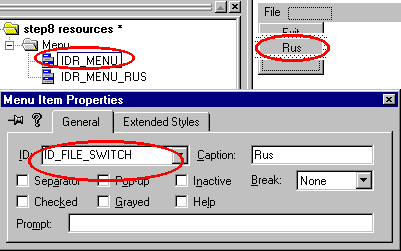


Не забудьте сохранить изменения, после того как добавите меню и его пункты.



И нам нужен еще один пункт в меню. Он будет нужен для того, чтобы переключать язык меню. Идентификатор у него должен быть **ID\_FILE\_SWITCH**. Этот пункт должен присутствовать в обоих меню.





### Создаем код

Заведем переменную, в которой будем хранить состояние меню:

......

void MenuExit(); // Процедура реакции на выбор пункта меню

~CMainWnd(); // Деструктор

private:

**BOOL blMenu; // какое меню активно**

CStatic\* MyStatic; // Указатель на объект надпись

CMyButton\* MyButton; // Элемент управления кнопка

......

При создании класса окна ее надо инициализировать. Мы договоримся, что если **IDR\_MENU**, то **TRUE**, а если **IDR\_MENU\_RUS**, то **FALSE**.

CMainWnd::CMainWnd()

{

**blMenu=FALSE; // инициализация TRUE соответствует IDR\_MENU**

Create(NULL,"Step8",WS\_OVERLAPPEDWINDOW,rectDefault,

NULL,NULL); // Создать окно программы

// оператор new по умолчанию в случае ошибки вернет NULL

// проверка указателя на NULL дает возможность избавиться от дальнейших ошибок

......

Реакция на событие выбора пункта меню в таблице откликов:

BEGIN\_MESSAGE\_MAP(CMainWnd, CFrameWnd) // таблица откликов на сообщения

......

ON\_COMMAND(ID\_FILE\_EXIT,MenuExit) // Вот она, обработка реакции на выбор меню

**ON\_COMMAND(ID\_FILE\_SWITCH,MenuSwitch)// Реакция на выбор пункта о переключении языка меню**

END\_MESSAGE\_MAP()

Описание функции реакции:

class CMainWnd : public CFrameWnd

{

......

void MenuExit(); // Процедура реакции на выбор пункта меню

void MenuSwitch(); // Процедура реакции на выбор пункта меню

**~CMainWnd(); // Деструктор**

private:

......

};

Ну и реализация:

void CMainWnd::MenuSwitch()

{

m\_wndMenu.DestroyMenu(); // Уничтожаем старое

switch (blMenu)

{

case TRUE:

m\_wndMenu.LoadMenu(IDR\_MENU); // Читаем новое

break;

case FALSE:

m\_wndMenu.LoadMenu(IDR\_MENU\_RUS); // Читаем новое

break;

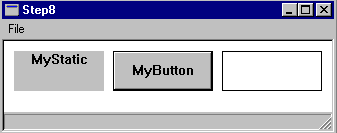
}

blMenu=!blMenu; // Меняет переменную

SetMenu(&m\_wndMenu); // Устанавливаем меню

}

Вот так должна работать Ваша программа:



### Описание

Функция **DestroyMenu()** уничтожает меню. Как видите, эта она является членом класса **CMenu**. Мы уничтожаем не просто меню программы, а меню класса меню связанного с окном программы. Уничтожается меню связанное функцией **SetMenu()**. Дальше Вам известно все. Читаем и устанавливаем.

На эту тему можно фантазировать. Например, меню на английском, немецком, французском на выбор? Или меню, которое постоянно изменяется при разных условиях. Или конкретное меню под конкретное рабочее место, под паролем на основании меню, в котором есть всё? Или .... будут, если подскажите идеи, где это можно применить.

### Шпаргалка

1. Удалить текущее меню:

m\_wndMenu.DestroyMenu();

1. Прочитать новое:

m\_wndMenu.LoadMenu(IDR\_MENU);

1. Установить:

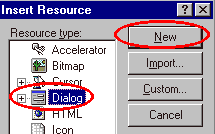
SetMenu(&m\_wndMenu);

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/8.zip) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?7) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?9) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc1.html)  
Автор **Каев Артем**.

## Шаг 9 - Диалоговая панель

### Создание проекта

Продолжаем развитие нашего проекта. В файле ресурсов необходимо создать диалоговую панель. Для этого перейдите на вкладку **Resource** и Меню **Insert**, далее **New Resource** и из списка выберите "диалоговая панель".



### Создаем код

Возьмем реакцию на двойной щелчок правой кнопкой мыши. И изменим процедуру реакции.

void CMainWnd::OnRButtonDblClk(UINT, CPoint)

{

CDialog about(IDD\_DIALOG1); // создаем объект класса диалогового окна

if (about.DoModal()==IDOK )

AfxMessageBox("Ok"); // пользователь выбрал OK

else

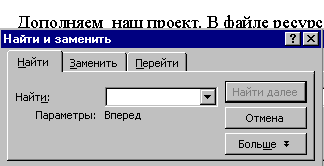
AfxMessageBox("Cancel"); // пользователь выбрал Canсel

}

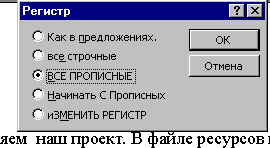
### Описание

Для работы с диалоговыми панелями в **MFC** есть класс **CDialog**. Этот класс необходим для создания модального и немодального диалогового окна.

На данный момент эти понятия нужно рассматривать как зависимое и независимое. То есть вывод модального окна на экран блокирует обращения к другим элементам приложения - меню, окна, кнопки и т.д. Немодальное окно позволяет обращаться к окну программы. Ниже на рисунке немодальное диалоговое окно поиска в **MS Word**.



И модальное окно в той же программе:



Мы создали свой статический класс **CDialog**. Он будет существовать только когда существует процедура. Вообще это понятие области видимости переменной. Если она объявлена на уровне процедуры, то живет только пока выполняется процедура. Правило, что после скобки **{** началось, то на скобке **}** и закончится. С мечом пришедши, от него погибнешь и т.п.

В параметр мы указали идентификатор окна из ресурса. По этому идентификатору и будет получена копия окна. Внимание - класс **CDialog** создает окно на основе ресурсов, беря идентификатор и строя по нему окно.

Функция **DoModal()** выводит окно в модальном режиме, то есть, блокируя главное окно программы. Она возвращает результат - код нажатой клавиши. Вот его мы и обрабатываем. По русски там написано так - Если в результате запуска окна нажата клавиша **ОК**, выведи сообщение **ОК**, иначе выведи сообщение **Cancel**.

Компилируем, запускаем, по двойному щелчку окно есть. Вы чувствуете, как растет наша компьютерная мускулатура?

### Шпаргалка

1. Создайте диалоговое окно в ресурсах.
2. На основе идентификатора создайте объект диалогового окна.

CDialog about(IDD\_DIALOG1);

1. Запустите просто:

about.DoModal();

Или с обработкой

if (about.DoModal()==IDOK )

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/9.zip) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?8) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?10) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc1.html)  
Автор **Каев Артем**.

## Шаг 10 - Рисуем в окне

### Создание проекта

Изменяем проект. Это очень хороший проект. На нем тренироваться и тренироваться...

### Создаем код

Нам необходимо переопределить виртуальную функцию прорисовки экрана. Она реализована в виде заглушки **afx\_msg OnPaint()**.

class CMainWnd : public CFrameWnd

{

public:

......

void MenuSwitch(); // Процедура реакции на выбор пункта меню

**afx\_msg void OnPaint(); // Переопределяем функцию рисования в окне**

~CMainWnd(); // Деструктор

private:

BOOL blMenu; // какое меню активно

......

};

Наша программа должна реагировать на события перерисовки экрана.

BEGIN\_MESSAGE\_MAP(CMainWnd, CFrameWnd) // таблица откликов на сообщения

......

ON\_WM\_KEYDOWN() // реакция на нажатие клавиши

ON\_WM\_CREATE() // событие создания окна

**ON\_WM\_PAINT() // Это реакция. Если нету этой строчки, бог вам судья.**

ON\_COMMAND(ID\_FILE\_EXIT,MenuExit) // Вот она, обработка реакции на выбор меню

......

END\_MESSAGE\_MAP()

А вот и сама функция:

void CMainWnd::OnPaint()

{

CPaintDC dc(this); // Получить контекст устройства

dc.TextOut(200,200,"Hello MFC Programm"); // Написать в нем привет

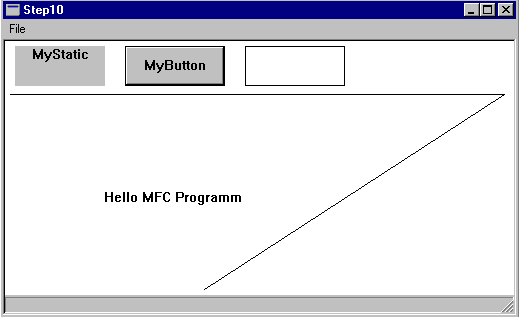
dc.MoveTo(300,300); // Начать рисовать линию

dc.LineTo(600,105); // И продолжить рисовать линию

dc.LineTo(105,105); // И еще продолжить рисовать линию

}

Вот такое получается окно программы.



### Описание

При программировании под **Windows** за работу с графикой отвечает **GDI Graphics Divice Interface**. Все функции работы с графикой сосредоточенны в **gdi32.dll**.

В программе не только нужно создать графику, нужен код. Этот код должен отвечать за время, место прорисовки. Главным моментом здесь является сообщение **WM\_PAINT**. Это сообщение посылается окну всегда, когда требуется перерисовка экрана. Например: при изменении размера окна, и если окно выходит на передний план, и при запуске программы естественно.

Внимание !!! **Windows** не хранит массивы пикселей для сохранения окон, как это часто делалось при программировании в **DOS**. При необходимости "освежить окно" - оно перерисовывается и делает это сообщение **WM\_PAINT**.

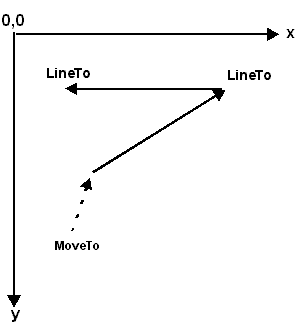
Для целей графики и близких к ним в **Windows** есть понятие контекст устройства. Эти контексты не зависят от самого устройства для программиста. Например, на мониторах разных типов команды одинаковы, при этом ВНИМАНИЕ команды рисования на мониторе и принтере тоже одинаковые. С **Windows** Вас, как программиста, не интересует какой язык использует принтер. А еще недавно я помню, как приходилось штудировать документацию по принтеру и писать драйвера к своим программам под каждый тип принтера.

Представьте, что вы установили у себя на компьютере новый монитор, и новый принтер, в программе не нужно править ни одного байта, всё будет работать. Кто теперь скажет, что **Windows** это плохо, а **Microsoft** не умеет писать программы ?

Первым делом в классе рамки окна мы описали функцию, которая вызывается в ответ на **WM\_PAINT**. Потом в обработчике событий указали, что на это событие реагировать. И написали сам код.

С помощью **CPaintDC dc(this);** получили контекст нашего окна для рисования. Вывели текст. Установили указатель в точку. Провели линию. Компилируем, запуcкаем на выполнение. Ну прям, как у Пикасо !

Давайте посмотрим отдельно, как работает **MoveTo()** и **LineTo()**.



Функция **MoveTo()** перемещает курсор к указанной точки без рисования. Функция **LineTo()** рисует линию от текущего положения до указанного и при этом перемещает курсор к указанному положению. То есть после того, как Вы нарисовали линию с помощью **LineTo()**, курсор оказывается в конечной точке.

### Шпаргалка

Опишите функцию рисования.

class CMainWnd : public CFrameWnd

{

......

afx\_msg void OnPaint();

......

}

Создайте реакцию на сообщение.

BEGIN\_MESSAGE\_MAP(CMainWnd, CFrameWnd)

......

ON\_WM\_PAINT()

......

END\_MESSAGE\_MAP()

Напишите код рисования.

void CMainWnd::OnPaint()

{

CPaintDC dc(this);

......

}

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/10.zip) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?9) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?11) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc1.html)  
Автор **Каев Артем**.

## Шаг 11 - Изучаем CPaintDC

### Создание проекта

Будем дальше работать с нашим проектом. Есть такая поговорка - родителей не выбирают. Но если папа миллионер это круто. Классу **CPaintDC** с папой очень повезло, папа **CDC** дал ему все, все что нужно, чтобы хорошо себя чувствовать в семье **MFC**.

### Создаем код

Мы будем мучить **OnPaint()**, этой функции не повезло, и мы её задергаем, и **Windows** будет обращаться к ней часто. Я буду приводить примеры кода, вы их по очереди вставляйте и смотрите результат.

Вывести строку:

void CMainWnd::OnPaint()

{

CPaintDC dc(this);

dc.TextOut( 200, 200,"TextOut Samples");

}

Вывести точку, смотрите внимательно, ну маленькая она и чем выше разрешение экрана, тем меньше.

void CMainWnd::OnPaint()

{

CPaintDC dc(this);

dc.SetPixel(500,200,RGB(255,0,0));

}

Дуга окружности:

void CMainWnd::OnPaint()

{

CPaintDC dc(this);

dc.Arc(200,200,100,100,400,400,10,10);

}

Замкнутая дуга:

void CMainWnd::OnPaint()

{

CPaintDC dc(this);

dc.Chord(250,250,100,100,400,400,10,10);

}

Эллипс:

void CMainWnd::OnPaint()

{

CPaintDC dc(this);

dc.Ellipse(450,450,50,150);

}

Линия:

void CMainWnd::OnPaint()

{

CPaintDC dc(this);

dc.MoveTo(200,200);

dc.LineTo(100,100);

}

### Описание

Функций очень много, это естественно маленькая часть. Но она показывает принцип, что примитивов очень много. И **CPaintDC** умеет их выводить на контекст устройства.

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?10) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?12) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc1.html)  
Автор **Каев Артем**.

## Шаг 12 - Учим рисовать CPaintDC

### Создание проекта

Наш проект жив, здоров, а чего ему будет. Хороший проект. Теперь мы попробуем создать свой **CPaintDC**, ведь он умеет рисовать далеко не все, крест например !

### Создаем код

Создаем свой класс от **CPaintDC**:

class CMyPaintDC: public CPaintDC

{

public:

CMyPaintDC(CWnd\* pWnd); // конструктор

void DrawCross(int iPosX, int iPosY); // функция рисования креста

};

И реализацию конечно:

CMyPaintDC::CMyPaintDC(CWnd\* pWnd)

:CPaintDC(pWnd) // мы дети CPainDC

{

}

void CMyPaintDC::DrawCross(int iPosX, int iPosY)// для создания своей фигуры используем

{ // функции CPaintDC

MoveTo(iPosX-50,iPosY);

LineTo(iPosX+50,iPosY);

MoveTo(iPosX,iPosY-50);

LineTo(iPosX,iPosY+50);

}

Пробуем:

void CMainWnd::OnPaint()

{

CMyPaintDC dc(this); // наш класс

CPen MyPen(PS\_DASHDOT,1,RGB(0,255,0)); // настраиваем перо

dc.SelectObject(MyPen); // выбираем перо

for (int x=1;x<400;x=x+50)

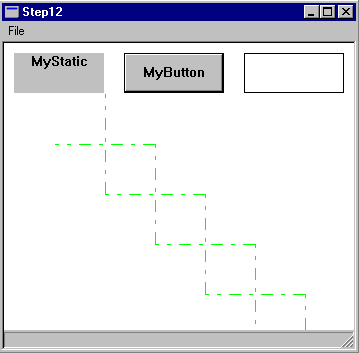
{

dc.DrawCross(100+x,100+x); // наша функция

}

}

Запускаем программу и смотрим результат.



### Описание

Примененный подход должен быть до боли знаком. Вспомним.

// кнопка

class CMyButton: public CButton

{

......

}

// рамка окна

class CMainWnd : public CFrameWnd

{

......

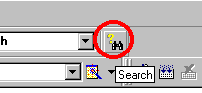
};

Обратите внимание на реализацию конструктора. Нам его не нужно переделывать, и поэтому сразу при создании вызываем конструктор папы и передаем ему параметры. Откуда я взял параметры в конструктор **CMyPaintDC**? От **CPaintDC** - написал, навел курсор и нажал **F1**. Хорошая кнопка помогает часто!

Позволю себе остановиться на поиске помощи.

**ВЕРСИЯ 5.0**  
Видите бинокль в проекте ? Нажмите на него левой кнопкой мышки. Появится диалоговое окно с вкладками **Index** и **Query**. В **Index** введите, ну там строка для ввода всего одна, наш **CPaintDC** и внизу будут темы. Про конструктор это **CPaintDC::CPaintDC**, щелкните мышкой по нему. Оттуда параметры я и взял. А вот в **Query** вы найдете все страницы **HTML** помощи где упоминается введенное вами слово, после нажатия кнопки. Что за кнопка догадаетесь.

**ВЕРСИЯ 6.0**  
Действуем аналогично, нажимаем на бинокль. Запустится **MSDN Library**, в которой вы можете произвести поиск нужного раздела справки. Либо по указателю, а если там не найдете, то в поиске, который просмотрит все документы и найдет все места где данное слово находится.



Далее в методе **OnPaint()** рисуем. Но не просто, а создаем сначала перо. **CPen** - это перо. Мы все делаем с параметрами - тип, толщина, цвет. **PS\_DASHDOT** - это пунктир. И методом **SelectObject()** указываем, что рисовать именно им. Рисуем кресты по диагонали.

Запускайте, компилируйте. Кресты, зеленые, по диагонали. Круто! Красиво!

### Шпаргалка

Создайте наследника от **CPaindDC**:

class CMyPaintDC: public CPaintDC

......

Создайте свои функции рисования:

class CMyPaintDC: public CPaintDC

{

......

void DrawCross(int iPosX, int iPosY);

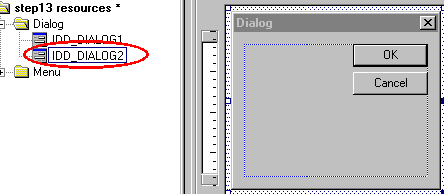
}

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/12.zip) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?11) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?13) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc1.html)  
Автор **Каев Артем**.

## Шаг 13 - Обмен данными с диалоговой панелью без DDX

### Создание проекта

Дополняем наш проект. В файле ресурсов необходимо создать еще одну простую диалоговую панель. Можете использовать старую, но не запутайтесь с идентификаторами !



### Создаем код

Нам нужно объявить идентификатор списка:

......

#define IDC\_MYEDIT 102 // Идентификатор поля редактирования

**#define IDC\_LIST1 103 // Идентификатор поля со списком**

......

Мы создадим наш класс **CMyDialog**. Описание класса:

class CMyDialog : public CDialog

{

public:

CMyDialog( UINT nIDTemplate,CWnd\* pParentWnd = NULL); // конструктор

~CMyDialog(); // деструктор

virtual BOOL OnInitDialog(); // инициализация

virtual void OnOK(); // реакция на нажатие OK

CString GetString(); // получить строку

private:

CListBox \*cListBox; // указатель на элемент управления

CString csListTextItem; // возвращаемое значение

};

Реализация:

CMyDialog::~CMyDialog()

{

if (cListBox!=NULL) delete cListBox; // если объект был создан удалить

}

CString CMyDialog::GetString()

{

return csListTextItem; // вернуть результат

}

void CMyDialog::OnOK() // когда нажмут кнопку OK

{

if (cListBox->GetCurSel()==LB\_ERR ) // если в списке ничего не выбрано

{

AfxMessageBox("Select Item List Box"); // сообщить о ошибке

}

else // иначе, то есть что-то выбрано

{

cListBox->GetText(cListBox->GetCurSel(),csListTextItem); // получить результат

CDialog::OnOK(); // вызвать функцию предка

}

}

CMyDialog::CMyDialog(UINT nIDTemplate, CWnd\* pParentWnd)

:CDialog(nIDTemplate,pParentWnd)

{

cListBox=NULL; // Объекта нет, NULL

}

BOOL CMyDialog::OnInitDialog()

{

CDialog::OnInitDialog(); // метод предка

CenterWindow(); // по центру окна

CRect rect(10, 10, 100, 100); // координаты для списка

cListBox=new CListBox(); // создать объект списка

cListBox->Create(WS\_CHILD|WS\_VISIBLE|WS\_TABSTOP|LBS\_NOTIFY|WS\_VSCROLL|WS\_BORDER,rect,

this, IDC\_LIST1); // и сам список связав его с идентификатором

if (cListBox!=NULL)

{

cListBox->AddString("God"); // добавить строку

cListBox->AddString("Bad"); // еще

cListBox->AddString("VOID"); // еще

return TRUE; // сказать что все удачно.

}

else return FALSE; // не смогли создать список

}

И применение:

void CMainWnd::OnLButtonDblClk( UINT, CPoint)

{

CMyDialog about(IDD\_DIALOG2); // создать объект нашего класса

if (about.DoModal()==IDOK) // если пользователь выбрал OK

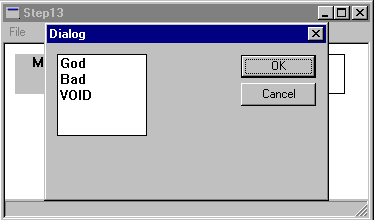
{

AfxMessageBox(about.GetString()); // показать результат

}

}

Обратите внимание, что вызывается панель по двойному щелчку левой кнопкой. Вот так будет выглядеть программа после вызова диалоговой панели.



### Описание

Смотрите, в объявление класса **CMyDialog** добавилась функция **OnInitDialog()**. Это аналог функции **OnCreate()**, т.е. в ней описываются все элементы создания диалога. Функция **OnOk()** виртуальная и будет вызвана по нажатию кнопки **OK**. **GetString()** для возврата данных, мы ведь договорились своими методами, правда? **CListBox\*** - указатель на будущий **ListBox** (окно со списком). **csListTextItem** - cтрока, которую вернет функция **GetString()**.

Функция возврат строки **GetString()**. Там написано **return csListTextItem** - это понятно, вернуть строку.

Функция **OnOK()**. Когда пользователь нажмет кнопку **Ok**, управление передается в эту функцию. Дальше она проверяет, выбрали ли вы слово из списка используя константу **LB\_ERR**. Эта константа возвращается когда нет выбора в поле со списком. Так, если выбора нет, тогда сообщается об этом и продолжить диалог. А вот если выбор был, с помощью функции **GetText()** получаем текст. В ней необходимо указать, куда будет возвращаться строка. Индекс мы берем текущий - **GetCurSel()** скажет, что выбрано(номер строки). И напоследок вызываем **OnOk** папы, он завершит диалог и вернёт результат.

В методе **OnInitDialog()** вызываем одноименный метод папы. Пусть он всё необходимое сделает. **CenterWindow()** поставит диалоговое окно по середине от текущего окна, это красиво. Создаем объект типа **CRect** для будущего списка. Создаем объект списка используя **new**, и окно списка используя **Create()**. Параметры - атрибуты списка, размер, куда вставлять и идентификатор. **Addstring()** добавляет строки в список.

И в конце изменяем реакцию на двойной щелчок левой кнопкой мыши. Вызываем панель, и в случае нажатия кнопки **Ok** показываем, что выбрано.

Запускайте и пробуйте, получится точно !

### Шпаргалка

Создайте свой класс диалогового окна, переменные для данных, переменные для элементов управления.

class CMyDialog : public CDialog

{

......

private:

CListBox \*cListBox;

CString csListTextItem;

}

Опишите идентификаторы будущих элементов:

......

#define IDC\_LIST1 103 // Идентификатор поля со списком

......

Перегрузите **OnInitDialog()**, создав в нем Ваши элементы управления:

BOOL CMyDialog::OnInitDialog()

{

......

CRect rect(10, 10, 100, 100);

cListBox=new CListBox();

cListBox->Create(WS\_CHILD|WS\_VISIBLE|WS\_TABSTOP|LBS\_NOTIFY|

WS\_VSCROLL|WS\_BORDER,rect,this, IDC\_LIST1);

......

}

Создайте функции возврата данных:

class CMyDialog : public CDialog

{

......

CString GetString();

......

}

Перегрузите **OnOk()** для того, чтобы проверить, что пользователь ввел данные и вернуть данные из элементов управления в переменные.

void CMyDialog::OnOK()

{

......

cListBox->GetText(cListBox->GetCurSel(),csListTextItem);

CDialog::OnOK();

......

}

После завершения работы диалогового окна получите Ваши данные:

void CMainWnd::OnLButtonDblClk( UINT, CPoint)

{

......

about.DoModal();

AfxMessageBox(about.GetString());

......

}

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/13.zip) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?12) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?14) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc1.html)  
Автор **Каев Артем**.

## Шаг 14 - Об отладке, размере файлов и т.д.

### Создание проекта

Создайте любой проект **Win32 Console**, например с именем **step14**. Не забудьте установить поддержку **MFC**, как в ["Шаг 1 - Минимальный проект"](http://www.firststeps.ru/mfc/steps/r.php?1).

### Создаем код

#include "stdafx.h"

#include "iostream.h"

#include "afxwin.h" // библиотека MFC

#include "conio.h"

void main()

{

cout << "This not debug" << endl;

#ifdef \_DEBUG

cout << "debug code" << endl;

#endif

getch(); // дождаться нажатия клавиши

}

### Описание

Запускайте, выполняйте. На экране должна появиться надпись.

This not debug

debug code

Это говорит о том, что у нас включена отладка в код. В нашем **EXE** есть имена переменных, классов, функций, и весь код. Плюс к тому в нем есть код включенный через детективы компилятора **\_DEBUG**. В **MFC** такого отладочного кода девать некуда. Честное слово. Посмотрите исподники **MFC** в каталоге:

Версия 5.0

Program Files\DevStudio\Vc\mfc\src

Версия 6.0

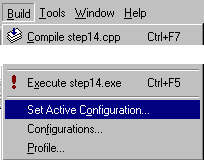
Program Files\Microsoft Visual Studio\VC98\MFC\SRC

А теперь посмотрим размер вашего файла. В каталоге проекта есть папка **Debug**, где лежит Ваш файл и его размер где-то около...

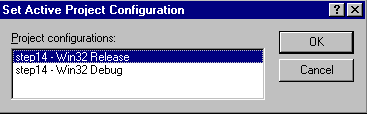
Версия 5.0 - 900 kb

Версия 6.0 - 1.25 mb

Как мне сказал один хороший знакомый - валерианки не хватит. Для того, чтобы получить нормальную версию, необходимо переключиться на проект **Release**. Это делается в меню **Build** далее **Set Active Configuration**:



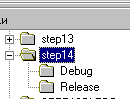
Дальше надо выбрать **Release**:



Запустите сборку **Rebuild all** и проект на выполнение. На экране появится.

This not debug

В папочке проекта у вас появится папочка **Release**. Посмотрите там размер **EXE** файла всего около **100 КБ**. Как впечатляет ?



Ещё размер зависит от того, как вы используете **MFC**. Есть два варианта **Static Library** и **Dinamic Library**. Если вы используете первый вариант, то код **MFC** будет в проекте и размер вырастет. Во втором варианте программа будет пользоваться **DLL** и размер будет меньше.

Так вот второй вариант, это потенциальные проблемы, вам надо следить при установке, чтобы все **DLL** были в системе. Кто гарантирует, что их потом не заменят или не удалят? Я помню на заре развития программирования на **VB** появились утилиты, которые прикрепляют **DLL** к **EXE**, они появились после горького опыта многих людей.

У меня пользователи не могут даже заставку на экране поменять. При таком подходе можно применять динамическую компоновку. А если ваша программа будет работать в экстремальных ситуациях и на большом удалении от Вас, оставляйте статическую компоновку. Жесткие диски по **10 GB** и больше уже норма. Ничего с ними не станет. Сотрут игры :) !!!

### Шпаргалка

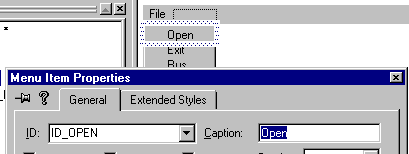
Завершая разработку программы, сделайте версию компоновки **Release**.

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/14.zip) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?13) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?15) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc1.html)  
Автор **Каев Артем**.

## Шаг 15 - Стандартный диалог открытия файлов

### Создание проекта

Продолжаем развивать проект остановленный на шаге ["Шаг 13 - Обмен данными с диалоговой панелью без DDX"](http://www.firststeps.ru/mfc/steps/r.php?13). Нам необходимо создать пункт меню **Open** в меню **File** с идентификатором **ID\_OPEN**.



### Создаем код

Реакция на событие выбора меню.

BEGIN\_MESSAGE\_MAP(CMainWnd, CFrameWnd) // таблица откликов на сообщения

......

ON\_WM\_PAINT()

ON\_COMMAND(ID\_OPEN,OnOpen) // Реакция на выбор меню Open

ON\_COMMAND(ID\_FILE\_EXIT,MenuExit) // Вот она, обработка реакции на выбор меню

......

END\_MESSAGE\_MAP()

Описание функции реакции:

class CMainWnd : public CFrameWnd

{

public:

CMainWnd(); // Конструктор по умолчанию

**void OnOpen(); // эта функция вызовется при выборе меню Open**

afx\_msg void OnLButtonDblClk(UINT, CPoint);

......

}

И реализация:

void CMainWnd::OnOpen()

{

CFileDialog fileDialog(TRUE,NULL,"\*.cpp"); //объект класса выбора файла

int result = fileDialog.DoModal(); //запустить диалоговое окно

if (result==IDOK) //если файл выбран

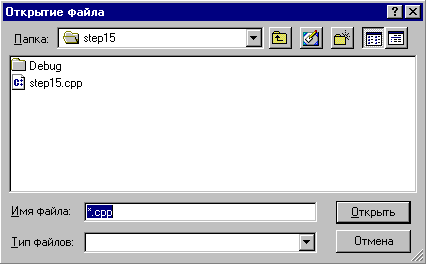
{

AfxMessageBox(fileDialog.GetPathName()); // показать полный путь

}

}

Вот такое диалоговое окно появится при выборе пункта меню **File -> Open**.



### Описание

В **MFC** есть стандартный класс для открытия и сохранения файлов **CFileDialog**. Смотрите, в функции **OnOpen()** объявляем класс, первый параметр простой **TRUE** - диалог открытия, **FALSE** диалог сохранения. Следующий параметр нам сейчас не нужен, он есть ниже. И расширения для открытия. Функция **DoModal()** вам знакома, если она возвращает код кнопки **OK**, то получаем имя файла функцией с названием, которое даже переводить не хочется.

Диалог сохранения файла:

CFileDialog fileDialog(FALSE,NULL,"\*.cpp");

Выбор нескольких файлов в диалоговом окне:

CFileDialog fileDialog(TRUE,NULL,"\*.\*",OFN\_ALLOWMULTISELECT);

int result = fileDialog.DoModal();

if (result==IDOK)

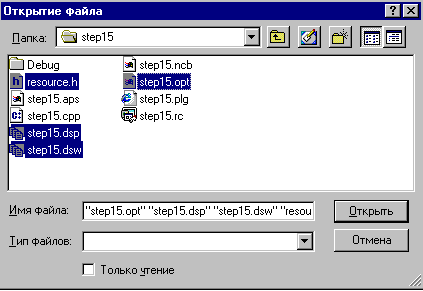
{

POSITION ps=fileDialog.GetStartPosition(); // получить начальную позицию

while (ps) AfxMessageBox(fileDialog.GetNextPathName(ps));// пока есть выбранные файлы

}

Так выглядит диалог в режиме выбора нескольких файлов.



При выборе нескольких файлов может возникнуть ситуация, что Вы не можете выбрать файлов больше некоторого количества. Выход в увеличении внутреннего буфера, вот так это можно сделать.

int iBufferSize = 100000;

CFileDialog fileDialog(TRUE,NULL,"\*.\*",OFN\_ALLOWMULTISELECT |

OFN\_NOVALIDATE ); // выбор нескольких файлов

fileDialog.m\_ofn.nMaxFile = iBufferSize; // устанавливаем размер буфера

char\* cNewBuffer = new char[iBufferSize]; // выделяем место

fileDialog.m\_ofn.lpstrFile = cNewBuffer; // даем указатель на новое место

fileDialog.m\_ofn.lpstrFile[0] = NULL; // ничего нет

int result = fileDialog.DoModal();

if (result==IDOK)

{

POSITION ps=fileDialog.GetStartPosition(); // получить начальную позицию

while (ps) AfxMessageBox(fileDialog.GetNextPathName(ps));// пока есть выбранные файлы

}

delete []cNewBuffer;// освобождаем выделенную память

После расширения можно указывать параметры окна диалога, вот я и указал, выбирай несколько файлов. А вот дальше мы должны получить позицию в переменную типа **POSITION**, эта переменная передается как указатель в функцию **GetNextPathName()**, а она выдает строку по этой позиции, и заодно уменьшает позицию. Когда файлов нет позиция станет равной **0**.

Фильтр на несколько файлов:

CFileDialog fileDialog(TRUE,NULL,"\*.mak;\*.rc");

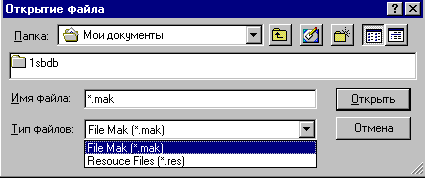
Как видите, через точку с запятой можно передавать несколько расширений. Это не **MFC** придумало естественно. В **Windows** функция открытия файлов уже есть, и эта возможность взята из **Win32** описания.

Выбор типа. За типом указан фильтр. Расширения для открытия ограничены скобками **|\*.mak|**.

CFileDialog fileDialog(TRUE,NULL,"\*.mak",NULL,"File Mak (\*.mak)|\*.mak|Resouce Files (\*.res)|\*.res|");

int result = fileDialog.DoModal();

Вот так это выглядит:



Добавление расширения для сохранения файлов. Как видите теперь установлена 2-я переменная. Это расширение, которое будет добавляться в конец файла.

CFileDialog fileDialog(FALSE,"obj","\*.obj");

### Шпаргалка

Создайте класс и настройте параметры:

CFileDialog fileDialog(TRUE,NULL,"\*.cpp");

Запустите окно диалога:

int result = fileDialog.DoModal();

Получите имя файла:

if (result==IDOK)

{

AfxMessageBox(fileDialog.GetPathName()); // показать полный путь

......

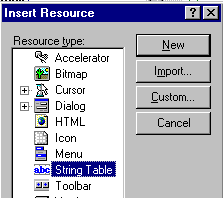
}

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/15.zip) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?14) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?16) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc1.html)  
Автор **Каев Артем**.

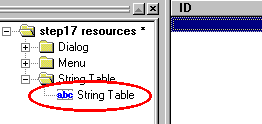
## Шаг 17 - Вывод данных в панель состояния

### Создание проекта

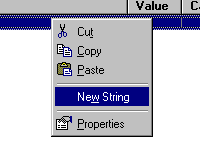
Дополняем наш проект. В нашем проекте необходимо создать ресурс содержащий две строки. Перейти на вкладку **ResourceView** и выбираем меню **Insert**, появится диалоговое окно, в нем нам нужно выбрать тип ресурса - **String Table**.



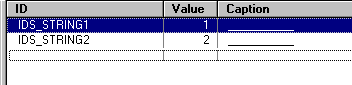
Далее нажать на кнопку **NEW** и у Вас появиться новый ресурс. Он называется **String Table** и используется для хранения текстовых строк.



В него нужно добавить две строки. В него нужно добавить две строки. Щелкаем в правом окне и выбираем пункт меню **New String**.



Вы должны создать две строки. Обратите внимание на то, что я специально сделал много символов подчеркивания. Это не просто так. На основе количества символов будет резервироваться место в стоке состояния.



Не забудьте сохранить. Посмотрите в ресурсах, теперь у нас есть идентификаторы строк.

// Used by step17.rc

//

#define IDS\_STRING1 1

#define IDS\_STRING2 2

#define IDR\_MENU 101

#define IDR\_MENU\_RUS 102

......

### Создаем код

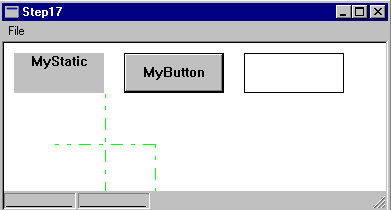
Нам нужен массив в котором будут находиться наши идентификаторы строк.

static UINT indicators[] =  
{  
 IDS\_STRING2, // идентификатор второй строки в ресурсах  
 IDS\_STRING1 // идентификатор первой строки в ресурсах  
};

Изменяем функцию **OnCreate** для того что бы установить индикаторы.

int CMainWnd::OnCreate(LPCREATESTRUCT lpCreateStruct)  
 {  
 if (CFrameWnd::OnCreate(lpCreateStruct) == -1) return -1;  
 m\_wndStatusBar.Create(this); // создать строку состояния  
 m\_wndStatusBar.SetIndicators(indicators,sizeof(indicators)/sizeof(UINT)); // установить в строку идентификаторы  
 m\_wndMenu.LoadMenu(IDR\_MENU); // Загрузить меню из файла ресурса  
 SetMenu(&m\_wndMenu); // Установить меню  
 return 0;  
}

Уже сейчас запустив программу у Вас есть возможность посмотреть на них.



В таблицу откликов реакцию на движение мыши.

BEGIN\_MESSAGE\_MAP(CMainWnd, CFrameWnd) // таблица откликов на сообщения  
 ........  
 ON\_WM\_CREATE() // событие создания окна  
 **ON\_WM\_MOUSEMOVE() // мышь двигается**  
 ON\_WM\_PAINT() // Это реакция. Если нету этой строчки ,  
 ............  
END\_MESSAGE\_MAP()

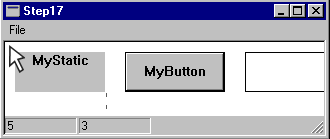
И описание соответствующей функции в классе рамки окна.

class CMainWnd : public CFrameWnd  
{  
public:  
 ...........  
 void OnOpen(); // эта функция вызовется при выборе меню Open  
 **afx\_msg void OnMouseMove( UINT, CPoint cp); // эта функция будет вызываться при движении мыши**  
 afx\_msg void OnLButtonDblClk( UINT, CPoint ); // виртуальная процедура ответа на левую кнопку  
 ...........  
};

Реализация нужна, таковы правила.

void CMainWnd::OnMouseMove( UINT, CPoint cp)  
{  
 char chX[10]; // буфер для координат  
 char chY[10]; // буфер для координат  
 itoa(cp.x,chX,10); // число переводим в строку  
 itoa(cp.y,chY,10); // число переводим в строку  
 CString csStatusX(chX); // Формируем строку  
 CString csStatusY(chY); // Формируем строку  
 m\_wndStatusBar.SetPaneText(0,csStatusX); // выводим первую панель  
 m\_wndStatusBar.SetPaneText(1,csStatusY); // выводим вторую панель   
}

Собрав и запустив проект Вы должны увидеть что при перемещении курсора его координаты показываться в строке состояния.



### Описание

Для отображения в панели состояния необходимо создать массив идентификаторов. Это необходимо для начала работы панели, а массив служит как шаблон. При запуске будет выведено именно то, что есть в строках, которые в ресурсах.

Функция **SetIndicators** говорит панели состояния что у нее будут наши панельки. В параметры, массив идентификаторов и количество элементов.

BOOL SetIndicators( const UINT\* lpIDArray, int nIDCount );  
// lpIDArray указатель на массив  
// nIDCount количество элементов в массиве

Мы с Вами вычисляем этот размер из массива. Это сделано для того, что бы в последствии добавляя новые панели, не приходилось заглядывать и сюда.

sizeof(indicators)/sizeof(UINT); // размер массива/размер одного элемента

В принципе если на этом остановиться то надписи выведутся. Но нам интересно, если будет показываться положение курсора. Ну, мне интересно, по крайней мере. А для этого необходимо отслеживать движение мыши. **ON\_WM\_MOUSEMOVE()** это делает, и функцию нужную вызывает **OnMouseMove**. Этой функции передается положение мыши в виде **CPoint**. Дальше мы обрабатываем. **itoa** перевод числа в строку , собирается обьект **CString** и **SetPaneText** с параметрами куда и что выводится в строку состояния.

BOOL SetPaneText( int nIndex, LPCTSTR lpszNewText, BOOL bUpdate = TRUE );  
//nIndex номер панели для вывода  
//lpszNewText текст для вывода  
//bUpdate обновлять

Запускайте, водите мышкой, наслаждайтесь.

### Шпаргалка

Создать массив идентификаторов.

static UINT indicators[] =  
{  
 IDS\_STRING2, // идентификатор второй строки в ресурсах  
 IDS\_STRING1 // идентификатор первой строки в ресурсах  
};

Связать массив с классом строки состояния.

m\_wndStatusBar.SetIndicators(indicators,sizeof(indicators)/sizeof(UINT)); // установить в строку идентификаторы

Вывод в строку состояния.

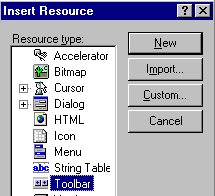
m\_wndStatusBar.SetPaneText(0,csStatusX); // выводим первую панель

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/17.zip) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?16) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?18) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc1.html)  
Автор **Каев Артем**.

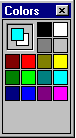
## Шаг 18 - Панель инструментов

### Создание проекта

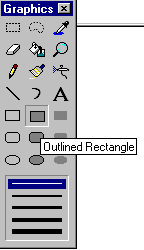
Дополняем наш проект. В нашем проекте необходимо создать ресурс панели инструментов. Перейти на вкладку **ResourceView** и выбираем меню **Insert**, появится диалоговое окно, в нем нам нужно выбрать тип ресурса - **ToolBar**.



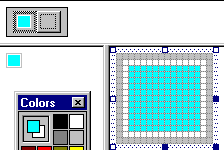
Теперь нужно две кнопки. Делается это так. Сначала выбираете цвет которым будете рисовать в окошке **Colors**.



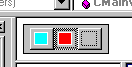
Потом инструмент, который хотите рисовать в окне **Graphics**. Выберите закрашенный прямоугольник.



И теперь нарисуйте на кнопки прямоугольник обратите внимание что сразу проявится новая пустая кнопка.



Вам нужно две кнопки.



Смотрим файл **resource.h** там появились идентификаторы наших элементов.

// Used by step18.rc  
//  
#define IDS\_STRING1 1  
#define IDS\_STRING2 2  
#define IDR\_MENU 101  
#define IDR\_MENU\_RUS 102  
#define IDD\_DIALOG1 103  
#define IDD\_DIALOG2 104  
**#define IDR\_TOOLBAR1 105**  
#define ID\_FILE\_EXIT 40001  
#define ID\_FILE\_SWITCH 40002  
#define ID\_OPEN 40003  
**#define ID\_BUTTON40004 40004  
#define ID\_BUTTON40005 40005**

### Создаем код

Добавляем в класс рамки окна класс панели инструментов.

class CMainWnd : public CFrameWnd  
{  
public:  
 .................  
 CMenu m\_wndMenu; // Это наш класс Меню  
 **CToolBar my\_ToolBar; // Это наш класс панели инструментов**   
 CMyClip clip; // Это класс буфера обмена  
 ............  
};

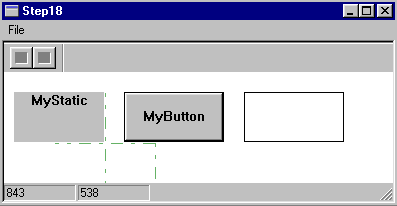
В функции **OnCreate CMainWnd** код для создания панели инструментов.

int CMainWnd::OnCreate(LPCREATESTRUCT lpCreateStruct)  
 {  
 ........  
 my\_ToolBar.Create(this); // создать панель интструментов  
 my\_ToolBar.LoadToolBar(IDR\_TOOLBAR1); // взять ее из ресурсов  
 my\_ToolBar.SetBarStyle(my\_ToolBar.GetBarStyle() | CBRS\_SIZE\_DYNAMIC); // настроить стили  
 my\_ToolBar.EnableDocking(CBRS\_ALIGN\_ANY); // можно ли панели швартовка  
 EnableDocking(CBRS\_ALIGN\_ANY); // функции окна для управления панелью  
 DockControlBar(&my\_ToolBar);   
 return 0;  
}

Перенесем элементы управления ниже .

CMainWnd::CMainWnd()  
{  
 blMenu=FALSE; // инициализация TRUE соответствует IDR\_MENU  
 Create(NULL,"Step18",WS\_OVERLAPPEDWINDOW,rectDefault,  
 NULL,NULL); // Создать окно программы  
 // оператор new по умолчанию в случае ошибки вернет NULL  
 // проверка указателя на NULL дает возможность избавиться от дальнейших ошибок  
 MyStatic = new CStatic();   
 if (MyStatic!=NULL) MyStatic->Create("MyStatic",WS\_CHILD|WS\_VISIBLE|SS\_CENTER,  
 CRect(10,50,100,100),this); // создали   
 MyButton = new CMyButton(); // Меняем класс на основе которого создается объект  
  
 if (MyButton!=NULL) MyButton->Create("MyButton",WS\_CHILD|WS\_VISIBLE|SS\_CENTER,  
 CRect(120,50,220,100),this,IDC\_MYBUTTON);   
 MyEdit = new CEdit();  
 if (MyEdit!=NULL) MyEdit->Create(WS\_CHILD|WS\_VISIBLE|WS\_BORDER,  
 CRect(240,50,340,100),this,IDC\_MYEDIT);  
}

Вот теперь можно запускать и пробовать.



### Описание

Создаем класс панели инструментов в классе рамки окна. Как и все элементы, панель инструментов необходимо создать. Для её создания мы выбрали место OnCreate, на мой взгляд это не самое плохое место. При создании требуется указать окно владельца. Естественно в нашем случае рамка окна, **this** сам за себя говорит. Далее мы читаем панель из ресурсов. Это так знакомо !

//для меню  
 m\_wndMenu.LoadMenu(IDR\_MENU);  
 SetMenu(&m\_wndMenu);  
 // для диалоговой панели  
 CDialog about(IDD\_DIALOG1);  
 about.DoModal();

**SetBarStyle** Мы унаследовали от папы **CControlBar**, к текущим стилям добавляем динамический размер. Следующей командой разрешаем её причаливание. **CBRS\_ALIGN\_ANY** - причаливать в любое место, а могли и не давать такой свободы. И необходимо разрешить "причалу" принимать панель. И в конце силовое причаливание, для начала. Запускайте, после всех необходимых процедур. Потаскайте панель, причальте туда, сюда. Работает !!! Как у Всех !!! Естественно необходимо проверять, создалась панель или нет, пpочиталась или нет. Не Забывайте.

### Шпаргалка

Обьявить класс панели.

class CMainWnd : public CFrameWnd  
{  
public:  
 .................  
 CToolBar my\_ToolBar; // Это наш класс панели инструментов   
 .................  
};

Создать и прочитать из ресурсов.

my\_ToolBar.Create(this); // создать панель интструментов  
 my\_ToolBar.LoadToolBar(IDR\_TOOLBAR1); // взять ее из ресурсов

Стиль и тип причала.

my\_ToolBar.SetBarStyle(my\_ToolBar.GetBarStyle() | CBRS\_SIZE\_DYNAMIC); // настроить стили  
 my\_ToolBar.EnableDocking(CBRS\_ALIGN\_ANY); // можно ли панели швартовка

Для рамки окна разрешить принимать суда и панель завести в порт.

EnableDocking(CBRS\_ALIGN\_ANY); // функции окна для управления панелью  
 DockControlBar(&my\_ToolBar);

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/18.zip) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?17) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?19) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc1.html)  
Автор **Каев Артем**.

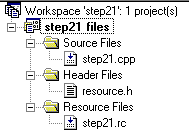
## Шаг 21 - Минимальный проект документ-вид

### Создание проекта

Лучше создать новый проект **Win32** с именем **step21**. В [шаге 1](http://www.firststeps.ru/mfc/steps/r.php?1) мы так делали. Создали проект с нуля. Не забудьте подключить использование **MFC**. Итак, **step21.cpp** у нас есть. **AppWizard** создает код, который интенсивно обращается к ресурсам. Обращение к ним происходит на основе имен. И если мы его эмулируем, то должны пользоваться теми же именами.

Теперь нам нужно добавить файл ресурсов с именем **step21.rc**. Смотрите [шаг 6](http://www.firststeps.ru/mfc/steps/r.php?6), если забыли. Сразу после создания файла ресурсов в папочке проекта появиться **resource.h**. Включите его через **Add File to Project** в наш проект.

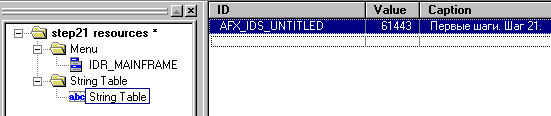
После всех этих магических действий три файла должны быть у нас в проекте.



В ресурсах нужно создать меню с идентификатором **IDR\_MAINFRAME**. Именно с этим именем так как классы **MFC** будут обращаться к этому ресурсу меню, именно к этому.



Еще нам нужен строковый ресурс **AFX\_IDS\_UNTITLED**. Если забыли, как добавлять смотрите [Шаг 17](http://www.firststeps.ru/mfc/steps/r.php?17).



Этот ресурс используется для задания имени приложения в левом верхнем углу. Если его не будет, то при выполнении вы получите сообщение об ошибке. В реализации **MFC** есть код.

void CSingleDocTemplate::SetDefaultTitle(CDocument\* pDocument)  
{  
 CString strDocName;  
 if (!GetDocString(strDocName, CDocTemplate::docName) ||   
 strDocName.IsEmpty())  
 {  
  **VERIFY(strDocName.LoadString(AFX\_IDS\_UNTITLED));** // Прочитать строку из ресурсов, успешность чтения проверить  
 }  
 pDocument->SetTitle(strDocName);  
}

Это проверка установки имени документа по умолчанию на основе строкового ресурса **AFX\_IDS\_UNTITLED**.

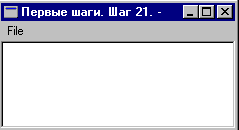
При создании рамки окна проверяется наличие ресурса с именем **IDR\_MAINFRAME**.

CFrameWnd\* CDocTemplate::CreateNewFrame(CDocument\* pDoc, CFrameWnd\* pOther)  
{  
 if (pDoc != NULL)  
 ASSERT\_VALID(pDoc);  
 ASSERT(m\_nIDResource != 0);// Здесь передается число и   
 //если этого ресурса нет то ошибка  
 .............  
}

### Создаем код

#include "afxwin.h" // это для MFC   
#include "resource.h" // идентификаторы ресурсов   
class CMyView : public CView // класс просмотра   
{  
protected:   
 CMyView();   
 DECLARE\_DYNCREATE(CMyView) // Эта функция вызывается для прорисовки вида   
public:  
 virtual void OnDraw(CDC\* pDC);  
};  
void CMyView::OnDraw(CDC\* pDC)   
{  
}  
IMPLEMENT\_DYNCREATE(CMyView, CView)  
CMyView::CMyView()   
{  
}  
class CMainFrame : public CFrameWnd // класс рамки окна   
{  
protected:   
 CMainFrame();   
 DECLARE\_DYNCREATE(CMainFrame)   
public:  
};  
IMPLEMENT\_DYNCREATE(CMainFrame, CFrameWnd)  
CMainFrame::CMainFrame()   
{   
}  
class CMyDoc : public CDocument // класс документа   
{  
protected:   
 CMyDoc();   
 DECLARE\_DYNCREATE(CMyDoc)   
public:  
};  
IMPLEMENT\_DYNCREATE(CMyDoc, CDocument)  
CMyDoc::CMyDoc()   
{  
}  
class CStep2App : public CWinApp // класс приложения   
{  
public:  
 CStep2App();  
 virtual BOOL InitInstance();  
};  
BOOL CStep2App::InitInstance()  
{  
 CSingleDocTemplate\* pDocTemplate; // новый шаблон   
 pDocTemplate = new CSingleDocTemplate(  
 IDR\_MAINFRAME, // идентификатор меню  
 RUNTIME\_CLASS(CMyDoc), // документ----|  
 RUNTIME\_CLASS(CMainFrame), // рамка окна--| Документ вид однако.  
 RUNTIME\_CLASS(CMyView)); // просмотр----|  
 AddDocTemplate(pDocTemplate); // добавить шаблон  
 CCommandLineInfo cmdInfo; // Класс команд  
 ParseCommandLine(cmdInfo); // разбор командной строки  
 if (!ProcessShellCommand(cmdInfo)) // запуск команд на выполнение  
 return FALSE;  
 m\_pMainWnd->ShowWindow(SW\_SHOW); // показать окно  
 m\_pMainWnd->UpdateWindow(); // запустить цикл обработки сообщений  
 return TRUE;  
}  
CStep2App::CStep2App()  
{  
}  
CStep2App theApp;

Вот так оно будет выглядеть после запуска.



### Описание

Идеология программирования в **MFC** заключается в концепции документ вид. Идея здесь в следующем. У вас есть класс, отвечающий за состояние памяти, им пользуются классы отвечающие за отображение. Например, у вас есть данные с координатами деталей, а вы хотите просматривать проекции. Можно создать несколько классов просмотров для проекций.

Для шаблона документа **CSingleDocTemplate** необходим идентификатор меню и указатели на три класса. Это шаблон для приложений **SDI**, т.е. с одним документом в окне. Класс просмотра **CView**. Класс документа **CDocument**. Класс рамки окна **CFrameWnd**. Созданный шаблон добавляется функцией **AddDocTemplate**, а вообще шаблонов может быть несколько[[1]](#endnote-1)!

Следующие функции запускают на выполнение ряд команд. Создать новый документ и т.д. Если что-то непонятно, это не страшно в следующих шагах положение изменится. Я считаю, что лучше раз испытать, чем слушать, и вероятно на примерах понятнее.

Мы с Вами создали код очень близкий к коду **AppWizard**. И к нему мы будем приближаться. Что будет происходить? При запуске будет сгенерирован шаблон на основе классов рамки, вида, документа. И при запуске процессор команд выполнит функцию Новый документ. После этого окно будет показано.

### Шпаргалка

Создать классы документа, вида, рамки окна.

class CMyDoc : public CDocument // класс документа  
...........  
class CMyView : public CView // класс просмотра  
...........  
class CMainFrame : public CFrameWnd // класс рамки окна

Создать и добавить шаблон документа.

pDocTemplate = new CSingleDocTemplate(  
 IDR\_MAINFRAME, // идентификатор меню  
 RUNTIME\_CLASS(CMyDoc), // документ----|  
 RUNTIME\_CLASS(CMainFrame), // рамка окна--| Документ вид однако.  
 RUNTIME\_CLASS(CMyView)); // просмотр----|  
 AddDocTemplate(pDocTemplate); // добавить шаблон

Запустить интерпретатор команд для создания документа.

if (!ProcessShellCommand(cmdInfo)) // запуск команд на выполнение  
 return FALSE;

Не забыть про ресурсы.

AFX\_IDS\_UNTITLED  
IDR\_MAINFRAME

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/21.zip) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?20) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?22) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc1.html)  
Автор **Каев Артем**.

## Шаг 22 - Организуем связь Документ-Вид

" Войди в меня , войди в мои сны  
 Войди в мои сны они так важны "  
 Ирина Алегрова  
 К Документ - Вид подходит очень :-).

### Создание проекта

Будем работать с проектом, созданным в прошлом шаге.

### Создаем код

Добавим функцию получения данных и сами данные к классу документа.

class CMyDoc : public CDocument // класс документа  
{  
protected:   
 CMyDoc();  
 **CString csDate; // здесь будем хранить**  
 DECLARE\_DYNCREATE(CMyDoc)  
public:  
 **CString GetDate(); // получить данные**  
};

Реализация.

CMyDoc::CMyDoc()  
{  
 csDate="Hello View - Doc"; // устанавливаем данные  
}  
CString CMyDoc::GetDate()  
{  
 return csDate; // выдаем  
}

В классе вида нужно создать функцию доступа к классу документа.

class CMyView : public CView // класс просмотр  
{  
protected:   
 CMyView(); **CMyDoc\* GetDocument(); // получить документ**  
 DECLARE\_DYNCREATE(CMyView) // Эта функция вызывается для прорисовки вида  
public:  
 virtual void OnDraw(CDC\* pDC);  
};

Реализуем ее.

CMyDoc\* CMyView::GetDocument()  
{  
 ASSERT(m\_pDocument->IsKindOf(RUNTIME\_CLASS(CMyDoc))); // проверить есть ли класс документа  
 return (CMyDoc\*)m\_pDocument; // вернуть результат.  
}

Вот сейчас проект не соберется. Вот такие ошибки будут.

Compiling...  
step22.cpp  
D:\VС\step22\step22.cpp(14) : error C2143: syntax error : missing ';' before '\*'  
D:\VС\step22\step22.cpp(14) : error C2501: 'CMyDoc' : missing storage-class or type specifiers

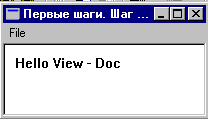
Это связанно с тем, что у нас класс документа объявлен ниже класса вида по тексту cpp файла. И класс вида просто не знает про него. Для решения этой проблемы мы просто перенесем описание класса документа выше описания класса вида.

#include "afxwin.h" // это для MFC  
#include "resource.h" // идентификаторы ресурсов  
class CMyDoc : public CDocument // класс документа  
{  
............  
};  
class CMyView : public CView // класс просмотр  
{  
.............  
};

Ну и изменим функцию рисования вида.

void CMyView::OnDraw(CDC\* pDC)  
{  
 CMyDoc\* tempDoc=GetDocument(); //получить класс документа  
 pDC->TextOut(10,10,tempDoc->GetDate()); //вывести строку на экран из документа  
}

Вот так выглядит наша программа в работе.



### Описание

В классе документа должны находится данные. Вот мы их и объявили в виде строки **CString csDate**. Так как данные должны быть защищены, то я их поместил в **private:**. Это соответствует классическим правилам объектно-ориентированного программирования. Доступ к данным только через функции. Для доступа создал функцию **CString GetDate()**. С ней я думаю просто, она возвращает строку.

Вид должен иметь возможность доступа к докумнту. Для этого создана функция **СMyDoc\* GetDocument();** в классе вида. Она возвращает указатель на класс документа. Именно указатель нам и нужен. **ASSERT(m\_pDocument->IsKindOf(RUNTIME\_CLASS(CMyDoc)));** - в общем по русски это так. Проверить принадлежность указателя **m\_pDocument** на принадлежность классу **CMyDOC IsKindOf(RUNTIME\_CLASS(CMyDoc))** и проверить, чтобы этот класс существовал **ASSERT(.....)**.Если все нормально вернуть указатель на класс документа - **return (CMyDoc\*)m\_pDocument;**

Мы изменили функцию прорисовки **OnDraw**. В этой функции мы получаем класс документа и выводим на экран строку.

Так вот, вникайте. Если вы захотите изменить вывод допустим шрифт букв, или место строки, то за всё это отвечает класс просмотра **CView**, а если необходимо изменить содержание строки то это делается в классе документа **CDocument**.

### Шпаргалка

В классе документа создай данные и доступ к ним.

class CMyDoc : public CDocument // класс документа  
{  
protected:   
.......  
 CString csDate; // здесь будем хранить  
...........   
public:  
 CString GetDate(); // получить данные  
};

В классе вида реализуй доступ к классу документа.

class CMyView : public CView // класс просмотр  
{  
 ........  
 CMyDoc\* GetDocument(); // получить документ  
 ..........  
};  
CMyDoc\* CMyView::GetDocument()  
{  
 ASSERT(m\_pDocument->IsKindOf(RUNTIME\_CLASS(CMyDoc))); // проверить есть ли класс документа  
 return (CMyDoc\*)m\_pDocument; // вернуть результат.  
}

Напиши код в функции рисования вида.

void CMyView::OnDraw(CDC\* pDC)  
{  
 CMyDoc\* tempDoc=GetDocument(); //получить класс документа  
 pDC->TextOut(10,10,tempDoc->GetDate()); //вывести строку на экран из документа  
}

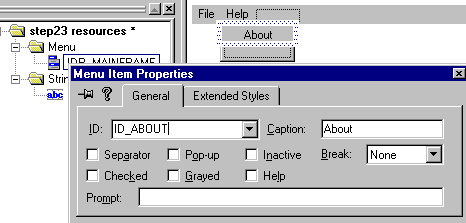
[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/22.zip) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?21) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?23) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc1.html)  
Автор **Каев Артем**.

## Шаг 23 - Взаимодействие документ-вид

Модель документ-вид - одна из наиболее существенных  
особенностей библиотеки MFC - пожалуй, наиболее трудна  
для понимания.  
Френк Крокет техническая поддержка Microsoft Visual C++.

### Создание проекта

Продолжаем развитие проекта. Нам надо только добавить к меню раздел **Help** и поле **About**. Для реакции на событие. Имя идентификатора для пункта меню **About - ID\_ABOUT**.



### Создаем код

Добавляем функции к описанию класса документа.

class CMyDoc : public CDocument // класс документа  
{  
protected:   
 CMyDoc();  
 CString csDate; // здесь будем хранить  
 DECLARE\_DYNCREATE(CMyDoc)  
public:  
 CString GetDate(); // получить данные  
 **void SetString(CString s); // установить строку  
 void OnAbout(); // реакция на выбор меню  
private:  
 DECLARE\_MESSAGE\_MAP(); // таблица откликов**  
};

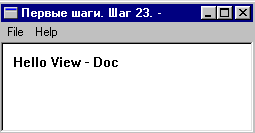
Нужно создать таблицу откликов на сообщения в классе документа.

BEGIN\_MESSAGE\_MAP(CMyDoc, CDocument)  
 ON\_COMMAND(ID\_ABOUT,OnAbout) // при выборе меню  
END\_MESSAGE\_MAP()

Реализация функций.

void CMyDoc::OnAbout()  
{  
 SetString("Первые шаги (с)"); // установить строку  
 UpdateAllViews(NULL); // обновить виды  
}  
void CMyDoc::SetString(CString s)  
{  
 csDate=s; // установить данные внутри документа  
}

Вот так будет реагировать программа на выбор пункта меню.



### Описание

Мы добавили функцию для изменения строки в классе документа **SetString**. И она изменяется после выбора пункта меню **OnAbout**. В **OnAbout** мы устанавливаем новую строку и вызываем **UpdateAllViews** эта функция заставляет виды перерисовать себя. А перерисовываются она на основе данных в документе. Им приходится вызывать свою функцию **OnDraw**, запрашивать документ и прорисовывать текст. В этом примере сама суть идеи документ - вид. Вы изменяете данные в документе, а виды их отражают.

### Шпаргалка

Меняйте данные в классе документа.

class CMyDoc : public CDocument // класс документа  
{  
 ...........  
 void SetString(CString s); // установить строку  
 ...........  
};

После изменения обновите виды.

void CMyDoc::OnAbout()  
{  
 ...........  
 UpdateAllViews(NULL); // обновить виды  
}

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/23.zip) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?22) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?24) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc1.html)  
Автор **Каев Артем**.

## Шаг 24 - Несколько видов для документа

### Создание проекта

Этот пример демонстрирует наиболее эффектную сторону технологии документ-вид. Будем дальше развивать проект.

### Создаем код

Добавляем описание для класса разделителя.

.................  
#include "afxwin.h" // это для MFC  
#include "afxext.h" // это для дополнительных возможностей  
#include "resource.h" // идентификаторы ресурсов  
class CMyDoc : public CDocument // класс документа  
...............

Создаем еще один класс вида.

class CNewView : public CView // еще один класс вида  
{  
protected:   
 CNewView();  
 DECLARE\_DYNCREATE(CNewView)  
public:  
 void OnDraw(CDC\* pDC);  
 CMyDoc\* GetDocument();  
};  
CMyDoc\* CNewView::GetDocument()  
{  
 ASSERT(m\_pDocument->IsKindOf(RUNTIME\_CLASS(CMyDoc)));  
 return (CMyDoc\*)m\_pDocument;  
}  
void CNewView::OnDraw(CDC\* pDC)  
{  
 CMyDoc\* tempDoc=GetDocument();  
 pDC->SetTextColor(RGB(255,0,0)); // установить красный цвет  
 pDC->TextOut(10,10,tempDoc->GetDate()); // вывести  
}  
IMPLEMENT\_DYNCREATE(CNewView, CView)  
CNewView::CNewView()  
{  
}

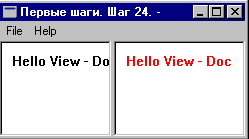
Добавляем описания класса и процедур для создания разделителя.

class CMainFrame : public CFrameWnd // класс рамки окна  
{  
protected: CMainFrame();  
 DECLARE\_DYNCREATE(CMainFrame)  
public:  
 **virtual BOOL OnCreateClient( LPCREATESTRUCT lpcs, CCreateContext\* pContext ); // создать клиента  
 CSplitterWnd m\_wndSplitter; // разделитель**  
};

Реализация.

BOOL CMainFrame::OnCreateClient( LPCREATESTRUCT lpcs, CCreateContext\* pContext )  
{  
 ASSERT(m\_wndSplitter.CreateStatic(this,1,2)); // создать разделитель  
 ASSERT(m\_wndSplitter.CreateView(0,0,RUNTIME\_CLASS(CMyView), // создать первый вид  
 CSize(100,50),pContext));  
 ASSERT(m\_wndSplitter.CreateView(0,1,RUNTIME\_CLASS(CNewView), // создать второй вид  
 CSize(100,50),pContext));  
 SetActiveView((CView\*)m\_wndSplitter.GetPane(0,1)); // установить активную панель  
 return TRUE;  
}

А вот и результат работы.



### Описание

Обратите внимание на то, что обработка сообщения от меню **About** находится в классе документа. Это гарантирует доступ к нему из разных видов. Класс документа у нас один. Обработка сообщений будет всегда в независимости от видов.

Описания класса разделителя нет в **afxwin.h**. Поэтому мы и подключили **afxext.h** где он есть. Логично? Создали второй класс просмотра, он совершенно аналогичен **CMyView** за исключением функции рисования. В ней текст выводится красным цветом. Для использования объявляем в классе рамки окна, объект класса разделитель **CSplitterWnd**.

**OnCreateClient** функция ответственна за создание клиента рамки окна, т.е. внутреннего наполнения рамки. Вот мы и устанавливаем наполнения, как создаем разделитель **m\_wndSplitter.CreateStatic(this,1,2)**, который вставляется в рамку (**this**) и состоит из двух частей 1,2. К каждой части привязываем просмотр **m\_wndSplitter.CreateView**, указывая класс просмотра, размер и т.д. В самом конце устанавливаем номер активной панели и сообщаем **MFC** о успешности нашей работы.

Две панели с текстом разного цвета. Выберите пункт меню **About** и текст поменяется сразу в двух панелях. А теперь фантазию надо применить. Где это используется. В **Excel** точно для просмотра частей таблицы, можно графику просматривать с разными масштабами и т.д.

### Шпаргалка

Включить дополнения.

#include "afxext.h" // это для дополнительных возможностей

Включить описание класса разделителя в класс рамки окна.

class CMainFrame : public CFrameWnd // класс рамки окна  
{  
 ..........  
 CSplitterWnd m\_wndSplitter; // разделитель  
};

Создать классы видов, сколько вы хотите.

Переопределить функцию **OnCreateClient**.

class CMainFrame : public CFrameWnd // класс рамки окна   
{  
 ............  
 virtual BOOL OnCreateClient( LPCREATESTRUCT lpcs, CCreateContext\* pContext ); // создать клиента  
};

И реализовать разделители.

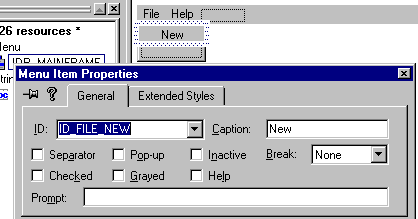
BOOL CMainFrame::OnCreateClient( LPCREATESTRUCT lpcs, CCreateContext\* pContext )  
{  
 ASSERT(m\_wndSplitter.CreateStatic(this,1,2)); // создать разделитель  
 ASSERT(m\_wndSplitter.CreateView(0,0,RUNTIME\_CLASS(CMyView), // создать первый вид  
 CSize(100,50),pContext));  
 ASSERT(m\_wndSplitter.CreateView(0,1,RUNTIME\_CLASS(CNewView), // создать второй вид  
 CSize(100,50),pContext));  
 SetActiveView((CView\*)m\_wndSplitter.GetPane(0,1)); // установить активную панель  
 return TRUE;  
}

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/24.zip) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?23) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?25) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc1.html)  
Автор **Каев Артем**.

## Шаг 26 - Пункт меню новый документ как в AppWizard

### Создание проекта

Работаем с текущим проектом. Только в нашем пункте меню **File** нужно создать пункт меню **New** с идентификатором **ID\_FILE\_NEW**. Хотя идентификатор может быть любой, но мы договорились сделать как в волшебнике.



### Создаем код

Добавляем таблицу откликов в класс приложения.

class CStep2App : public CWinApp // класс приложения  
{  
public:  
 CStep2App();  
 virtual BOOL InitInstance();  
private:  
 **DECLARE\_MESSAGE\_MAP() // таблица откликов для класса приложения**  
};

И её реализация.

BEGIN\_MESSAGE\_MAP(CStep2App, CWinApp)  
 **ON\_COMMAND(ID\_FILE\_NEW, CWinApp::OnFileNew) // пункт меню New**   
END\_MESSAGE\_MAP()

Добавляем функцию **новый документ** в класс документа.

class CMyDoc : public CDocument // класс документа  
{  
protected:   
 CMyDoc();  
 CString csDate; // здесь будем хранить  
 **virtual BOOL OnNewDocument(); // будет вызываться при выборе меню New**  
 DECLARE\_DYNCREATE(CMyDoc)  
public:  
 CString GetDate(); // получить данные  
 void SetString(CString s); // установить строку  
 void OnAbout(); // реакция на выбор меню  
private:  
 DECLARE\_MESSAGE\_MAP(); // таблица откликов  
};

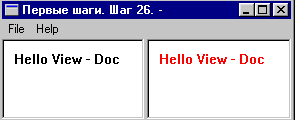
И реализация.

BOOL CMyDoc::OnNewDocument()  
{  
 if (!CDocument::OnNewDocument()) // Если не удалось создать документ  
 return FALSE; // Вернуть ошибку  
 csDate="Hello View - Doc"; // Изменить данные в документе  
 return TRUE;  
}

Изменить конструктор документа.

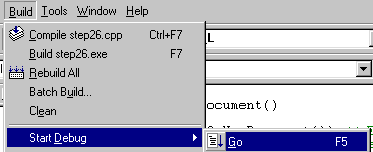
CMyDoc::CMyDoc()  
{  
 TRACE("CMyDoc::CMyDoc()\n"); // вывести в окно отладки  
}

Вот так приложение должно работать.



### Описание

Описание я начну с механизма реализации функции новый документ. В **MFC** заложены механизмы автоматизации пункта меню **New**. Конструктор документа вызывается только один раз при создании приложения. Больше он не вызывается. Это вы можете проверить, запустив приложение в режиме отладки **(F5 Go)**.



Запускайте, получите сообщение от конструктора. Выбирайте пункт меню **New** и ничего. В окне отладки будет только одна надпись о создании объекта документа.

Loaded 'C:\WINDOWS\SYSTEM\MFC42LOC.DLL', no matching symbolic information found.  
Warning: no document names in string for template #101.  
Warning: no document names in string for template #101.  
**CMyDoc::CMyDoc()**  
Loaded 'C:\WINDOWS\SYSTEM\INDICDLL.DLL', no matching symbolic information found.  
The thread 0xFFF89A49 has exited with code 0 (0x0).  
The program 'D:\VС\step26\Debug\step26.exe' has exited with code 0 (0x0).

Так вот, для создания нового документа используется виртуальная функция **OnNewDocument**. Именно поэтому я убрал из конструктора определение строки **csDate**. Все данные должны определяться в **OnNewDocument**.

Ну, теперь смотрим сверху. Первое добавили таблицу отклика в класс приложения. В этом классе **CWinApp** есть механизм вызова **NewDocument**. Вот мы его вызываем обращаясь к функции **CWinApp::OnFileNew**. Далее мы создаем функцию **OnNewDocument** для класса **CDocument** и её реализацию. Вызываем функцию отца для нормальной работы. Устанавливаем данные и говорим, новый документ создан удачно **return TRUE;**.

Запускайте. А теперь ходовые испытания. Первое. Вызываем меню **About** для изменения данных, если вы нечего не меняли, то должен появиться надпись Первые шаги. А теперь вызовем **New**, и в видах меняются надписи. Они вернутся к начальному состоянию.

Вот так, работает. Это классно. Классно потому что понятно. И если вы будете, а вы будете пользоваться волшебником обязательно, то я надеюсь поймете в чем дело.

### Шпаргалка

Создать пункт меню с идентификатором.

ID\_FILE\_NEW

Создать реакцию на него в классе приложения.

BEGIN\_MESSAGE\_MAP(CStep2App, CWinApp)  
 **ON\_COMMAND(ID\_FILE\_NEW, CWinApp::OnFileNew) // пункт меню New**   
END\_MESSAGE\_MAP()

Создать функцию новый документ.

BOOL CMyDoc::OnNewDocument()  
{  
 if (!CDocument::OnNewDocument()) // Если не удалось создать документ  
 return FALSE; // Вернуть ошибку  
 csDate="Hello View - Doc"; // Изменить данные в документе  
 return TRUE;  
}

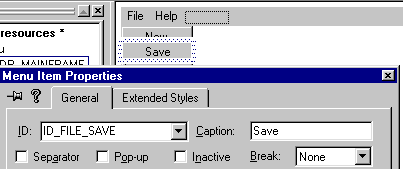
[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/26.zip) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?25) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?27) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc1.html)  
Автор **Каев Артем**.

## Шаг 27 - Пункт меню сохранить документ как в AppWizard

### Создание проекта

Мы будем учить наш проект сохранять данные. Поэтому оставим старый. Но в ресурсах надо добавить идентификаторы и пункт меню **Save**.

Итак, идем в ресурсы и в меню создаем пункт меню **Save**, а идентификатор ставим **ID\_FILE\_SAVE**.



Теперь нам нужно в ресурсах создать еще одну строку. С идентификатором **AFX\_IDS\_ALLFILTER** и содержанием **\*.my**. Вообще то надо больше. Мы делаем самый минимум.

27_2.gif (1692 b)

### Создаем код

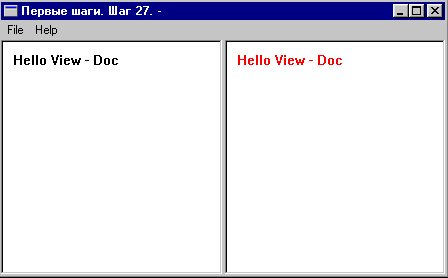
В классе документа.

class CMyDoc : public CDocument // класс документа  
{  
 ........  
 virtual BOOL OnNewDocument(); // будет вызываться при выборе меню New  
 **virtual void Serialize(CArchive& ar); // сохранение востановление файлов**  
 DECLARE\_DYNCREATE(CMyDoc)  
 ........  
};

И реализация.

void CMyDoc::Serialize(CArchive& ar)  
{  
 if (ar.IsStoring()) // если сохранение  
 {  
 ar << csDate; // записываем  
 }  
 else  
 {  
 // для чтения  
 }  
}

Вот так она будет работать.



### Описание

Итак, функция **Serialize**. Если кратко, то это реализация сохранения на диск и похожа она на работу с потоками. Присваивание пункту меню идентификатора **ID\_FILE\_SAVE** приведет к тому, что данные будут сохраняться на диск. Создайте пункт меню, и он сразу будет доступен даже без описания в таблице откликов. В **MFC** есть механизм обработки этого идентификатора без нашего вмешательства. При вызове пункта меню с таким идентификатором **MFC** автоматически вызывает функцию **Serialize** из класса документа зарегистрированного в шаблоне.

pDocTemplate = new CSingleDocTemplate(  
 IDR\_MAINFRAME, // идентификатор меню  
 RUNTIME\_CLASS(CMyDoc), // документ ----| Вот у этого класса вызовется Serialize !!!  
 RUNTIME\_CLASS(CMainFrame), // рамка окна ---| Документ вид однако.  
 RUNTIME\_CLASS(CMyView)); // просмотр ----|

А в этой функции мы автоматически сохраняет или читаем данные.

Только для работы этого пункта меню нужны строковые ресурсы. Именно с помощью них окну сохранения становится известно какой фильтр, какое имя файла. Если вы вспомните ["Шаг 15 - Стандартный диалог открытия файлов"](http://www.firststeps.ru/mfc/steps/r.php?15), то знаете, сколько параметров нужно передать классу **CFileDialog**. В данном случае эти параметры передаются через строковые ресурсы.

Компилируйте. Запускайте. Сохраните. Потом откройте в любом тестовом редакторе и вы увидите текст, который был на экране. Вот почти так работает волшебник, то есть создает такой код.

### Шпаргалка

Сохранение происходит с использованием функции **Serialize** класса документа

Пункт сохранения должен иметь идентификатор **ID\_FILE\_SAVE**

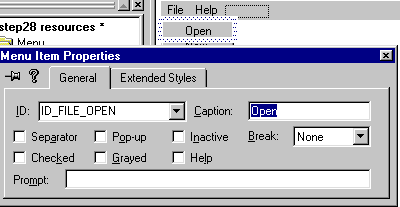
Необходимо объявить как минимум один строковый ресурс **AFX\_IDS\_ALLFILTER**

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/27.zip) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?26) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?28) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc1.html)  
Автор **Каев Артем**.

## Шаг 28 - Пункт меню открыть документ как в AppWizard

### Создание проекта

Продолжаем развитие нашего проекта. В ресурсах в меню необходимо создать пункт меню **Open** в меню **File**. И присвоить ему идентификатор **ID\_FILE\_OPEN**.



### Создаем код

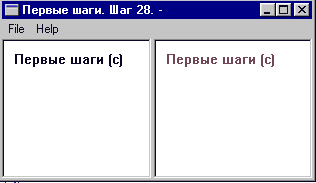
В таблице реакций на события создаем реакцию на пункт меню.

BEGIN\_MESSAGE\_MAP(CStep2App, CWinApp)  
 ON\_COMMAND(ID\_FILE\_NEW, CWinApp::OnFileNew) // пункт меню New  
 ON\_COMMAND(ID\_FILE\_OPEN, CWinApp::OnFileOpen) // пункт меню Open  
END\_MESSAGE\_MAP()

И добавим чтение данных.

void CMyDoc::Serialize(CArchive& ar)  
{  
 if (ar.IsStoring()) // если сохранение  
 {  
 ar << csDate; // записываем  
 }  
 else  
 {  
 **ar >> csDate; // для чтения  
 UpdateAllViews(NULL); // обновить виды**  
 }  
}

Вот так будет все это работать.



### Описание

Учтите на всякий случай, что это учебный пример для демонстрации механизмов **MFC**. Как видите это всё. Первый идентификатор необходим для заголовка окна открытия файла. Вызываем процедуру открытия файла и **MFC** всё делает автоматом.

В процедуру **Serialize** добавили код чтения. Она вызывется из **CWinApp::OnFileOpen**. Итак, запускайте. Вызовете **About**, чтобы появилась надпись Первые шаги и сохраните документ. Потом выберете пункт меню **New** для обновления экрана. На экране будет другая запись. А Вот теперь откройте и опять появиться Первые шаги. Всё работает как часы.

### Шпаргалка

Создай пункт меню и реакцию.

BEGIN\_MESSAGE\_MAP(CStep2App, CWinApp)  
 ........  
 ON\_COMMAND(ID\_FILE\_OPEN, CWinApp::OnFileOpen) // пункт меню Open  
 ........  
END\_MESSAGE\_MAP()

Создай процедуру чтения.

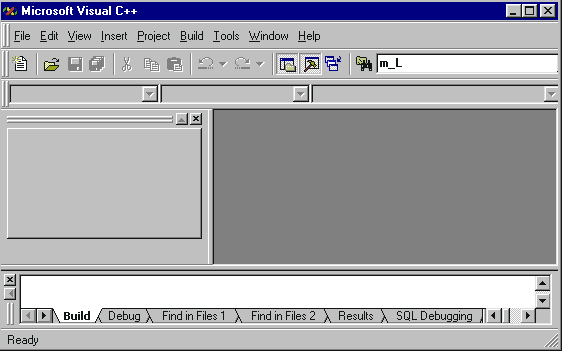
void CMyDoc::Serialize(CArchive& ar)  
{  
 .......  
 else  
 {  
 ar >> csDate; // для чтения  
 UpdateAllViews(NULL); // обновить виды  
 }  
}

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/28.zip) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?27) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?29) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc1.html)  
Автор **Каев Артем**.

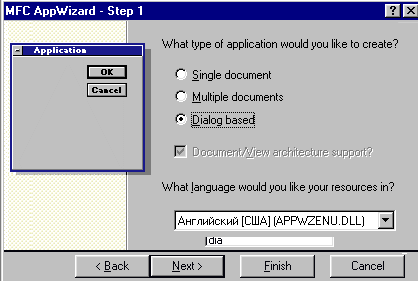
## Шаг 30 - Первый проект с AppWizard

### Создание проекта

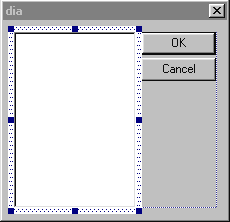
Ну, настал этот час. Запускайте **VC++**. Выбирайте меню **File** и **New**. Далее **Project** и тип создаваемого приложения **MFC AppWizard**. В поле **Project Name** необходимо указать название проекта. Назовите его **Dia** (это сокращение от диалог). Подтвердите Ваши серьезные намерения кнопкой **OK**.



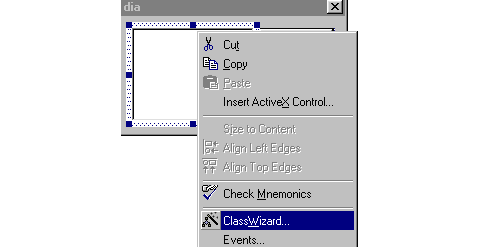
**Step 1** Мы будем создавать диалоговое окно. И поэтому выберем тип **Dialog Bases**. **Step 2** Дополнительные параметры. Отключите **About Box** он нам не нужен, это окно о программе. Также отключаем **3D Control** (он нужен для красоты, а нам не нужен). Отключаем **ActiveX Control**, у нас очень простое приложение. Идем дальше. **Step 3** Устанавливает статическую компоновку **As a statically library**. **Step 4** Идем дальше и нажимаем **Finish**. Проект готов.



Теперь в ресурсах изменим окно диалога, так как нарисовано ниже. Идентификатор окна должен быть о таким **IDD\_DIA\_DIALOG**. В диалоговое окно вставляем список **List Box**.



А вот теперь нам надо создать переменную для нашего списка. Выделяем его мышкой. Правую клавишу нажимаем и выбираем **ClassWizard[[2]](#endnote-2)**. А там вкладку **Member Variable**. Двойной щелчок на **IDC\_LIST**. И после **m\_** вводим имя переменной. Подтверждаем **OK** и всё закрываем. Обратите внимание на **Category**, это очень важно.

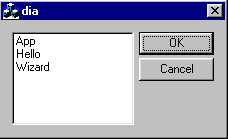


### Создаем код

Изменяем BOOL CDiaDlg::OnInitDialog(), чтобы при загрузки формы в списке были данные.

BOOL CDiaDlg::OnInitDialog()  
{  
 ...........  
 // TODO: Add extra initialization here  
 m\_List.AddString("Hello"); // Заносим наши данные  
 m\_List.AddString("App");  
 m\_List.AddString("Wizard");  
 return TRUE; // return TRUE unless you set the focus to a control  
}

А вот результат.



### Описание

Только мышка и три строки кода, приложение готово. Как в **Visual Basic** или **Delphi**.

### Шпаргалка

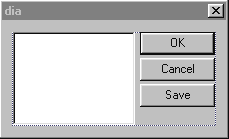
Главное здесь - это связь элемента в диалоговом окне с переменной типа элемент управления. Не забывайте указывать категорию **Control**.

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/30.zip) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?29) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?31) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc1.html)  
Автор **Каев Артем**.

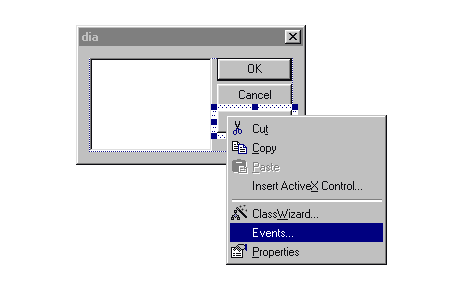
## Шаг 31 - Автоматизация реакции на события с ClassWizard

### Создание проекта

Работаем с предыдущим проектом. На диалоговое окно добавим кнопку. Теперь нас идентификаторы не волнуют, они задаются автоматически. И обработка их автоматически устанавливается. Даже функция реакции автоматически. Кнопка такая.



После того как Вы её поместили необходимо создать реакцию на событие нажатия. Нажмите правую кнопку мышки и выберите **Events[[3]](#endnote-3)**. У Вас появится диалоговое окно и с левой стороны будет список событий. Два раза щелкните на событии, чтобы создать реакцию.



Подтвердите всё кнопками **Ok**. Всё, реакция на событие создана. Ниже приведены основные моменты генерируемого кода.

// diaDlg.h : header file  
//  
class CDiaDlg : public CDialog  
{  
 // Construction  
public:  
 .......  
 enum { IDD = IDD\_DIA\_DIALOG };  
 **CListBox m\_List;**  
 //}}AFX\_DATA  
 ..........  
 afx\_msg HCURSOR OnQueryDragIcon();  
 **afx\_msg void OnButton1();**  
 //}}AFX\_MSG  
 ............  
};  
// diaDlg.cpp : implementation file  
//  
BEGIN\_MESSAGE\_MAP(CDiaDlg, CDialog)  
 //{{AFX\_MSG\_MAP(CDiaDlg)  
 ON\_WM\_PAINT()  
 ON\_WM\_QUERYDRAGICON()  
 **ON\_BN\_CLICKED(IDC\_BUTTON1, OnButton1)**  
 //}}AFX\_MSG\_MAP  
END\_MESSAGE\_MAP()  
.............  
void CDiaDlg::OnButton1()   
{  
 // TODO: Add your control notification handler code here  
}

### Создаем код

Нам нужно, что бы убедиться что все работает, как то видеть результат нажатия. Вот и будет окно сообщения.

void CDiaDlg::OnButton1()   
{  
 AfxMessageBox("Save");  
}

### Описание

**VC++** делает за Вас всё необходимое кроме реализации функции. Но если он и это будет делать, то мы все останемся без работы.

### Шпаргалка

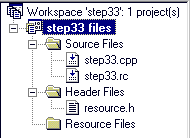
Реакция на действия с кнопкой устанавливается с помощью пункта **Events**.

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/31.zip) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?30) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?32) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc1.html)  
Автор **Каев Артем**.

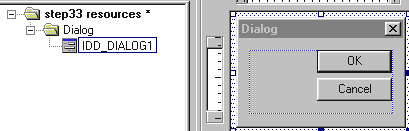
## Шаг 33 - Диалоговое окно приложения без волшебника

### Создание проекта

В данном проекте мы создадим окно программы как диалоговое окно, но без применения **AppWizard**, для понимания сути происходящего. Создавайте проект **Win32** с поддержкой **MFC**, как в примерах ранее и два файла **CPP** и **RC**. В файле ресурсов создайте простой диалог с двумя кнопками.



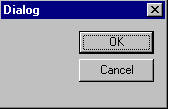
В ресурсы нужно добавить диалоговое окно.



### Создаем код

#include "afxwin.h"  
#include "resource.h" // ссылка на ресурсы  
class CTestDlg : public CDialog // класс диалогового окна  
{  
 public:  
 CTestDlg(CWnd\* pParent=NULL);  
 protected:  
};  
CTestDlg::CTestDlg(CWnd\* pParent)  
 : CDialog(IDD\_DIALOG1, pParent)  
{  
}  
class CTestApp : public CWinApp  
{  
public:  
 virtual BOOL InitInstance();  
};  
BOOL CTestApp::InitInstance()  
{  
 CTestDlg dlg; // Создать объект диалогового окна  
 m\_pMainWnd = &dlg; // главное окно программы это диалоговое окно  
 dlg.DoModal();  
 return FALSE;  
}  
CTestApp theApp;

Теперь наша программа будет выглядеть так.



### Описание

Мы создали класс приложения **CTestApp**, в котором оставили только метод **InitInstance**, который отвечает за создания окна. В этом методе вызываем создание диалогового окна. Присвоим указателю окна приложения указатель на диалоговое окно **m\_pMainWnd = &dlg;**.

Смотрите сверху. Включили описание классов **MFC** и описание идентификаторов ресурсов, где наше диалоговое окно **resource.h**. Создали класс диалогового окна, просто определив конструктор, который вызывает папу для своего построения.

CTestDlg::CTestDlg(CWnd\* pParent)  
 : CDialog(IDD\_DIALOG1, pParent)

В этот конструктор передали идентификатор диалогового окна из ресурсов **IDD\_DIALOG1**. После чего в функции **InitInstance** создали объект диалогового окна. И вызвали его для выполнения. Но вот результат возврата функции **FALSE**, чем обманули **MFC** сказав, что инициализация не удалась.

### Шпаргалка

Создай класс диалогового окна.

class CTestDlg : public CDialog // класс диалогового окна  
{  
 public:  
 CTestDlg(CWnd\* pParent=NULL);  
 protected:  
};

Создай объект и запусти диалоговое окно на выполнение.

BOOL CTestApp::InitInstance()  
{  
 CTestDlg dlg; // Создать объект диалогового окна  
 m\_pMainWnd = &dlg; // главное окно программы это диалоговое окно  
 dlg.DoModal();  
.........  
}

Заверши инициализацию неудачей.

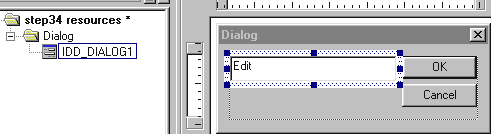
BOOL CTestApp::InitInstance()  
{  
 .......  
 return FALSE;  
}

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/33.zip) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?32) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?34) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc1.html)  
Автор **Каев Артем**.

## Шаг 34 - Понимание DDX

### Создание проекта

Для работы применим прошлый проект. Но на диалоговое окно добавим элемент управления **Edit** - строка редактирования.



### Создаем код

Добавляем функции к классу диалогового окна.

class CTestDlg : public CDialog // класс диалогового окна  
{  
 public:  
 CTestDlg(CWnd\* pParent=NULL);  
 **virtual void DoDataExchange(CDataExchange\* pDX); // функция обмена данными   
 void SetEditStr(CString csEdit); // установить строку  
 CString GetEditStr(); // получить строку**  
 protected:  
 **CString m\_edit; // данные**  
};

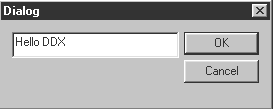
Реализация этих функций.

void CTestDlg::SetEditStr(CString csEdit)  
{  
 m\_edit=csEdit;  
}  
CString CTestDlg::GetEditStr()  
{  
 return m\_edit;  
}  
void CTestDlg::DoDataExchange(CDataExchange\* pDX)  
{  
 CDialog::DoDataExchange(pDX); // функция папы  
 DDX\_Text(pDX, IDC\_EDIT1, m\_edit); // наш обмен  
}

Изменяем **BOOL CTestApp::InitInstance()**.

BOOL CTestApp::InitInstance()  
{  
 CTestDlg dlg; // Создать объект диалогового окна  
 m\_pMainWnd = &dlg; // главное окно программы это диалоговое окно  
 dlg.SetEditStr("Hello DDX"); // установить данные  
 dlg.DoModal();   
 TRACE(dlg.GetEditStr()); // получить данные и вывести в окно отладки  
 return FALSE;  
}

Вот так будет работать программа. Не забудьте для того, что бы видеть вывод отладочной информации Вам надо запустить программу в режиме отладки (**F5**).



### Описание

**DDX** - это реализация функций обмена данных с диалоговым окном. **DDX** работает на основе связи переменной и идентификатора элемента управления в окне. Естественно этот элемент управления должен соответствовать типу переменной.

Для реализации связи поля редактирования я завел переменную **m\_edit** как строку **CString** в классе диалогового окна. И реализовал две функции для доступа к ней. Это функции **SetEditStr** и GetEditStr. В классе диалога описана виртуальная функция **DoDataExchange**, именно она отвечает за обмен данными. Вы не должны вызывать эту функцию непосредственно, вызов её происходит из функции **UpdateDate**. Эта функция вызывается при создании окна, закрытия окна, но вы можете вызвать её в любой момент. Внутри этой функции вызывается метод папы. А дальше устанавливается связь, но связь может быть нескольких типов в зависимости от типа элемента управления и типа переменной. Например, если бы я хотел установить связь с этим же элементом управления, но не с текстовой строкой и с объектом **CEdit**, то это выглядеть будет так.

class CTestDlg : public CDialog  
{  
 .......  
 CEdit m\_edit;  
}  
void CTestDlg::DoDataExchange(CDataExchange\* pDX)  
{  
 CDialog::DoDataExchange(pDX);  
 DDX\_Control(pDX, IDC\_EDIT1, m\_edit);  
}

Обратите внимание, что тип связи зависит и от элемента управления. Например, **DDX\_Radio** для переключателя. Дальше я перед вызовом диалогового окна устанавливаю переменную, а после показываю значение.

### Шпаргалка

Создайте переменную для связи.

class CTestDlg : public CDialog // класс диалогового окна  
{  
 ..........  
 CString m\_edit; // данные  
};

Опишите функцию связи и её реализацию.

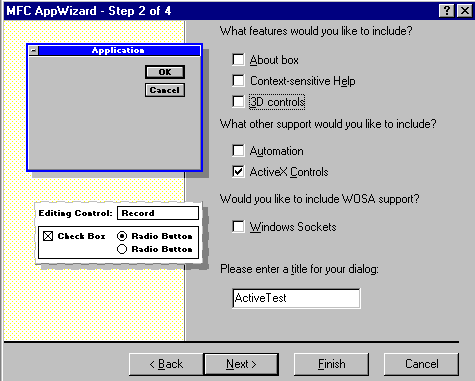
class CTestDlg : public CDialog // класс диалогового окна  
{  
 .....  
 virtual void DoDataExchange(CDataExchange\* pDX); // функция обмена данными   
 ....  
};  
void CTestDlg::DoDataExchange(CDataExchange\* pDX)  
{  
 CDialog::DoDataExchange(pDX); // функция папы  
 DDX\_Text(pDX, IDC\_EDIT1, m\_edit); // наш обмен  
}

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/34.zip) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?33) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?35) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc1.html)  
Автор **Каев Артем**.

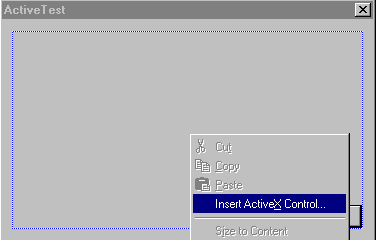
## Шаг 35 - Использование ActiveX с волшебниками

### Создание проекта

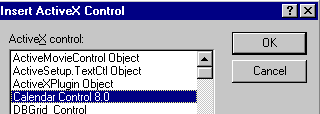
Задача этого проекта попробовать использование элементов ActiveX. Создавайте проект на основе диалогового окна. Всё нужно сделать как в [шаге 30](http://www.firststeps.ru/mfc/steps/r.php?30) за исключением добавления поддержки **ActiveX**, вот здесь.



После генерации кода открывается диалоговая панель, на основе которой создается окно программы. Теперь в диалоговое окно необходимо вставить элемент **ActiveX**. В рамке окна щелкайте правой кнопкой мыши и выбирайте пункт меню **Insert ActiveX Control**.



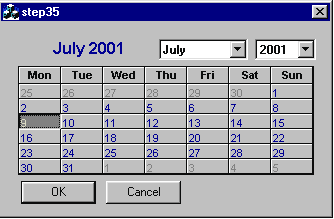
В результате у Вас появится список элементов управления зарегистрированных в системе.



Выберите календарь и нажмите **Ok**. В результате у Вас появится элемент управления на панели диалога. Установите размер для просмотра.

### Создаем код

Код не нужен, что очень хорошо. Так будет работать наш пример.



### Описание

Запускайте. После запуска у Вас будет функционировать календарь. Вот и Всё. Кода нет. Но и возможностей мало.

### Шпаргалка

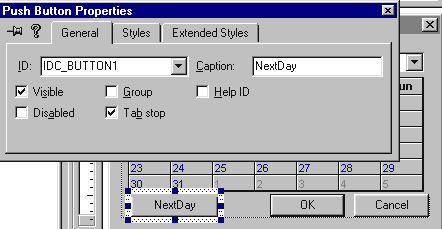
Для использования **ActiveX** включите поддержку при создании проекта.

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/35.zip) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?34) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?36) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc1.html)  
Автор **Каев Артем**.

## Шаг 36 - Класс для ActiveX с волшебником

### Создание проекта

Используем предыдущий. На диалоговой панели необходимо создать кнопку **NextDay**.

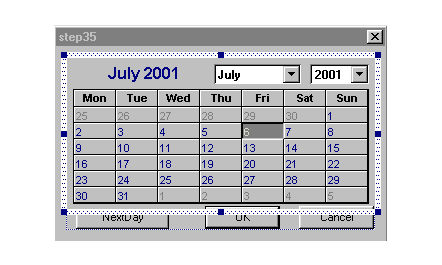


И процедуру реакции на эту кнопку. Это мы делали [раньше](http://www.firststeps.ru/mfc/steps/r.php?31).

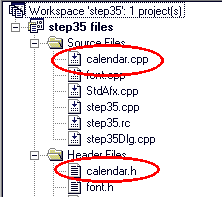
У вас должна появиться функция в ответ на нажатие кнопки.

void CStep35Dlg::OnButton1()   
{  
 // TODO: Add your control notification handler code here  
}

А теперь необходимо создать класс для элемента **ActiveX**. Это делается точно также, как и для обычного элемента управления. Только разница в том, что **VC** выдаст предупреждение.



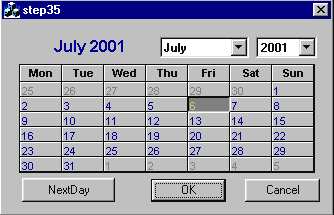
В результате в проекте должно появится два новых файла.



Автоматически описание класса календаря добавиться к описанию диалогового окна.

............  
//{{AFX\_INCLUDES()  
#include "calendar.h"  
//}}AFX\_INCLUDES  
#if !defined(AFX\_STEP35DLG\_H\_\_CEC26FC7\_71EB\_11D5\_B617\_E7F5E7CCE647\_\_INCLUDED\_)  
#define AFX\_STEP35DLG\_H\_\_CEC26FC7\_71EB\_11D5\_B617\_E7F5E7CCE647\_\_INCLUDED\_  
...........  
class CStep35Dlg : public CDialog  
{  
// Construction  
public:  
 CStep35Dlg(CWnd\* pParent = NULL); // standard constructor  
// Dialog Data  
 //{{AFX\_DATA(CStep35Dlg)  
 enum { IDD = IDD\_STEP35\_DIALOG };  
 CCalendar m\_Calender;  
 //}}AFX\_DATA  
...........

Вот так будет работать программа.



### Создаем код

В функции реакции на нажатие кнопки.

void CStep35Dlg::OnButton1()   
{  
 // TODO: Add your control notification handler code here  
 m\_Calender.NextDay();  
}

### Описание

**VС** создал нам всё необходимое, и мы этим воспользовались для перевода календаря на следующий день по нажатию кнопки. Запускайте и нажимайте на кнопку **NextDay**, число будет переводиться на следующий день. Всего одна строка кода! На основе этого примера есть один интересный нетривиальный подход. Например, вы разрабатываете программный продукт на **VC** и собираетесь использовать несколько своих элементов управления. Так вот. Можно создать их на **VB 5.0**, так как это просто и быстро сосредоточившись на функциональности. И на основе этого элемента проводить разработку дальше, в то время как Ваши специалисты создадут аналог на **VC**. Обратите внимание, что раньше элементы управления создавались на **VC** для **VB**, а теперь наоборот.

### Шпаргалка

Для управления элементом **ActiveX** необходимо создать соответствующий класс.

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/36.zip) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?35) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?37) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc1.html)  
Автор **Каев Артем**.

## Шаг 37 - Пользовательская функция DDX

### Создание проекта

Для работы с этой темой Вам понадобится проект созданный [в шаге 34](http://www.firststeps.ru/mfc/steps/r.php?34). В него будем вносить изменения.

### Создаем код

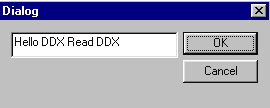
Создадим нашу **DDX** функцию.

void AFXAPI DDX\_MyText(CDataExchange\* pDx,int ID,CString& Date )  
{  
 if (pDx->m\_bSaveAndValidate) // Что происходит. Чтение или запись ?  
 {  
 DDX\_Text(pDx,ID,Date);  
 Date= Date+" Save DDX";  
}  
 else  
 {  
 Date=Date+ " Read DDX";  
 DDX\_Text(pDx,ID,Date);  
 }  
}

Изменения в функции обмена.

void CTestDlg::DoDataExchange(CDataExchange\* pDX)  
{  
 CDialog::DoDataExchange(pDX); // функция папы  
 **DDX\_MyText(pDX, IDC\_EDIT1, m\_edit); // наш обмен**  
}

Вот так вмешивается наша **DDX** функция.



А вот строка в результате отладки.

Loaded 'C:\WINDOWS\SYSTEM\INDICDLL.DLL', no matching symbolic information found.  
Read DDX Save DDXThe thread 0xFFF5086F has exited with code 0 (0x0).  
The program 'D:\VС\step37\Debug\step37.exe' has exited with code 0 (0x0).

### Описание

Итак, мы с Вами создали свою функцию обмена. Зачем это нужно? Пригодится точно. Например, для перевода строки в число и назад, перевода строки или добавления изменения. Вообще как фантазия сработает. Мы объявили функция как **AFXAPI** - это модификатор, который устанавливает вызов функции в соответствии с требованиями **MFC**. Объявить её можно в отдельном файле, и тем самым локализовать модификации типов в отдельном файле. В функцию мы передаем несколько параметров. Первое это объект, который поддерживает обмен данными. Дальше идентификатор элемента управления и тип передаваемых данных. **CDataExchange** знает направление, в котором идет обмен, своё понимание текущей ситуации он хранит в переменной **pDx->m\_bSaveAndValidate**, которую мы и анализируем. При сохранении мы вызываем функцию **DDX\_Text** для получения данных и потом модифицируем, а при чтении наоборот - сначала изменяем, а потом вызываем **DDX\_Text**. Нам осталось только описать её в функции **CTestDlg::DoDataExchange**, что мы и сделали. Запускайте.

### Шпаргалка

Создайте свою функцию **DDX**.

void AFXAPI DDX\_MyText(CDataExchange\* pDx,int ID,CString& Date )  
{  
 if (pDx->m\_bSaveAndValidate) // Что происходит. Чтение или запись ?  
 {  
 .............  
 }  
 else  
 {  
 ........  
 }  
}

Вызывайте её для обмена.

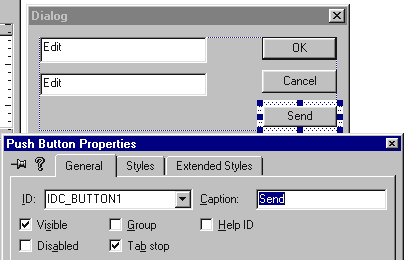
void CTestDlg::DoDataExchange(CDataExchange\* pDX)  
{  
 CDialog::DoDataExchange(pDX); // функция папы  
 DDX\_MyText(pDX, IDC\_EDIT1, m\_edit); // наш обмен  
}

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/37.zip) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?36) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?38) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc1.html)  
Автор **Каев Артем**.

## Шаг 40 - Как волшебник связывает кнопку с событием

### Создание проекта

Мы воспользуемся проектом с которым ведем работу. Создайте свою кнопку **Send** с идентификатором **IDC\_BUTTON1**, код работы с реестром лучше удалить.



### Создаем код

class CTestDlg : public CDialog // класс диалогового окна  
{  
.........   
 CString GetEditStr(); // получить строку  
 **void OnSend(); // функция реакции**  
 protected:  
.........   
 CString m\_newedit; // вторая строка  
 **DECLARE\_MESSAGE\_MAP(); // таблица откликов**  
};

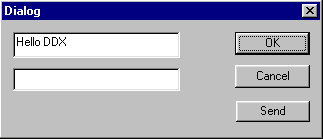
Реализация таблицы откликов.

BEGIN\_MESSAGE\_MAP(CTestDlg, CDialog)  
 ON\_BN\_CLICKED(IDC\_BUTTON1, OnSend)  
END\_MESSAGE\_MAP()

Сама функция.

void CTestDlg::OnSend()  
{  
 AfxMessageBox("Clicked !");  
}

А вот так это работает.



### Описание

Итак, раз нажатие кнопки вызовет реакцию, то для реализации реакции необходима процедура вот её **OnSend** я и описал в классе диалогового окна. **DECLARE\_MESSAGE\_MAP()** Вам знакома. **ON\_BN\_CLICKED(IDC\_BUTTON1, OnSend)** Вам знакома из [Шага 16](http://www.firststeps.ru/mfc/steps/r.php?16). И наконец реализация. Этот шаг простой и вероятно не очень важный, но **AppWizard** делает именно так, когда вы устанавливаете реакцию на событие нажатие кнопки, и если Вы должны понимать как это происходит, кроме того он умеет далеко не всё :-).

### Шпаргалка

Связь события нажатия устанавливается через обработку.

BEGIN\_MESSAGE\_MAP(CTestDlg, CDialog)  
 ON\_BN\_CLICKED(IDC\_BUTTON1, OnSend)  
END\_MESSAGE\_MAP()

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/40.zip) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?39) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?41) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc1.html)  
Автор **Каев Артем**.

## Шаг 41 - Вызов HTML страницы в Internet из приложения

### Создание проекта

Используем предыдущий проект.

### Создаем код

Добавляем библиотеку с функциями.

........  
#include "resource.h" // ссылка на ресурсы  
#pragma comment(lib,"shell32.lib") // подключили LIB файл  
class CTestDlg : public CDialog // класс диалогового окна  
.........

Процедура нажатия на кнопку.

void CTestDlg::OnSend()  
{  
 ShellExecute(NULL,"open", "http://www.mjk.msk.ru/~dron/mfc/", "", "c:\\", SW\_SHOWNORMAL);  
}

### Описание

Это всё . Вообще среди классов **MFC** есть и классы доступа к **HTTP** и много других для работы с **Internet**, но короче вы не напишете, вероятно. И если в вашей программе есть справка со ссылкой на помощь в **Internet** это не самое плохое решение. Но осторожно этот подход может вызвать проблемы, если вы используете **Netscape**.

Итак, первая строка подсоединяет к проекту библиотеку, т.е. указывает компоновщику на её присоединение. Очень эффективно и наглядно, чем постоянно ходить в меню настройки и добавлять библиотеки. Функция **ShellExecute** открывает файл вот её описание.

HINSTANCE ShellExecute(  
 HWND hwnd, // указатель на окно (откуда вызвано)  
 LPCTSTR lpOperation, // Указатель но определенные операции для выполнения  
 LPCTSTR lpFile, // Указатель на имя фала или папки   
 LPCTSTR lpParameters, // Параметры выполнения операции   
 LPCTSTR lpDirectory, // Указатель на каталог по умолчанию   
 INT nShowCmd // Тип окна для вывода.   
);

Смотрим на мои объявления. Вызывающее окно не указываем, поэтому устанавливаем **NULL**. Тип операции **open**, указатель на файл - адрес сайта, параметров нет, директория по умолчанию диск **С**, и показывать нормальное окно, т.е. не минимизированное или наоборот максимизированное. Проверяете работает !

### Шпаргалка

Для подключения LIB файлов используйте.

#pragma comment(lib,"shell32.lib") // подключили LIB файл

Автоматизируйте процесс вызова страниц с помощью.

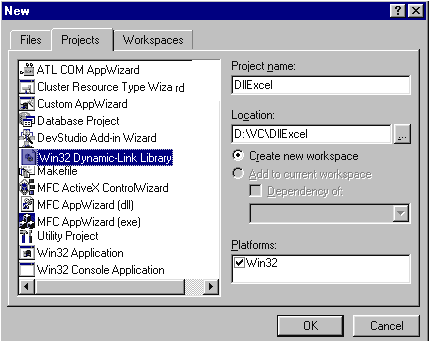
ShellExecute(NULL,"open", "http://www.mjk.msk.ru/~dron/mfc/", "", "c:\\", SW\_SHOWNORMAL);

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/41.zip) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?40) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?42) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc1.html)  
Автор **Каев Артем**.

## Шаг 46 - Dll для Excel

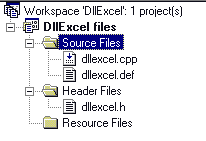
### Создание проекта

Создаем обычную динамическую библиотеку. Выбираем файл, создать, потом **Win32 Dynamik-Link Library**. Не забудьте указать имя проекта (**DllExcel**).



И три файла нужно создать в проекте. Как мы это делали в ["Шаг 1 - Минимальный проект"](http://www.firststeps.ru/mfc/steps/r.php?1).

dllexcel.cpp  
dllexcel.h  
dllexcel.def



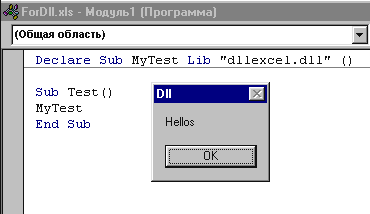
### Создаем код

// Step46  
// Сайт Первые шаги (firststeps.narod.ru)  
// Каев Артем (1999-2001)  
// создано 1999 г.  
// обновление 10.07.2001  
// dllexcel.h  
#include "windows.h"  
extern "C"  
{  
void WINAPI MyTest();  
}  
// Step46  
// Сайт Первые шаги (firststeps.narod.ru)  
// Каев Артем (1999-2001)  
// создано 1999 г.  
// обновление 10.07.2001  
// dllexcel.cpp  
#include "dllexcel.h"  
void WINAPI MyTest()  
{   
 MessageBox(0,"Hellos","Dll",MB\_OK);  
}  
; dllexcel.def  
EXPORTS  
MyTest @1

Описание и макрос **Excel**

Declare Sub MyTest Lib "dllexcel.dll" ()  
Sub Test()   
 MyTest  
End Sub

Вот так работает наша **DLL**.



### Описание

Первым шагом мы реализовали саму функцию в **CPP** файле. В общем всё тривиально, за исключением модификатора **WINAPI**. Существует несколько моделей вызовов функций. Для использования в **Windows** модель вызовов функций **PASCAL**. Для соответствия требованиям **WIN32 API** и добавлен модификатор. Дальше в **exceldll.h** мы описали данную функцию. Но применили опять модификатор для приведения имен функций к стандарту **C**. Применение этого стандарта гарантирует поддержку Ваших имен функций другими программами. Если **DLL** создается именно для вашего приложения, то это не важно, но если Вы собрались поделиться ей с миром, то должны обеспечить совместимость.

И последним шагом заполнили **def** файл, описав функцию **MyTest** как функцию экспорта, то есть что она будет извлекаться из **DLL**. Теперь собирайте проект. Полученную **DLL** нужно поместить в каталог **Windows/System** или просто **Windows** или **WinNT**, если **Windows NT**. При вызове функции из **DLL** необходимо чтобы **DLL** была найдена. В описанных каталогах операционная система обязательно посмотрит.

Теперь запустите **Excel**, создайте макрос, опишите функцию и код макроса. Пояснений я сюда писать не буду, т.к. это касается программирования **VBA**. А если вы этого не умеете, зачем вам этот шаг, хотя на Бейсике программировать может каждый. Запускайте макрос. Если Вы всё сделали правильно, то увидите диалоговое окно созданное функцией **DLL**.

Это очень хорошая возможность. Применив данную технику Вы можете расширить возможности любой программы, которая имеет макроязык и функции работы с **DLL**. Перечислю на взлет программы **Microsoft Office, Corel Draw (Corel Script), AutoCad (VBA), ArcView (Avenue)** и т.д.

### Шпаргалка

Функция должна быть описана в стандарте **С**.

extern "C"  
{  
 ........  
}

Вызов должен быть в стандарте вызовов **Windows**

..... WINAPI .........  
{   
 ............  
}

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/46.zip) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?45) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?47) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc1.html)  
Автор **Каев Артем**.

## Шаг 49 - Меняем размер и икону диалогового окна

### Создание проекта

Используем проект из [прошлого шага.](http://www.firststeps.ru/mfc/steps/r.php?48)

Добавим в ресурсы икону. Мы этого не делали, но принцип тот же. Как и меню, панели инструментов и строковые ресурсы.



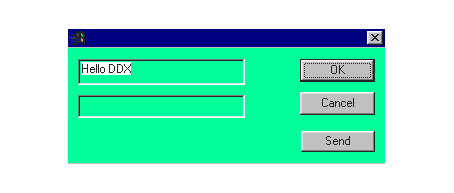
Можете нечего с ней не делать, то есть оставить, как есть.

### Создаем код

Изменим нашу функцию реакции нажатия на кнопку.

void CTestDlg::OnSend()  
{  
 MoveWindow( 100, 100 , 300, 300,TRUE ); // Переместить окно  
 HICON newIcon= AfxGetApp()->LoadIcon(IDI\_ICON1); // Прочитать икону  
 SetIcon(newIcon,TRUE); // Установить икону  
 SetWindowText("Move"); // Поменять заголовок  
}

Вот так будет реагировать окно на нажатие кнопки **Send**.



### Описание

Цель данного шага научиться делать то, что вроде запрещено. Итак, у нас диалоговое окно, в котором изменять размер запрещено. Это часто бывает обоснованно. Проще запретить изменение размера окна, чем постоянно центрировать элементы. При установке удаленного доступа, по нажатию на СВЕДЕНИЯ диалоговое окно увеличивается, хотя мышкой размер изменить нельзя. Все эти эффекты и многие другие основаны на том, что диалоговое окно имеет хорошего папу в виде **CWnd**. Свойствами папы мы и воспользуемся. **MoveWindow** перемещает окно. Первые два параметра - это положение относительно рабочего стола, а вторые два параметра это ширина и высота, **TRUE** указывает на необходимость перерисовки окна. А так как ширина и высота указывается, мы этим и пользуемся для изменения диалогового окна. Следующие строки кода меняют иконку диалогового окна. Для этого нам необходимо получить указатель на иконку, этот указатель, получается, через класс окна. Это функция **LoadIcon**, результатом её пополнения является указатель на иконку. В эту функцию передается идентификатор иконки из ресурсов. Функция **SetIcon** используя указатель, устанавливает иконку в диалоговом окне. **SetWindowsText** изменяет текст заголовка диалогового окна. Запускайте и компилируйте. Нажимайте на кнопку. Окно переместится , изменится иконка, и заголовок окна.

### Шпаргалка

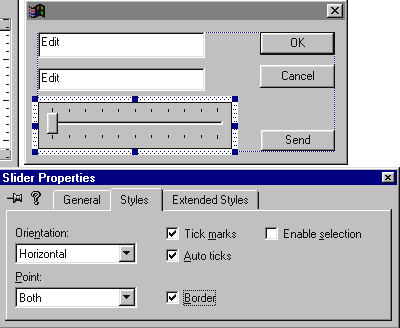
Дополнительные возможности можно получить исследую базовые классы.

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/49.zip) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?48) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?50) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc1.html)  
Автор **Каев Артем**.

## Шаг 50 - Что такое CSliderCtrl

### Создание проекта

Работаем со старым. Только на диалоговую панель необходимо поместить элемент управления - бегунок. Как внизу.



### Создаем код

Включение описание класса для элемента управления.

#include "afxdlgs.h" // для диалогового окна печати  
**#include "afxcmn.h" // это для элемента управления**  
#pragma comment(lib,"shell32.lib") // подключили LIB файл

Описание класса в классе окна диалога.

class CTestDlg : public CDialog // класс диалогового окна  
{  
 ........   
 CBrush my\_color; // сама кисть  
 **CSliderCtrl csSlider; // класс элемента управления**  
 protected:  
 .........   
};

Реализация обмена данными элемента управления с классом.

void CTestDlg::DoDataExchange(CDataExchange\* pDX)  
{  
 ................  
 DDV\_MaxChars(pDX, m\_newedit, 2); // проверка данных  
 **DDX\_Control(pDX, IDC\_SLIDER1, csSlider); // наш обмен**  
}

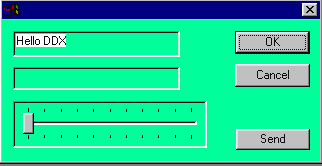
Реакция нажатия на кнопку для просмотра показаний бегунка.

void CTestDlg::OnSend()  
{  
 int i=csSlider.GetPos(); // получить позицию  
 char csPos[10]; // буфер для результата   
 itoa(i,csPos,10); // перевести в строку  
 AfxMessageBox(csPos); // показать результат  
}

Инициализация диалогового окна для настройки ползунка.

BOOL CTestDlg::OnInitDialog()  
{   
 ...........  
 my\_color.CreateSolidBrush(RGB(0, 255, 155)); // создать кисть  
 **csSlider.SetRangeMin(0,TRUE); // минимум  
 csSlider.SetRangeMax(10,TRUE); // максимум**  
 return TRUE;  
}

Вот так все это будет работать.



### Описание

Класс **CSliderCtrl** - это класс бегунка. Он включает в себя все функции по управлению этим элементом. Самые примирительные мы и рассмотрим. Так как этот элемент управления используется не во всех проектах, его описание вынесено за пределы **afxwin.h** и находится в **afxcmn.h**. Вот мы и подключаем описание заголовочного файла. Далее в классе диалогового окна мы включаем класс. Этот класс имеет конструктор по умолчанию, поэтому создается вместе с диалоговым окном. В последствии с использованием механизма **DDX** мы связываем класс с элементом управления.

При нажатии на кнопку мы запрограммировали вывод показаний элемента управления. Происходит это следующим образом. В переменную **i** мы помещаем значение, используя функцию **csSlider.GetPos()**. Объявляем массив символов. Функция **itoa** переводит число в строку на основе установленной системы счисления. Нам нужна десятичная. Можете другую, если есть настроение. И используя **AfxMessageBox** доводим до Вашего сведения.

В функции диалогового окна мы с Вами устанавливаем минимальное и максимальное значения элемента. **TRUE** отвечает за перерисовку элемента управления. Пользоваться функциями **csSlider.SetRangeMin** и **csSlider.SetRangeMax** можно в любой момент. Вот для этого и существует последний параметр. Если элемент не виден, то вероятно и перерисовывать его нет смысла. Запускайте. Продвиньте бегунок на две-три позиции и нажмите кнопку. Появится диалоговое окно с выбранным значением.

### Шпаргалка

включить описание класса.

#include "afxcmn.h" // это для элемента управления

Объявить его.

class CTestDlg : public CDialog // класс диалогового окна  
{  
 ........   
 CSliderCtrl csSlider; // класс элемента управления  
 .........   
};

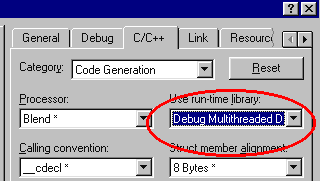
Связать класс элемента управления с элементом в окне.

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/50.zip) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?49) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?51) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc2.html)  
Автор **Каев Артем**.

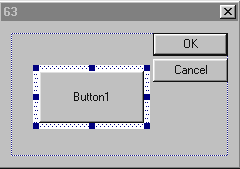
## Шаг 63 - Многозадачность

### Создание проекта

Создаем проект на базе диалогового окна. Убедитесь, что у Вас стоит поддержка многопоточных приложений.



Помещаем кнопку и функцию, связанную с ее нажатием.



### Создаем код

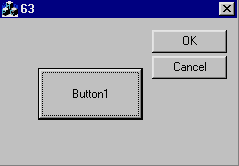
Создаем две функции для потоков.

UINT proc1(LPVOID Param)  
{  
 for (int x=1;x<30;x++)  
 {  
 Sleep(100); // задержка  
 }  
 AfxMessageBox("proc1");   
 return 0;  
}  
UINT proc2(LPVOID Param)  
{  
 for (int x=1;x<30;x++)  
 {  
 Sleep(100); // задержка  
 }  
 AfxMessageBox("proc2");  
 return 0;  
}

Создаем реакцию на нажатие кнопки для запусков потоков.

void CMy63Dlg::OnButton1()   
{  
 AfxBeginThread(proc1,this); // Первый поток  
 AfxBeginThread(proc2,this); // Второй поток  
 AfxMessageBox("Thread"); // И основной поток  
}

Программа в работе.



### Описание

Основу кода составляет **AfxBeginThread**. Эта функция запускает процесс. У неё два обязательных параметра , адрес процедуры потока и параметры. Смотрите сами.

CWinThread\* AfxBeginThread( AFX\_THREADPROC pfnThreadProc, LPVOID pParam, int nPriority = THREAD\_PRIORITY\_NORMAL, UINT nStackSize = 0, DWORD dwCreateFlags = 0, LPSECURITY\_ATTRIBUTES lpSecurityAttrs = NULL );  
CWinThread\* AfxBeginThread( CRuntimeClass\* pThreadClass, int nPriority = THREAD\_PRIORITY\_NORMAL, UINT nStackSize = 0, DWORD dwCreateFlags = 0, LPSECURITY\_ATTRIBUTES lpSecurityAttrs = NULL );

Указатель на функцию в потоке должен иметь вид.

UINT MyControllingFunction( LPVOID pParam );

Второй параметр устанавливает приоритет выполнения функции. Я использовал **this** для того, чтобы функции потока имели тот же приоритет, что и программа. Но установка этих параметров Ваше право. Собирайте, запускайте.

Смотрите последовательность выполнения. **OnButton1** не остановилась на процедурах, а запустила их в виде потоков и выполняется дальше, о чем говорят диалоговые окна **Thread**, потоки завершаются позднее и примерно в одно время, как и начинались. Когда выполнение приложения распределено на несколько направлений, выполнение каждого из которых не зависит от другого. Якобы не зависит. Процессор то один. Но в нашем случае в процедурах реализованы задержки, которые не требуют процессорного времени и влияние друг на друга у них маленькое.

### Шпаргалка

Проверить, что приложение сможет использовать потоки.

Project - Setting - C/C++ - Code Generation - Use Run-Time Library

Реализовать функции потоков.

UINT proc1(LPVOID Param)  
{  
..........  
}

Запустить поток.

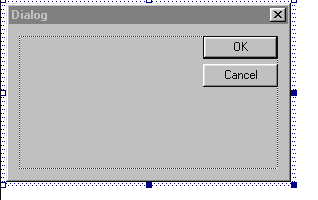
AfxBeginThread(proc1,this); // Первый поток

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/63.zip) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?62) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?64) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc2.html)  
Автор **Каев Артем**.

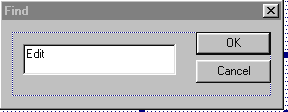
## Шаг 69 - Диалоговое окно для поиска в CEditView

### Создание проекта

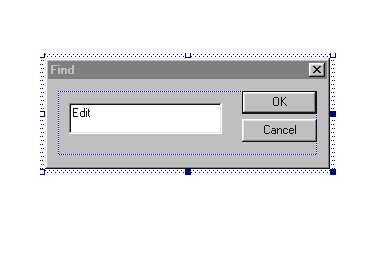
В прошлом шаге мы с Вами создали поиск. Теперь пора приготовить диалоговое окно для ввода строки поиска. Берите предыдущий проект и вперед. Нам нужно добавить диалоговое окно к проекту как мы это делали в [шаге 9](http://www.firststeps.ru/mfc/steps/r.php?9). Теперь нам нужно поменять заголовок окна. Для этого вызываем свойства диалогового окна и меняем свойства **Caption**.



Теперь нам нужно поместить на диалоговое окно элемент управления **Edit**.



После создания диалогового окна и расположения элементов управления для создания нового класса можно сделать двойной щелчок на панели диалогового окна. По нему будет создан класс на основе данного диалогового окна. Появится диалоговое окно **Create New Class** , здесь нужно согласиться, ввести имя нового класса и в **ClassView** появится наш класс.



После всех приготовлений мы можем связать элемент управления диалогового окна, используя механизм **DXX** с переменной, только сначала её нужно создать. Запускайте **ClassWizard**, выбирайте вкладку **Member Variables**. Идентификатор строки редактирования будет иметь имя **IDC\_EDIT1** свяжем его с переменной типа **CString**. Это мы уже делали в [шаге 30](http://www.firststeps.ru/mfc/steps/r.php?30).

### Создаем код

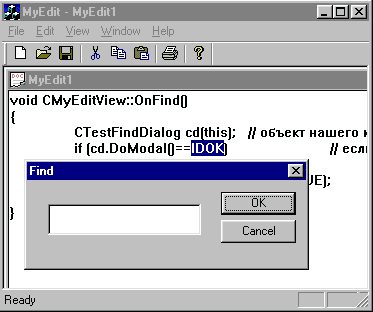
Подключаем описание класса диалогового окна поиска к реализации вида.

// MyEditView.cpp : implementation of the CMyEditView class  
//  
..............  
#include "MyEditDoc.h"  
#include "MyEditView.h"  
**#include "TestFindDialog.h"**

Меняем функцию реакции на вызов поиска.

void CMyEditView::OnFind()   
{  
 CTestFindDialog cd(this); // объект нашего класса  
 if (cd.DoModal()==IDOK) // если в окне нажали OK  
 {   
 FindText(cd.m\_Search,TRUE); // искать текст  
 }  
}

Вот она, в работе милая.



### Описание

Для того что улучшить поиск мы с Вами создали класс диалогового окна. Этот класс можно создавать на основе диалога в ресурсах. **VC++** это делает автоматически. В дальнейшем нам надо подключить его и использовать.

И еще один момент. Не ищите, пустую строку или обрабатывайте ошибку.

### Шпаргалка

Вы можете расширить интерфейс стандартных элементов управления, создав свои диалоги для передачи параметров.

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/69.zip) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?68) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?70) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc2.html)  
Автор **Каев Артем**.

## Шаг 71 - Что формирует MFC AppWizard

### Описание

**AppWizard** формирует файлы и классы в зависимости от типа формируемого приложения. Конечно, включение поддержки баз данных и составных документов влияет на формируемые файлы. Мы рассмотрим простые случаи.

### SDI

При генерации приложения данного типа формируются следующие классы в виде заголовочного файла **\*.H** и файла реализации **\*.CPP**. Будет создано 5 классов.

\* - имя проекта  
CAboutDlg - класс диалога о программе (About)  
C\*SDIApp - класс приложения ( включает порожденый класс от CWinApp внутри которого WinMain)   
C\*SDIDoc - класс документаC\*SDIView - класс просмотра   
CMainFrame - класс рамки окна

### MDI

Обратите внимание, что различия очень маленькие **SDI** меняется на **MDI**. Кроме того различия очень маленькие есть в коде. Например, для формирования шаблона в **SDI** используется **CSingleDocTemplate**, а в **MDI - MultiDocTemplate**.

\* - имя проекта  
CAboutDlg - класс диалога о программе (About)  
C\*MDIApp - класс приложения ( включает порожденый класс от CWinApp внутри которого WinMain)   
C\*MDIDoc - класс документаC\*MDIView - класс просмотра  
CMainFrame - класс рамки окна

### Dialog Based

Формируется 3 класса.

\* - имя проекта  
CAboutDlg - класс диалога о программе (About)  
C\*DialogApp - класс приложения ( включает порожденый класс от CWinApp внутри которого WinMain)   
C\*DialogDlg - класс главного диалогового окна.

### Некторые дополнения.

Приложения созданные **AppWizard** уже умеют работать с командной строкой. Это реализуется функцией **ProcessShellCommand**. Ниже перечисляются поддерживаемые параметры.

нет параметров - запускает приложения и открывает пустой файл  
имя файла - запускает и открывает данный файл  
/p имя файла - запускает и печатает файл на принтере по умолчанию  
/pt имя файла Принтер Драйвер Порт - запускает и печатает на специфичном принтере  
/dde - запускает и ждет команд DDE   
/Automation - запускает в качестве сервера OLE  
/Embedding - запускает в режиме внедренного объекта OLE

Как видите автоматически у Вас формируется большое количество возможностей, например, перетащить файл на принтер и бросить, если зарегистрировано расширение и многое другое.

### Немного о коде

В начале кода Вы можете увидеть, что-то типа -

#if !defined(AFX\_CHILDFRM\_H\_\_58CE2B44\_BDCB\_11D3\_A392\_000001260696\_\_INCLUDED\_)  
#define AFX\_CHILDFRM\_H\_\_58CE2B44\_BDCB\_11D3\_A392\_000001260696\_\_INCLUDED\_

В этой надписи нет нечего страшного, это аналог -

#if !defined(My\_H)   
 #define My\_H

И создано всё это для того, чтобы исключить повторные включения в такой большой код это просто для гарантии, тем более что Вам это самим не вносить. В коде Вы можете увидеть конструкции типа.

// Overrides   
 // ClassWizard generated virtual function overrides  
 //{{AFX\_VIRTUAL(CChildFrame)   
 virtual BOOL PreCreateWindow(CREATESTRUCT& cs);  
 //}}AFX\_VIRTUAL

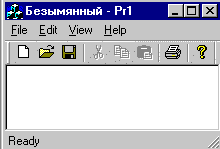
Не меняйте нечего внутри AFX\_\*, так как эту конструкцию использует **ClassWizard** для добавления и удаления функций.

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?70) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?72) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc2.html)  
Автор **Каев Артем**.

## Шаг 72 - Тренировка - приложение Pr1

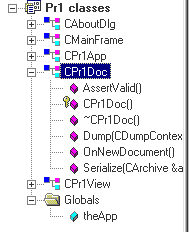
### Создание проекта

Запускайте **VC**. Выбирайте **MFC AppWizard** и дайте приложению имя **Pr1**. Шаг 1 мы выберем **Single document** - документ с одним окном. Теперь можете смело жать **Finish** всё остальное оставим по умолчанию. Полюбуйтесь на диалоговое окно с информацией и закройте его. Выберите пункт меню **Build - Build Pr1.exe**. После генерации приложения давайте его запустим. **Build**, а далее **Execute Pr1.exe**. У Вас должно быть примерно такое окно.



### Создаем код

Что ж, окно есть. А теперь нам надо выводить строку в окно. Для этого нам нужно завести переменную типа **CString**, в которой будет храниться строка. Вопрос, где её заводить? Для этого есть класс документа. Там должны храниться все данные. Перейдите в окно **ClassView**, там вы увидите класс **CPr1Doc** именно в нем и должна храниться наша переменная. Щелкните правой кнопкой мыши на классе **CPr1Doc** и выберите **Add Member Variable**. Этот пункт меню позволяет добавлять пененные к классу. В поле **Variable Type** введите **CString**, а в поле **Variable Name - m\_string**. Для простоты в поле **Access**, то есть доступ, оставим **Public**. Теперь всё готово, жмем **OK**. Ваша переменная появится в списке методов и данных класса.



А что произошло в коде? Щелкните 2 раза на имени **m\_string** и Вы попадете на её описание в классе в файле **Pr1Doc.h**

..........  
 //}}AFX\_VIRTUAL  
// Implementation  
public:  
 CString m\_string;  
 virtual ~CPr1Doc();  
#ifdef \_DEBUG  
 virtual void AssertValid() const;  
 virtual void Dump(CDumpContext& dc) const;  
#endif  
........

Теперь нам необходимо инициализировать её содержание. Лучше сделать это в методе **OnNewDocument**. Этот метод будет всегда вызываться, когда создаётся новый документ. Щелкните на этом методе два раза. Здесь будет код.

BOOL CPr1Doc::OnNewDocument()  
{  
 if (!CDocument::OnNewDocument())  
 return FALSE;  
 // TODO: add reinitialization code here  
 // Добавите код для инициализации   
 // (SDI documents will reuse this document)  
 // SDI будут повторно использовать этот документ.   
 return TRUE;  
}

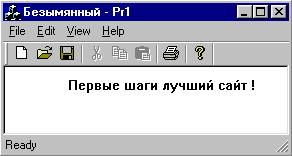
Замените комментарии **Todo**: на команду заполнения строки **m\_string = "Первые шаги - лучший сайт!";**. Вот так!

BOOL CPr1Doc::OnNewDocument()  
{  
 if (!CDocument::OnNewDocument())  
 return FALSE;  
 m\_string = "Первые шаги - лучший сайт!"; // инициализация   
 return TRUE;  
}

Следующая задача вывести эту строку на экран. Отображения класса документа (**CPr1Doc**) производится классом **CPr1View** и методом этого класса **OnDraw**. Переключитесь на класс **VPr1View** и щелкните два раза на методе **OnDraw**. Вот код.

/////////////////////////////////////////////////////////////////////////////  
// CPr1View drawing  
  
void CPr1View::OnDraw(CDC\* pDC)  
{  
 CPr1Doc\* pDoc = GetDocument(); // получить класс документа  
 ASSERT\_VALID(pDoc); // проверить как получили  
 pDC->DrawText(pDoc->m\_string,CRect(10,10,300,100),DT\_CENTER); // вывести текст в контекст экрана  
 // взяв его из класса документа   
}

Вот работающая программа.



### Описание

Попробуем сделать выводы. Ну, первое, **AppWizard** многое делает за Вас. Данные хранятся в классе документа (**CPr1Doc**) и для их наличия необходимо к этому классу добавить переменные. Переменные надо инициализировать. Для этого инициализацию нужно сделать в методе **OnNewDocument[[4]](#endnote-4)** класса документа, обратите внимание на то, что не в конструкторе. Дело в том каждый раз, когда создается новый документ, вызывается именно метод **OnNewDocument**, а не конструктор. Отображаются данные в классе вида (**CPr1View**). Для их отображения используется метод **OnDraw**.

### Шпаргалка

Данные должны храниться в классе документа.

Получить документ из вида можно воспользовавшись функцией - **GetDocument()**

CPr1Doc\* pDoc = GetDocument(); // получить класс документа  
ASSERT\_VALID(pDoc); // проверить как получили

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/72.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?71) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?73) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc2.html)  
Автор **Каев Артем**.

## Шаг 73 - Сохранение и восстановления данных.

### Создание проекта

Работаем с проектом из [прошлого шага](http://www.firststeps.ru/mfc/steps/r.php?72).

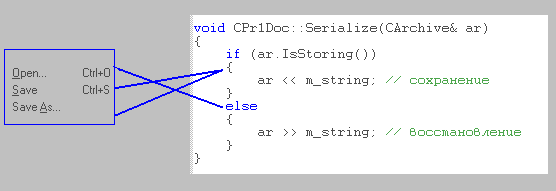
### Создаем код

У нас есть функция **CPr1Doc::Serialize**. Перепишем ее.

void CPr1Doc::Serialize(CArchive& ar)  
{  
 if (ar.IsStoring())  
 {  
 ar << m\_string; // сохранение  
 }  
 else  
 {  
 ar >> m\_string; // восстановление  
 }  
}

### Описание

Эта функция будет всегда вызываться при открытии или сохранении файла. То есть, если Вы выбираете пункты меню **File - > Open -> Save -> Save As...** всегда вызывается эта функция.



Для записи и чтения данных определены операторы << и >>. Эти операторы аналогичны потокам. Они определены как минимум для всех простых типов.

BYTE   
WORD   
int   
LONG   
DWORD   
float   
double

Кроме того, многие классы поддерживают данные операторы. Например, **CString**. Но для каждого класса надо смотреть отдельно. Если Вы используете свои классы и хотите производить подобные операции за один заход, то Вам придется перегрузить данные операции для Вашего класса. Следует отметить, что **Serialize** позволяет иметь только последовательный доступ. Для более тонкой работы Вы можете использовать класс **CFile**. Но мы пока на этом остановимся.

### Шпаргалка

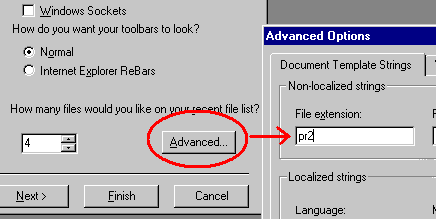
Итак, класс документа обеспечивает Вас функцией Serialize, с помощью который Вы можете сохранять и восстанавливать данные. Для использования этой возможности Ваши данные должны поддерживать операции **<<** и **>>**.

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/73.zip) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?72) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?74) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc2.html)  
Автор **Каев Артем**.

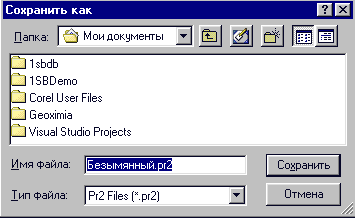
## Шаг 74 - Тренировка - приложение Pr2

### Создание проекта

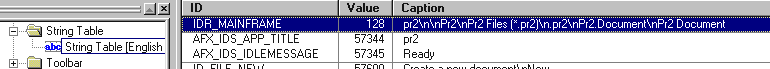
Ну что-же, запускайте **VC++** и **MFC AppWizard**. Имя приложению дайте **PR2**. В первом шаге тип приложения надо дать на основе одного документа **Single Document**. Дальше до 4 шага идем без изменений. А вот **Step4** нам нужен. Так как наше приложение будет записывать в файл, то нам необходимо решить с расширением. Вот как это делается.



На **Step4** жмем на **Advanced** и поле **File extension** указываем расширение. Дальше все можно не менять **Finish** это то, что нужно. Следует отметить, что слово **Finish** здесь понимается в положительном плане. А нет так как делаешь что - то, а потом - всё финиш. Если Вам это знакомо. Скомпилируйте и запустите. А потом выберите пункт меню **Save As..** и в поле **Save as type** вы увидите Ваше расширение.



Где хранится это расширение? В ресурсах. Откройте их. Дальше откройте **String Table** и идентификатор **IDR\_MAINFRANE**. Там такая строка.



Видите? О данном идентификаторе более полно в [подробностях](http://www.firststeps.ru/mfc/detail/r.php?9).

### Создаем код

Итак, мы будем записывать, и считывать 3 переменные типа **CString**. Все данные программы должны храниться в классе документа. Он у нас называется **CPr2Doc** туда мы и должны их добавить. Помните как в **PR1**?. Правой кнопкой мыши на **CPr2Doc** и **Add member variable**. Их нужно три **m\_string1 m\_string2 m\_string\_3**. В файле **pr2Doc** в описании класса документа добавятся строки вот так.

class CPr2Doc : public CDocument

{

// Implementation

public:

**CString m\_string3;**

**CString m\_string2;**

**CString m\_string1;**

virtual ~CPr2Doc();

#ifdef \_DEBUG

virtual void AssertValid() const;

virtual void Dump(CDumpContext& dc) const;

#endif

};

Инициализировать их нужно в **OnNewDocument** класса документа. Перейдите в него и внесите код.

BOOL CPr2Doc::OnNewDocument()

{

if (!CDocument::OnNewDocument())

return FALSE;

m\_string1="String1";

m\_string2="String2";

m\_string3="String3 Finish String";

return TRUE;

}

А вот теперь и будем организовывать процедуры записи. Они реализуются в функции **Seliarize** класса документа. Найдите её в **ClassView** и внесите изменения.

void CPr2Doc::Serialize(CArchive& ar)

{

if (ar.IsStoring())

{

ar << m\_string1;

ar << m\_string2;

ar << m\_string3;

} else {

ar >> m\_string1;

ar >> m\_string2;

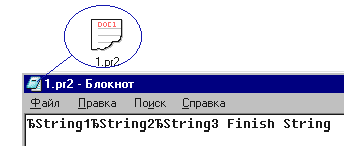
ar >> m\_string3;

}

}

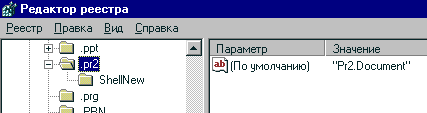
### Описание

Запускайте проект и сохраните файл под именем 1. Что произойдет ? В каталоге проекта появится файл с именем **1.pr2**. Расширение автоматически связано с файлом. В **exploler** два раза щелкните на имени файла и откроется Ваше приложение. Откройте файл **1.pr2** в тестовом редакторе там будет примерно вот что.



### Шпаргалка

Связать расширение с приложением просто. Воспользуйтесь **AppWizard**. Он сделает необходимые записи в реестре для реализации этой возможности.



[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/74.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?73) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?75) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc2.html)  
Автор **Каев Артем**.

## Шаг 75 - Команды и сообщения

**Windows** от **DOS** отличается многими вещами. Но одно из отличий это наличие в **Windows** сообщений. Если в **Dos** программа сама должна была опрашивать все устройства, например, мышь и клавиатуру, то в **Windows** это делает операционная система и передает программе (окну) сообщения. Забота программиста обработать эти сообщения и его уже не заботит, как они получены. В **Windows** с устройствами работают драйвера виртуальных устройств **VXD**. Их функционирование забота производителя. Вам уже не надо считывать буфер клавиатуры или перехватывать прерывания мышки. Ваша программа просто ждет, когда её известят о событии. Известят сообщением.

Обычное программирование на **C** для **Windows** превращается в сплошную обработку операторов **switch**. **MFC** берет на себя всю черную работу позволяя Вам сконцентрироваться на обработке. Идентификация сообщений в **MFC** производится целыми числами с использованием **define**. Вот часть файла **Winuser.h** с определением сообщений.

....

#define WM\_MOUSEFIRST 0x0200

#define WM\_MOUSEMOVE 0x0200

#define WM\_LBUTTONDOWN 0x0201

#define WM\_LBUTTONUP 0x0202

#define WM\_LBUTTONDBLCLK 0x0203

#define WM\_RBUTTONDOWN 0x0204

....

Как видите сообщения, поступающие от Windows имеют префикс **WM\_**. В **MFC** для обработки сообщений принято использование карты сообщений **Message Map**. Эту карту сообщений имеют все классы порожденные от **CCmdTarget**. И не просто имеют, а должны иметь обязательно. Карта сообщений описывается в заголовочном файле с описанием класса ответственного за её обработку.

....

// Описание функций откликов на сообщения

// Generate message map function

// {{AFX\_MSG(CMyDialog)

virtual BOOL OnInitDialog();

afx\_msg void OnPaint();

....

//}}AFX\_MSG

// Декларация наличия карты обработки сообщения

DECLARE\_MESSAGE\_MAP()

И в файле реализации связывая сообщение с непосредственной функцией, которая будет выполняться по его поступлению.

BEGIN\_MESSAGE\_MAP((CMyDialog,CDialog)

//{{AFX\_MSG\_MAP(CMyDialog)

.....

ON\_WM\_SYSCOMMAND()

ON\_WM\_PAINT()

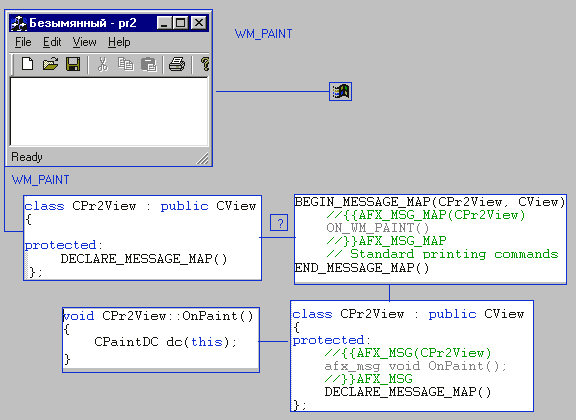
ON\_BN\_CLICKED(IDC\_EDIT,OnEdit)

.....

//}}AFX\_MSG

END\_MESSAGE\_MAP()

Комментарии **AFX\_MSG** позволяют **ClassWizard** быстро ориентироваться в карте обработки для вставки сообщений. **ON\_WM\_....** это макрос для обработки сообщения.

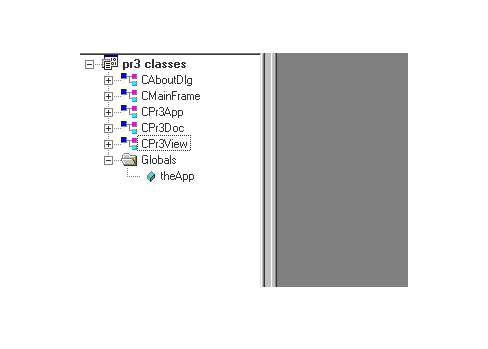


[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?74) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?76) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc2.html)  
Автор **Каев Артем**.

## Шаг 76 - Тренировка - приложение Pr3

### Создание проекта

Запускаем **VC** и **AppWizard**. Имя приложения **pr3** выбираем приложение на базе одного документа и сразу **Finish**. Мы с Вами будет создавать отклик на сообщение **Windows** в нашей программе. Выбираем класс. Правой кнопкой мыши вызываем свойства. Дальше **Add Windows Message Handles**. Вы выбираем сообщения. Щелкаем по нему. Справа в окне появится выбранное сообщение. Выбираем его и говорим **Add Handler**.



### Создаем код

Изменим код для того, чтобы вызвать окно сообщения в момент вызова данной функции.

void CPr3View::OnSize(UINT nType, int cx, int cy)   
{  
 CView::OnSize(nType, cx, cy);  
 AfxMessageBox("WM\_SIZE"); // в момент обработки сообщения появится диалоговое окно  
}

Cоздайте обработку сообщения **WM\_KILLFOCUS**

void CPr3View::OnKillFocus(CWnd\* pNewWnd)   
{  
 CView::OnKillFocus(pNewWnd);  
 AfxMessageBox("WM\_KILLFocus"); // в момент обработки сообщения появится диалоговое окно  
}

### Описание

Скомпилируйте и запустите. Каждый раз, когда вы будете переключаться на другое окно появится это сообщение. Оно так же появится при закрытии программы. Проверьте сами. Кроме того сообщение **WM\_SIZE** будет появляться каждый раз при изменение размера окна, кстати при создании приложения тоже.

Что происходит при проведении всех этих процедур? Добавляются описания функций в заголовочном файле класса.

class CPr3View : public CView  
{  
...........  
// Generated message map functions  
protected:  
 //{{AFX\_MSG(CPr3View)  
 afx\_msg void OnSize(UINT nType, int cx, int cy);  
 afx\_msg void OnKillFocus(CWnd\* pNewWnd);  
 //}}AFX\_MSG  
 DECLARE\_MESSAGE\_MAP()  
};

В таблице обработки сообщений добавляются записи.

BEGIN\_MESSAGE\_MAP(CPr3View, CView)  
 //{{AFX\_MSG\_MAP(CPr3View)  
 ON\_WM\_SIZE()  
 ON\_WM\_KILLFOCUS()  
 //}}AFX\_MSG\_MAP  
 // Standard printing commands  
 ON\_COMMAND(ID\_FILE\_PRINT, CView::OnFilePrint)  
 ON\_COMMAND(ID\_FILE\_PRINT\_DIRECT, CView::OnFilePrint)  
 ON\_COMMAND(ID\_FILE\_PRINT\_PREVIEW, CView::OnFilePrintPreview)  
END\_MESSAGE\_MAP()

В файле реализации создаются функции.

/////////////////////////////////////////////////////////////////////////////  
// CPr3View message handlers  
void CPr3View::OnSize(UINT nType, int cx, int cy)   
{  
 CView::OnSize(nType, cx, cy);  
}  
void CPr3View::OnKillFocus(CWnd\* pNewWnd)   
{  
 CView::OnKillFocus(pNewWnd);  
}

### Шпаргалка

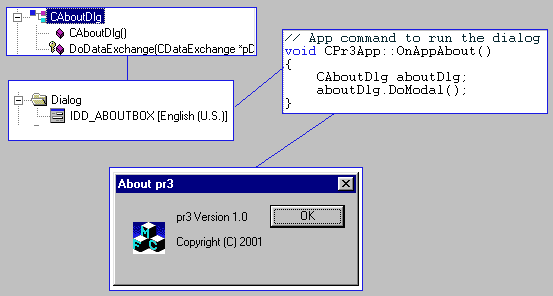
Можно обрабатывать любые сообщения **Windows** самостоятельно.

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/76.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?75) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?77) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc2.html)  
Автор **Каев Артем**.

## Шаг 77 - О диалоговых окнах

В понятии графического интерфейса **GDI** есть понятие диалогового окна (**dialog box**). Если дать простое определение, то это окно, которое не может изменять свои размеры и у которого нет меню. Каждое приложение может иметь огромное количество диалоговых окон или ни одного. Но всё же большинство приложений содержат хотя бы одно диалоговое окно **About** . Для работы с диалоговыми окнами в **MFC** нужно -

Ресурс диалогового окна.  
Класс окна.



Ресурс диалогового окна используется для вывода окна на экран самого окна и элементов управления, которые находиться на нем. Класс диалогового окна необходим для программной реализации управления окном и его элементами. Ресурсы диалогового окна создаются редактором ресурсов. Класс окна создает **ClassWizard** на основе ресурсов диалогового окна. Обычно все эти классы это дети от класса **CDialog**. Каждый элемент управления обычно имеет переменную в классе диалогового окна. При генерации приложения **AppWizard'ом** у Вас автоматически генерируется окно о программе.



**IDD\_ABOUTBOX** это идентификатор диалоговго окна. Он описан в файле **resoure.h**.

.....  
// Microsoft Visual C++ generated include file.  
// Used by EE.RC  
//  
#define IDD\_ABOUTBOX 100  
#define IDR\_MAINFRAME 128  
#define IDR\_EETYPE 129  
.....

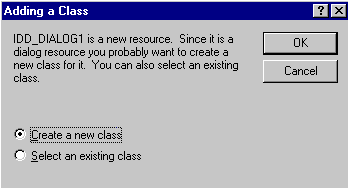
В файле реализации класса приложения описан класс для этого окна и сразу за ним реализация.

class CAboutDlg : public CDialog  
{  
public:  
 CAboutDlg();  
 ......   
};

В последствии организуется вывод диалогового окна функцией **DoModal**. Надо отметить, что диалоговые окна бывают модальные и немодальные. Модальные диалоговые окна не дают работать с окном программы до их закрытия, немодальные на работу с программой не влияют, а просто висят и мешаются. У каждого окна своё предназначение.

// App command to run the dialog  
void CEeApp::OnAppAbout()  
{  
 CAboutDlg aboutDlg;  
 aboutDlg.DoModal();  
}

Создается диалоговое окно в ресурсах. Нажимаете правую кнопку мыши. Выбираете **Insert** и **Dialog**. Для создания класса надо на диалоговом окне нажать правую кнопку мыши и выбрать пункт меню **ClassWizard**. В ответ вам предложат создать новый класс или выбрать существующий. Для подробнойстей смотрите ["Шаг 69 - Диалоговое окно для поиска в CEditView"](http://www.firststeps.ru/mfc/steps/r.php?69).



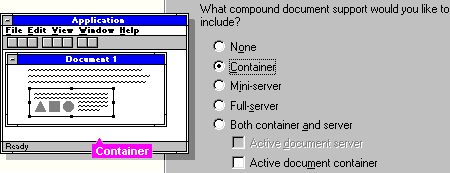
Создавая новый класс у Вас спросят имя. Задаете имя и нажимаете **OK**. В результате создастся класс. Его можно увидеть в **ClassView**. Создастся два файла заголовочный **H** и **CPP**.

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?76) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?78) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc2.html)  
Автор **Каев Артем**.

## Шаг 78 - Управление приложениями из VC++

### Создание проекта

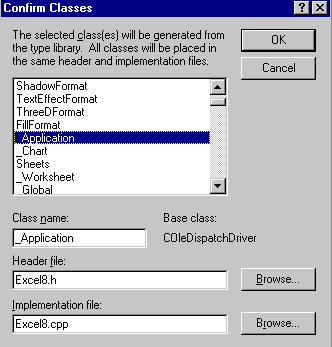
Итак, запускайте **MFC AppWizard**. Дайте имя проекту **Test1**. В **шаге 1 - Single Document**. **Шаг 2** без изменений. **Шаг 3** установите **Container** и смело жмите **Finish**.



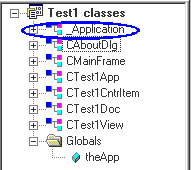
Итак, немного теории. Практически все современные приложения являются серверами **OLE**. То есть они написаны так, что ими может управлять любая программа, которая является контейнером. Вобщем-то за нас это сделал **AppWizard**. Мы только ему пояснили, что и где. Информация о функциях данного элемента хранится в специальных файлах **TLB,OLB**. Например, информация о функциях и классах **Excel** находится здесь.

..\Programm Files\Microsoft Office\Office\Excel8.olb

Дайте поиск по дискам расширений **OLB** и Вы найдете много интересного. Итак, давайте добавим классы **Excel** к нашему приложению. Запускайте **ClassWizard**. Выбирайте вкладку **Automation**. Дальше **Add class** и из выпадающих вариантов **From type library**. Найдите **Excel8.olb** и нажмите **OK**. У Вас появится диалоговое окно со списком классов, которые Вы можете сформировать. Нас пока интересует **\_Application** найдите его и выделите.



После этого жмите **OK** и закрываете окно **ClassWizard**. Смотрите окно **ClassView** появился класс **\_Application** в двух файлах **excel8.h** и **excel8.cpp**.



Ну, все пора писать код. Мы по щелчку мышки будем активизировать **Excel** получать имя текущего принтера, показывать его на несколько секунд на экране и закрывать.

### Создаем код

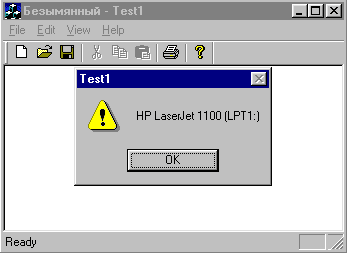
Нам необходимо поместить ссылку на класс **\_Application** вот так.

static char THIS\_FILE[] = \_\_FILE\_\_;  
#endif  
#include "excel8.h"  
/////////////////////////////////////////////////////////////////////////////  
// CTest1View

Теперь мы перегрузим события нажатия левой кнопки в окне и напишем код.

void CTest1View::OnLButtonDown(UINT nFlags, CPoint point)   
{  
 \_Application ap; // объект приложение  
 ap.CreateDispatch("Excel.Application"); // связаться  
 CString thePrinter = ap.GetActivePrinter(); // получить текущйи принтер   
 AfxMessageBox(thePrinter); // показать принтер   
 ap.SetVisible(TRUE); // показать Excel  
 Sleep(1000); // подождать немного  
 CView::OnLButtonDown(nFlags, point);  
}

Вот такое окно будет, потом запустится **Excel**.



### Описание

Смотрите сами. Создаем экземпляр класса. Связываем класс с реальным объектом. А теперь обращаемся к функции **Excel** взять текущий принтер. Выводим полученную строку на экран. Выводим сам **Excel** на экран. Задержка для того, чтобы увидеть чего-нибудь. Компилируйте, запускайте и наслаждайтесь. Это очень приятно, не нужен никакой **VBA**!. Реальная работа сложнее и требует других классов, но начало положено !

### Шпаргалка

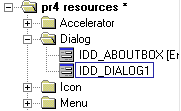
**Automation** позволяет управлять Вашими приложениями из программ **VC++**. Для этого нужно найти соответствующею библиотеку типов, и на основе ее создать классы для использования в приложении. Но в этом случает, Ваше приложение будет сильно привязано к приложению функции которого оно использует и может выступать только в качестве дополнения.

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/78.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?77) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?79) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc2.html)  
Автор **Каев Артем**.

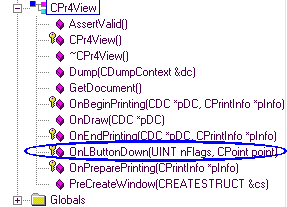
## Шаг 79 - Тренировка приложение PR4

### Создание проекта

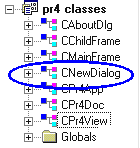
Запускайте **VC** и **MFC AppWizard**. Давайте имя ему **PR4**. Выберите **Multiple Document** и сразу жмите **FINISH**. Нам нужно добавить свой диалог в проект. Как мы это делали в проекте ["Шаг 9 - Диалоговая панель"](http://www.firststeps.ru/mfc/steps/r.php?9).



Давайте добавим обработку нажатия клавиши мышки для Вида, как в ["Шаг 76 - Тренировка - приложение Pr3"](http://www.firststeps.ru/mfc/steps/r.php?76).



Теперь нужно создать класс диалогового окна. Это мы делали в ["Шаг 69 - Диалоговое окно для поиска в CEditView"](http://www.firststeps.ru/mfc/steps/r.php?69).



### Создаем код

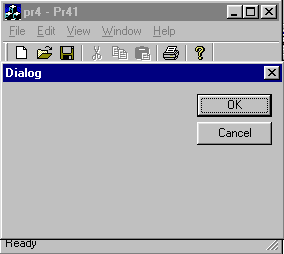
Подключаем описание класса диалогового окна.

..........  
static char THIS\_FILE[] = \_\_FILE\_\_;  
#endif  
#include "NewDialog.h"  
/////////////////////////////////////////////////////////////////////////////  
// CPr4View  
..........

Код.

/////////////////////////////////////////////////////////////////////////////  
// CPr4View message handlers  
void CPr4View::OnLButtonDown(UINT nFlags, CPoint point)   
{  
 // TODO: Add your message handler code here and/or call default  
 CNewDialog cdialog; // объект класса  
 cdialog.DoModal(); // сам класс  
 CView::OnLButtonDown(nFlags, point);  
}

Программа в действии.



### Описание

Этот пример является сбором всего, что проходили ранее. Вызов диалогового окна при выборе объекта в виде не является новостью и используется во многих программных решениях.

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/79.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?78) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?80) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc2.html)  
Автор **Каев Артем**.

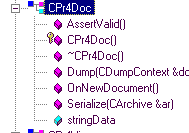
## Шаг 80 - Тренировка приложение PR5

От: Dmitry Panferov   
 Дата: 10 января 2000 г. 3:51   
 Тема: MFCByStep - question  
 Пример: Есть MDI приложение, из меню вызывается диалоговое окно в котором  
 выводятся данные из документа и затем поступают обратно в документ,  
 в соответствии с новыми данными должно измениться и представление документа

### Создание проекта

Создаем точно такое приложение как в [PR4](http://www.firststeps.ru/mfc/steps/r.php?79). Все один к одному. Зайдем в Resource и поместим на диалоговое окно Edit Box и свяжем его с переменной типа **CString**. Имя дайте **m\_Edits**, как в ["Шаг 30 - Первый проект с AppWizard"](http://www.firststeps.ru/mfc/steps/r.php?30).

Теперь нам надо создать переменную в классе документа для хранения и отображения. Класс документа **CPr4Doc**. Выделите его в **ClassView**, нажмите правую кнопку мыши и **Add Member Variable**. Тип переменной **CString** имя **stringData**, а тип доступа **public**. Жмите **OK**.



### Создаем код

Переменную необходимо инициализировать - это в функции OnNewDocument.

BOOL CPr4Doc::OnNewDocument()  
{  
 if (!CDocument::OnNewDocument())  
 return FALSE;  
 stringData=""; // наша переменная  
 return TRUE;  
}

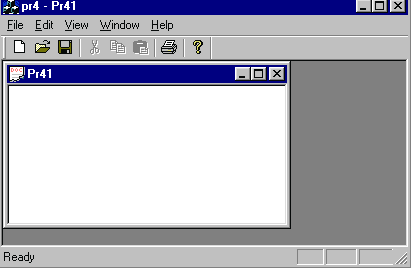
Для отображения этой переменной необходимо изменить функции рисования **OnDraw** класс просмотра **CPr4View**.

void CPr4View::OnDraw(CDC\* pDC)  
{  
 CPr4Doc\* pDoc = GetDocument();  
 ASSERT\_VALID(pDoc);  
 pDC->TextOut(10,10,pDoc->stringData); // вывести надпись  
}

А вот теперь можно менять функцию нажатия на кнопку мыши в виде.

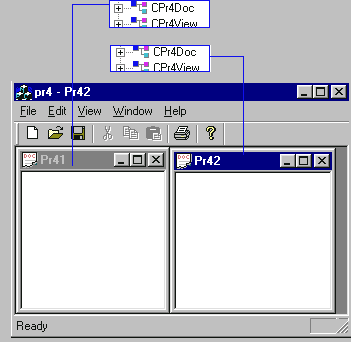
void CPr4View::OnLButtonDown(UINT nFlags, CPoint point)   
{  
 CNewDialog cdialog; // объект класса  
 if (cdialog.DoModal()==IDOK) // если пользователь OK выбрал  
 {   
 CPr4Doc\* pDoc = GetDocument(); // получить класс документа  
 pDoc->stringData=cdialog.m\_Edits; // поменять переменную класса документа   
 Invalidate(); // обновить вид  
 }  
 CView::OnLButtonDown(nFlags, point);

Смотрим, как будет работать.



### Описание

Приложение **MDI** работает немного не так как **SDI**. Создавая новый документ вы создаете дубликаты классов **Doc** и **View**. Два документа, два дубликата, три документа - три дубликата. В работающем приложении они независимые. Ваш ввод будет обрабатывать тот вид, который сейчас активный.



### Шпаргалка

Каждый документ в приложении **MDI** независим.

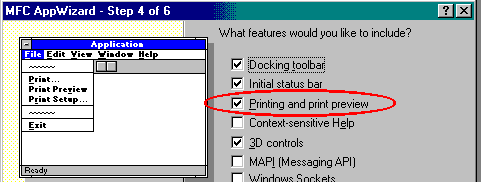
[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/80.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?79) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?81) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc2.html)  
Автор **Каев Артем**.

## Шаг 81 - Распечатка и предварительный просмотр

От: Dmitry Panferov   
 Дата: 10 января 2000 г. 3:51   
 Тема: MFCByStep - question  
 Хотелось бы еще поподробнее узнать о печати на принтер  
 т.к. известно что на экране мы все измеряем пикселами  
 а на печать уже все идет в физических величнах  
 Поэтому мне пришлось практически полностью переделывать функцию  
 OnDraw чтобы распечатать данные на принтер

### Создание проекта

Первое, что нужно сделать - это реализовать проект на основе **MFC AppWizard**, который поддерживает предварительный просмотр. Для этого на шаге 4 волшебника необходимо установить **Print and Priview** опцию.



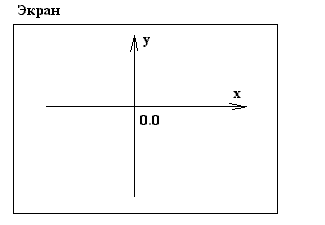
### Описание

С этого момента всё, что Вы будете рисовать в функции **OnDraw** класса вида, всё будет отражаться в окне предварительного просмотра. Только Вы должны будете заметить, что размеры далеко не совпадают. Это связанно с тем, что контекст устройства по умолчанию работает с графикой в единицах **MM\_TEXT**. Так как принтер и экран могут иметь разное количество единиц на страницу, это порождает и проблемы масштабирования.

При работе с графикой используйте режим **MM\_LOENGLISH**. Вообще-то их несколько. Вот перечисление режимов.

MM\_HIENGLISH 0.001 дюйма  
MM\_HIMETRIC 0.01 мм  
MM\_ISOTROPIC определяет пользователь  
MM\_LOENGLISH 0.01 дюйма  
MM\_LOMETRIC 0.1 мм  
MM\_TEXT пиксель устройства  
MM\_TWIPS 1/1440 дюйма

Вы можете выбирать любой. Но **MM\_LOENGLISH** достаточно удобный. Только следует точно знать о расположении координат и нуля. Для каждого режима они разные. Для **MM\_LOENGLISH** вот так.



Для установки режима у контекста устройства есть функция **SetMapMode**, с помощью которой переключается режим. Вызывайте её первой, и всё будет в порядке. Режим **MM\_LOENGLISH** гарантирует, что и на экране и на принтере будет выводиться графика одинаково на основе размеров заданных в дюймах.

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?80) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?82) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc2.html)  
Автор **Каев Артем**.

## Шаг 82 - Тренировка приложение PR6

### Создание проекта

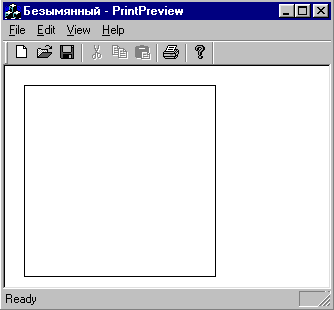
Запускаем **VC++** и **MFC AppWizard**. Имя проекту дайте **PrintPreview**. На первом шаге поставьте только **Single Document**. На втором и третьем ничего не трогаем, а на шаге четыре оставьте активной только опцию **Print and Print Priview**. И можно нажимать **Finish**

### Создаем код

Меняет функцию **OnDraw**

void CPrintPreviewView::OnDraw(CDC\* pDC)  
{  
 CPrintPreviewDoc\* pDoc = GetDocument();  
 ASSERT\_VALID(pDoc);  
 pDC->SetMapMode(MM\_LOENGLISH); // режим экрана  
 pDC->Rectangle(20,-20,220,-220); // эллипс  
}

Программа в работе.



### Описание

Как известно функции рисования графики должны располагаться в функции **OnDraw** класса вида. Все что рисуется в **OnDraw**, будет видно в окне предварительного просмотра. Мы привели координаты экрана к единицам **MM\_LOENGLISH** для того, что бы точно знать какой размер получим на принтере. Размер изображения на принтере будет 2 дюйма. 200/100 = 2.

### Шпаргалка

Предварительный просмотр использует функцию OnDraw класса вида.

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/82.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?81) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?83) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc2.html)  
Автор **Каев Артем**.

## Шаг 92 - Резиновый контур

Если Вы работали с графическими программами типа **Corel Draw** или с этим Вы точно работали **Pain**, то вы должны знать этот прием. Когда выделяете несколько объектов или рисуете приямоугольник, то у Вас появляется резиновый контур, который будет тянуться за мышкой. С технологией программирования данного метода мы и познакомимся.

Создавайте проект **MFC AppWizard** имя проекта **MouseSelect** ставьте **Single Document** и жмите **Finish**. Вы можете делать по другому скажем для **Doc** здесь все упрощено ;-)))).

Все методы будут для класса **CMouseSelectView**. Итак давайте разберемся с логикой.

Нажали мышкой.

Тут надо сообщить, что начали рисовать и

получить начальные координаты

Тащим.

Постоянного перерисовываем контур

Отпускаем.

Прекращаем рисовать о чем сообщаем

Создаем две переменные одна типа **CRect**, в которой мы будем хранить координаты квадрата вторая **BOOL**, чтобы определить, что мы действительно рисуем квадрат, а не метаемся по экрану от скуки.

class CMouseSelectView : public CView

{

.....

BOOL TestDraw;

CRect MouseRect;

........

};

Теперь нужно начальная инициализация

CMouseSelectView::CMouseSelectView()

{

// TODO: add construction code here

TestDraw=FALSE;

}

Нажатие мышкой

void CMouseSelectView::OnLButtonDown(UINT nFlags, CPoint point)

{

// TODO: Add your message handler code here and/or call default

TestDraw=TRUE;

DrawLine();

MouseRect.left=point.x;

MouseRect.top=point.y;

MouseRect.right =point.x;

MouseRect.bottom =point.y;

SetCapture();

CView::OnLButtonDown(nFlags, point);

}

Тащим

void CMouseSelectView::OnMouseMove(UINT nFlags, CPoint point)

{

if (TestDraw)

{

DrawLine();

MouseRect.right=point.x;

MouseRect.bottom = point.y;

DrawLine();

CView::OnMouseMove(nFlags, point);

}

}

Отпускаем

void CMouseSelectView::OnLButtonUp(UINT nFlags, CPoint point)

{

TestDraw=FALSE;

ReleaseCapture();

CView::OnLButtonUp(nFlags, point);

}

А это загадочная **DrawLine**:

void CMouseSelectView::DrawLine()

{

CClientDC dc(this);

int oldMode=dc.SetROP2(R2\_NOT);

CBrush\* oldBrush=(CBrush\*)dc.SelectStockObject(NULL\_BRUSH);

CPen\* oldPen = (CPen\*) dc.SelectStockObject(BLACK\_PEN);

dc.Rectangle(MouseRect);

dc.SelectObject(oldBrush);

dc.SelectObject(oldPen);

dc.SetROP2(oldMode);

}

Итак смотрим. При нажатии левой кнопкой мы устанавливаем координаты **MouseRect**, в точку. После этого вызываем **SetCapture**. Это интересная функция. Она скажет **Windows**, что мышка работает только для данного приложения даже за его приделами. Когда мы мышку перемещаем, то стираем старый прямоугольник и рисуем новый с измененными координатами. В момент когда мышка отпущена освобождаем мышку :-) **ReleaseCapture** и сбрасываем флаг рисования.

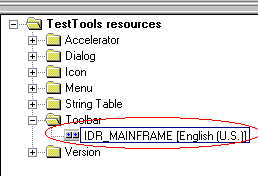
Самое интересно происходит в **DrawLine** мы получаем контекст устройства, с которым работаем **dc(this)**. Мы устанавливаем режим инвертирования пикселей **dc.SetROP2(R2\_NOT)** сохраняя старый режим. Зачем ? Инвентировали в одну сторону видно, еще раз и изображение восстановилось. Это много лучше, чем перерисовывать весь экран. Дальше устанавливаем перья, рисуем прямоугольник и восстанавливаем режим и перья контекста.

Здесь не предусмотрено много. Например поменяйте размер окна и изображение будет подпорченно. Появится квадрат. Ну это уже не очень большая проблемма. Пишите дополнения :-)))).

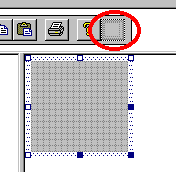
[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/92.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?91) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?93) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc2.html)  
Автор **Каев Артем**.

## Шаг 94 - Добавление кнопок на панель инструментов

Создавайте приложение **TestTools**, как **SDI**. После выбора **SDI** жмите **Finish**. Все остальные настройки пока не важны. Переходите на **ResourseView**. Там уже подгтовлена панель управления. Вот где она.

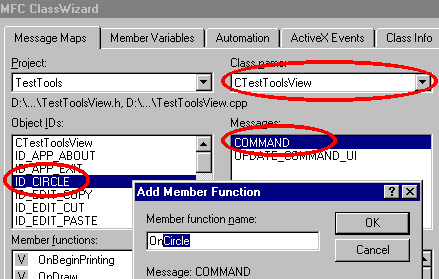


Если на ней шелкнуть два раза, то у Вас появится возможность редактировать её. Я хочу сделать кнопку, которая будет рисовать круг по нажатию на неё на экране. Для этого мне нужно создать рисунок круга. В панели уже придусмотрено место для новой кнопки. Надо выбрать пустую кнопку и нарисовать круг. Вот она пустая.



Нарисуйте в неё круг. Только вы это сделаете в правой части появится место для новой кнопки. Двойной щелчок на нашей кнопке с кругом позволит изменить её идентификатор. Замените его на **ID\_CIRCLE**. И закройте редактирование панели инструментов. Все готово для создания функции рисования. Запускайте **Class Wizard**.

Здесь можно создать событие для любого класса. То есть функцию, которая будет вызываться при нажатии на кнопку. Создадим для **CToolsVies**.



Всё, что осталось сделать это написать код для этой функции. Вот он.

void CTestToolsView::OnCircle()

{

CDC\* pDC;

pDC=GetDC();

pDC->Ellipse(CRect(10,10,100,100));

}

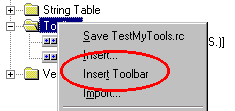
Компилируйте и запускайте. По нажатию на кнопку появится круг. Во как просто оказывается.

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/94.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?93) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?95) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc2.html)  
Автор **Каев Артем**.

## Шаг 95 - Добавление своей панели инструментов

В этом шаге мы создадим свою панель инструментов и поместим туда нашу кнопку. Это полезно, если вы хотите вынести на панель специфические действия Вашей программы, которые на стандартной панели как то не к месту.

Создаем приложение **TestMyTools**, как **SDI** и после выбора **SDI** сразу **Finish**. Нормально и быстро. Переходим в **ResourceView** там нам нужен **Toolbar**. Щелкаем правой кнопкой мыши и выбираем пункт **Insert Toolbar**.



Дальше как в тот раз рисуем круг и присваиваем идентификатор **IDC\_CURCLE**. Панель есть теперь надо её вывести на экран.

Создаются панели инструментов в функции **OnCreate** класса **CMainFrame**. Только перед этим нам необходимо завести класс панели интсрументов. Один у нас есть он был сгенерирован **App Wizard**, а второй мы добавим.

class CMainFrame : public CFrameWnd

{

.....

CToolBar m\_wndToolBar; // было

CToolBar m\_wndToolBar1; // добавили

....

};

Теперь нам нужно повторить инииализацию в **OnCreate** только использовать идентификатор нашей панели и наш класс. Жирным дополнения

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)

{

if (CFrameWnd::OnCreate(lpCreateStruct) == -1)

return -1;

if (!m\_wndToolBar.CreateEx(this, TBSTYLE\_FLAT, WS\_CHILD | WS\_VISIBLE | CBRS\_TOP

| CBRS\_GRIPPER | CBRS\_TOOLTIPS | CBRS\_FLYBY | CBRS\_SIZE\_DYNAMIC) ||

!m\_wndToolBar.LoadToolBar(IDR\_MAINFRAME))

{

TRACE0("Failed to create toolbar\n");

return -1; // fail to create

}

if (!m\_wndToolBar1.CreateEx(this, TBSTYLE\_FLAT, WS\_CHILD | WS\_VISIBLE | CBRS\_TOP

| CBRS\_GRIPPER | CBRS\_TOOLTIPS | CBRS\_FLYBY | CBRS\_SIZE\_DYNAMIC) ||

!m\_wndToolBar1.LoadToolBar(IDR\_TOOLBAR1))

{

TRACE0("Failed to create toolbar\n");

return -1; // fail to create

}

if (!m\_wndStatusBar.Create(this) ||

!m\_wndStatusBar.SetIndicators(indicators,

sizeof(indicators)/sizeof(UINT)))

{

TRACE0("Failed to create status bar\n");

return -1; // fail to create

}

// TODO: Delete these three lines if you don't want the toolbar to

// be dockable

m\_wndToolBar.EnableDocking(CBRS\_ALIGN\_ANY);

m\_wndToolBar1.EnableDocking(CBRS\_ALIGN\_ANY);

EnableDocking(CBRS\_ALIGN\_ANY);

DockControlBar(&m\_wndToolBar);

DockControlBar(&m\_wndToolBar1);

return 0;

}

И как в прошлом примере реакцию на нажатие на диалоговой панели.

/////////////////////////////////////////////////////////////////////////////

// CTestMyToolsView message handlers

void CTestMyToolsView::OnCurcle()

{

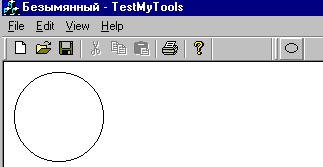
CDC\* pDC;

pDC=GetDC();

pDC->Ellipse(CRect(10,10,100,100));

}

Мы с Вами получили настоящую диалоговую панель. Эту панель можно таскать и прислонять к стенкам. Все можно кроме как убирать с экрана. :-))))



[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/95.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?94) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?96) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc2.html)  
Автор **Каев Артем**.

## Шаг 96 - Кисть

29 января на Северо-Востоке России показалось Солнце.

Поляной ночи конец !!!!!!

Для рисования графики **GDI** предлагает два инструмента кисти и перья. Они вроде похожи, а и вроде нет. Главное отличие в том, что перо позволяет рисовать линии заданной толщины, а кисть заливает замкнутую фигуру. По русски закрашивает. Кисти можно делать любые, сплошные, трафаретные и даже из растровых файлов !!!. Ну к делу ?

Создавайте приложение **Win 32 MFC App Wizard**, скажите **SDI** и всё, как говорят **Finish**. И хотя тип окон значения не имеет, пусть будет просто **SDI**. Нам в **CTestBrushView**, там наш любимый **OnDraw**, в котором мы и будет творить.

Итак, чего я хочу ? Сделать закрашенный прямоугольник. Вот чего я хочу !!!!. Начинаем.

Надо сделать класс пера для контура. Параметры описывать не буду, не о том речь.

CPen pen(PS\_SOLID, 5, RGB(0,0,255));

Выберем перо и сохраним старое

CPen\* oldPen = pDC->SelectObject(&pen);

Теперь класс кисти.

CBrush brush(HS\_BDIAGONAL, RGB(255,0,0));

У него несколько конструкторов в зависимости от того, чего Вам надо. А чего Вам надо ? Мне штриховку лично.

CBrush( ); // просто так ;-)

CBrush( COLORREF crColor ); // сплошным цветом

throw( CResourceException );

CBrush( int nIndex, COLORREF crColor ); // со штриховкой

throw( CResourceException );

CBrush( CBitmap\* pBitmap ); // растровой картинкой

throw( CResourceException );

Как видите я выбрал третий вариант со штриховкой. Все варианты шриховок смотрите в [подробностях](http://www.firststeps.ru/mfc/detail/17.html).

Выбираем для рисования кисть.

CBrush\* oldBrush =pDC->SelectObject(&brush);

Пора рисовать

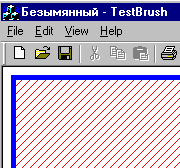
pDC->Rectangle(10,10,200,200);

Теперь надо все вернуть на место. Меня мама в детстве учила: "Взял, потом положи на место". Так и не научила, теперь жена учит :-)))

pDC->SelectObject(oldPen);

pDC->SelectObject(oldBrush);

Можно собирать и запускать. А вот и результат.



[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/96.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?95) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?97) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc2.html)  
Автор **Каев Артем**.

## Шаг 97 - Структура окна на шаг впереди MFC

При классическом программирование на **C** под **Windows**(это ужас честное слово) для того, чтобы окно появилось на экране необходимо заполнить структуру **CREATESTRUCT**. Вот её описание

typedef struct tagCREATESTRUCT {

LPVOID lpCreateParams;

HANDLE hInstance;

HMENU hMenu;

HWND hwndParent;

int cy;

int cx;

int y;

int x;

LONG style;

LPCSTR lpszName;

LPCSTR lpszClass;

DWORD dwExStyle;

} CREATESTRUCT;

Полное её описание описание надо делать отдельно. Но догадаться о назначении пуктов можно и так. Например **cx** и **cy** отвечают за высоту и ширину. Если вы хотите пообщаться с этой структурой, то вы должны перехватить её до начала создания окна и вызова функции **CreateWindow**. **MFC** вообще-то сам умеет задавать там параметры, вопрос в том устраивают они Вас или нет. Для того, чтобы получить доступ к этой структуре Вам нужно перегрузить **PreCreateWindow** из класса **CMainFrame**. Туда передается указатель на эту структуру и Вы сможете её подправить.

Теория ясна к делу теперь. Итак создаем **MFC AppWizard** приложение **TestCreate** как **SDI**. Сейчас функция вот такая и в её параметрах есть указатель на нашу структуру.

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)

{

if( !CFrameWnd::PreCreateWindow(cs) )

return FALSE;

// TODO: Modify the Window class or styles here by modifying

// the CREATESTRUCT cs

return TRUE;

}

Правим ????

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)

{

if( !CFrameWnd::PreCreateWindow(cs) )

return FALSE;

cs.cx = 50;

cs.cy = 50;

return TRUE;

}

Попробуйте :-))) Будет маленькое такое окно !!!

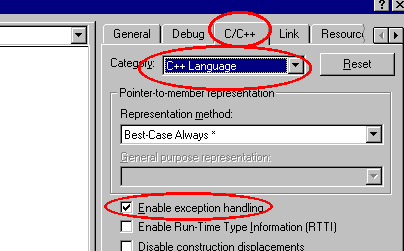
[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?96) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?98) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc2.html)  
Автор **Каев Артем**.

## Шаг 98 - Как используются исключения в VC и MFC

Есть два механизма исключений.

* Исключения, как операторы С++.
* Макросы исключений

Макросы исключений считаются устаревшими, для всех новых программ рекомендуют использовать только операторы **C++**. Правда для того, чтобы в Вашей программе использовались исключения необходимо их разрешить в меню **Project Setting** в разделе **C/C++** должен быть установлен флаг **Enable exception handing**.



Для обработки исключений **MFC** используется класс **CException** или классы поражденные от него. Для обработки исключительных ситуаций в **MFC** есть свои классы

CMemoryException недостаточно памяти

CFileException ошибка работы с файлами

CArhiveException ошибка записи или востановления обьекта

CNotSupportedException ошибка неизвестного устройства

CResourceException ошибка работы с ресурсами

CDaoException ошибка работы с базами данных

CDBException ошибка рабюоты с базами данных

COleException ошибка работы с обьектами OLE

COleDispatchException ошибка работы с серверами автоматизации

CUserException исключение с окном сообщения

К сожалению во всем этом есть неприятность. Механизмы работы исключений **MFC** не работают со структурированными обработчиками исключений **SEH** для **NT**. Следует отметить, что объекты **MFC** порождают исключения только производные от класса **CException**. Вот пример отлавливания исключения по нехватке памяти.

try

{

// здесь мы эту память......

// короче на лево и направо раздаем

}

catch(CMemoryException)

{

// а вот здесь мы напишем

// что будет делать потом

// когда она закончится

}

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?97) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?99) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc2.html)  
Автор **Каев Артем**.

## Шаг 99 - Как обрабатывать исключения

Естественно, что исключения нарушает нормальный ход программы и к этому надо быть готовым заранее. Главный принцип это обрабатывать исключения локально. Смотрите пример.

void MyFuct()

{

CPen \*cp=new CPen(255,0,0)

// --- код ----

......

ИСКЛЮЧЕНИЕ

......

// -------------

delete cp;

}

В данном примере при появлении исключения память выделенная переменной **cp** не будет освобождена !!!. При вызове исключения обработка будет передана куда-то дальше в обработчик и работа функции не будет завершена. Поэтому при исключении **delete** не будет вызван. Давайте изменим код ближе к правильному.

void MyFuct()

{

CPen \*cp=new CPen(255,0,0)

try

{

// --- код ----

......

ИСКЛЮЧЕНИЕ

......

// -------------

}

catch(CException\* e)

{

e->Delete();

}

delete cp;

}

Здесь мы вроде как всё предусмотрели. И объект удалили и исключение обработали и удалили. Что делать если исключение Вы не хотите обрабатывать локально ? Вот некоторый выход.

........

catch(CException\* e)

{

delete cp;

throw;

}

delete cp;

Да, напоследок механизм исключений автоматически уничтожает оконные объекты. Это где-то в книге мне попадалось только не помню в какой.

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?98) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?100) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc2.html)  
Автор **Каев Артем**.

## Шаг 100 - Коллекции в MFC

В **MFC** есть три коллекции - это массивы **CArray**, списки **CList** и словари **CMap**. Кроме того, классы делят на созданные с помощью шаблонов или без них. Конечно на данный момент стандарт **C++** требует использования шаблонов. Все коллекции порождены от **CObject**

**Массив** - упорядоченный набор объектов, каждому из которых соответствует индекс.

**Список** - упорядоченный неиндексированный набор элементов с виде связанного списка. В списке доступ может быть к первому, последнему и текущему элементу, для остальных необходимо производить поиск. Поиск проводится по указателю на последний и предыдущий элемент списка, который хранится в каждом элементе в виде указателя. Удобна для быстрого дополнения элементов.

**Словарь** - для доступа используется ключевое поле.

Есть коллекции объектов и коллекции указателей. При использовании коллекции имя меняется путем добавления вставки **TypedPrt**, например, для **CArray** это **CTypedPtrArray**. И так далее.

Для выбора конкретной коллекции нужно учесть кучу параметров, вот некоторые логические

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Упорядоченность** | **Индексируемость** | **Скорость вставки** | **Поиск** |
| **Список** | Да | Нет | Быстро | Медленно |
| **Массив** | Да | Да | Медленно | Быстро |
| **Словарь** | Да | Да | Быстро | Быстро |

Анализ этой таблицы показывает, что не всё коту масленница. Если хотим иметь всё, то нужен словарь. Но это не всегда удобно, а дальше выбор простой, что нужно быстро искать или быстро вставлять :-))). Так что Выбор за Вами. Вот еще некоторые параметры для размышления

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Шаблоны С++** | **Работа с архивом** | **Встроеная диагностика** | **Проверка типа** |
| **Массив** | Да | Да | Да | Нет |
| **Список** | Да | Да | Да | Нет |
| **Словарь** | Да | Да | Да | Нет |

Вы скажете "Ну и что здесь такого? Они же одинаковые". Конечно, но у каждой коллекции есть специализированные разновидности на определенный тип данных. Например, для **CArray**

CArray

CByteArray

CDWordArray

CObArray

CPtrArray

CStringArray

CTypedPtrArray

CUintArray

CWordArray

И вот тут-то всё и начинается. Например, с архивом не может работать **CPtrArray**, а **CByteArray** может проверять типы :-))) Поэтому при выборе обязательно проверьте, а умеет ли Ваш контейнер делать то, что вы скажете ?.

Объявления коллекций находятся в заголовочном файле

#include "afxtempl.h"

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?99) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?101) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc3.html)  
Автор **Каев Артем**.

## Шаг 101 - Переводим SDI в MDI

Ну что, давайте сгенирируем приложение **SDI** с именем **SDIMDI**. Запускаем **AppWizard** сразу после выбора **SDI** жмем **Finish**. Готово, теперь пояснения. Я сделаю проект и Вы сможете его загрузить. Все изменения я буду коментировать вот так. Это на случай если в описании забуду или еще чего. Если Вы уже шаги делали, то понимаете о чем разговор :-)))

// ----- Comments SDI -------

.......

// ----- End Comments SDI ---

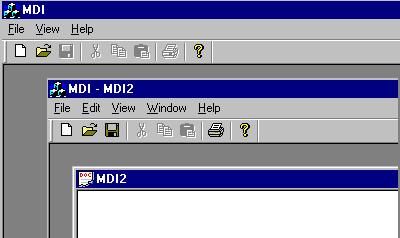
// ------ New MDI -----------

.....

// -----End New MDI ---------

Отличия **MDI** и **SDI** простые: **MDI** много окон, а в **SDI** одно окно. У нас пока окно одно. Надо много. Задача ясна.

Первое отличие это то, что у приложения **MDI** два меню. Первое меню имеет имя **IDR\_MAINFRAME** и используется когда дочернии окна не открыты. Второе меню появляется когда открывается или появляется окно. Оно имеет имя типа **IDR\_ИМЯ\_ПРОЕКТTYPE**. Смотрите рисунок ниже.



Раз такое дело, то и нам надо два меню. Давайте его создадим, имя его будет исходя из правил **IDR\_SDIMDITYPE**. Идем в ресурсы и создаем. Только создаем не просто так. Наводим мышь на **IDR\_MAINFRAME** и говорим **Inset Copy**. В поле **Condition** введите просто букву **a**. Появится копия. Потом вызовите свойства и дайте имя **IDR\_SDIMDITYPE**, а из поля **Condition** удалите **a**. Теперь из **IDR\_MAINFRAME** нужно удалить ненужные пункты меню.

SAVE

SAVE AS

PRINT

PRINT PRIVIEW

RESENT FILE

Лишнии разделительные линии

EDIT полностью

В принципе нам бы надо создать меню управления окнами, но черт с ним пока. Каждой строке меню нужен строковый ресурс. Вот теперь пойдите в строковые ресурсы скопируйте и вставьте **IDR\_MAINFRAME** и переименуйте в **IDR\_SDIMDITYPE**.

Теперь нужен еще и значок для окна. Он должен быть с тем же идентификатором. Создайте иконку с идентификатором **IDR\_SDIMDITYPE**. Он у Вас вообще-то должен был появиться в виде **DOC1**. Посмотрите. Вот добрались и до кода. Первое - это нужно сменить класс рамки окна на **CMDIFrameWnd**.

//----- Comments SDI -------

//class CMainFrame : public CFrameWnd

//----- End Comments SDI ---

//------ New MDI -----------

class CMainFrame : public CMDIFrameWnd

// -----End New MDI ---------

CMainFrame() нужно сделать публичным.

protected: // create from serialization only

//----- Comments SDI -------

// CMainFrame();

//----- End Comments SDI ---

DECLARE\_DYNAMIC(CMainFrame)

// Attributes

public:

//------ New MDI -----------

CMainFrame();

// -----End New MDI ---------

// Operations

Обратили внимание на **DECLARE\_DYNAMIC**, а не так как в **SDI**. Теперь нам нужно изменить файл реализации, вот где.

//----- Comments SDI -------

//IMPLEMENT\_DYNCREATE(CMainFrame, CFrameWnd)

//----- End Comments SDI ---

//------ New MDI -----------

IMPLEMENT\_DYNAMIC(CMainFrame, CMDIFrameWnd)

// -----End New MDI ---------

Теперь нужно изменить создание шаблона.

//----- Comments SDI -------

//CSingleDocTemplate\* pDocTemplate;

//pDocTemplate = new CSingleDocTemplate(

//IDR\_MAINFRAME,

//----- End Comments SDI ---

//------ New MDI -----------

CMultiDocTemplate\* pDocTemplate;

pDocTemplate = new CMultiDocTemplate(

IDR\_SDIMDITYPE,

// -----End New MDI ---------

RUNTIME\_CLASS(CSDIMDIDoc),

RUNTIME\_CLASS(CMainFrame), // main SDI frame window

RUNTIME\_CLASS(CSDIMDIView));

AddDocTemplate(pDocTemplate);

Пора создавать класс рамки окна.

// Parse command line for standard shell commands, DDE, file open

//----- Comments SDI -------

//CCommandLineInfo cmdInfo;

//ParseCommandLine(cmdInfo);

// Dispatch commands specified on the command line

//if (!ProcessShellCommand(cmdInfo))

// return FALSE;

//----- End Comments SDI ---

//------ New MDI -----------

CMainFrame\* pMain = new CMainFrame;

if (!pMain->LoadFrame(IDR\_MAINFRAME))

return FALSE;

m\_pMainWnd=pMain;

// -----End New MDI ---------

// The one and only window has been initialized, so show and update it.

m\_pMainWnd->ShowWindow(SW\_SHOW);

m\_pMainWnd->UpdateWindow();

return TRUE;

Поменяем в шаблоне класс рамки окна

//----- Comments SDI -------

// RUNTIME\_CLASS(CMainFrame), // main SDI frame window

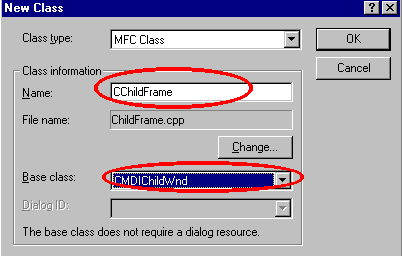
//----- End Comments SDI ---

//------ New MDI -----------

RUNTIME\_CLASS(CChildFrame),

// -----End New MDI ---------

Но у нас его нет !!! Создадим. Перейдите на вкладку **ClassView**, далее **NewClass**.



И естественно, что этот класс нужно подключить в смысле заголовочного файла:

#include "SDIMDIDoc.h"

#include "SDIMDIView.h"

//------ New MDI -----------

#include "ChildFrame.h"

// -----End New MDI ---------

Всё вроде, проверяйте. Если не работает грузите проект.

Итак кратко.

Два меню, иконы, строковых ресурса.

Меняем класс рамки на MDI

Меняем шаблон на MDI

Меняем рамку шаблона с ChildMDI

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/101.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?100) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?102) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc3.html)  
Автор **Каев Артем**.

## Шаг 102 - Создаем WMF файл

Итак, дошла очередь и до **WMF**. Что эа такое **WMF** файлы ? Работая с графикой рано или позно у Вас возникнет задача в передаче информации от одной программы к другой. Существует большое количество стандартов, например, **DXF**. Для его использования необходимо тщательно изучить стандарт и реализовать его поддержку. Существуют и другие стандарты. Но изучение стандартов это дела долгое. Что делать еcли надо, чтобы ваша программа могла быстро и с наименьшими трудностями для Вас сохранить данные. Для этого нужно воспользоваться стандартом **Windows - WMF**. На Ваш вопрос о том, чем он отличается от **DXF**, например, я могу сказать, что стандарта **WMF** практически нет :-)). То есть так же как вы рисуете с помощью **GDI** теми же командами и пишите в **WMF** файл. Информация в файле будет векторная. **WMF** - это список структур, которые содержат информацию о вызове **GDI** функций. Ладно на примере думаю будет понятно :-)

Создаем приложение как **MFC AppWizard** с именем **WmfTest**, как **SDI** и нажимаем **Finish**. Теперь создадим функцию рисования в **OnDraw**.

/////////////////////////////////////////////////////

// CWmfTestView drawing

void CWmfTestView::OnDraw(CDC\* pDC)

{

CWmfTestDoc\* pDoc = GetDocument();

ASSERT\_VALID(pDoc);

// TODO: add draw code for native data here

CPen pen(PS\_SOLID,4,RGB(255,0,0));

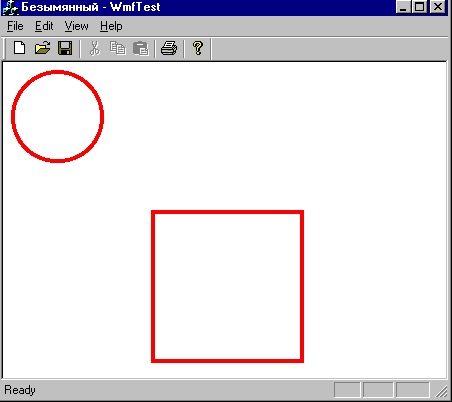
pDC->SelectObject(pen);

pDC->Ellipse(CRect(10,10,100,100));

pDC->Rectangle(CRect(150,150,300,300));

}

Запустим программу и посмотрим.



Для записи в **WMF** файл создадим функцию реакции на нажатие левой кнопки мыши в окне. То есть нажали кнопку мыши информация в **WMF** файл записалась.

///////////////////////////////////////////////////////

// CWmfTestView message handlers

void CWmfTestView::OnLButtonDown(UINT nFlags, CPoint point)

{

HDC MetaDC;

MetaDC=CreateMetaFile("test.wmf");

if (MetaDC!=NULL)

{

CPen pen(PS\_SOLID,4,RGB(255,0,0));

SelectObject(MetaDC,pen);

Ellipse(MetaDC,10,10,100,100);

Rectangle(MetaDC,150,150,300,300);

}

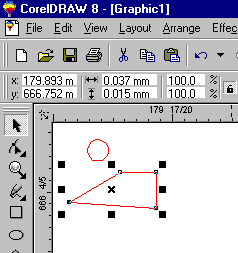
CloseMetaFile(MetaDC);

CView::OnLButtonDown(nFlags, point);

}

Смотрите, создаем контекст метафайла **HDC MetaDC**, после этого сам метафайл **CreateMetaFile("test.wmf")** и теперь мы используя контекст метафайла и функции **GDI** из **API** !!! спокойно в этот контекст пишем. В конце закрываем метафайл. Готов !.

А теперь проверяем. Запустите приложение. Щелкните левой кнопкой. И экспортируйте файл **test.wmf** в **Corel Draw 8.0**. Рисунок будет маленький. Увеличьте его. Разгруппируйте **Ungroup**. Щелкните в стороне. Теперь это два векторных объекта. На рисунке ниже я специально переместил один узел, чтобы это показать.



[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/102.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?101) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?103) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc3.html)  
Автор **Каев Артем**.

## Шаг 107 - Добавим значек в область SysTray

От: Aleksander Romanovsky

Кому: kaev@yandex.ru

Дата: 10 февраля 2000 г. 13:15

Тема: Можно вопросик...

Добрый день.

Не подскажешь как избавиться от пиктограммы в панели задач.

Я перехватил сообщение минимизации из системной панели,скинул

пиктограмму в SysTray,но вот убрать пиктограмму из панели задач

не получается.

Заранее благодарен.

Alex.

От: Andy

Кому: kaev@yandex.ru

Тема: MFCByStep

Дата: 4 февраля 2000 г. 14:06

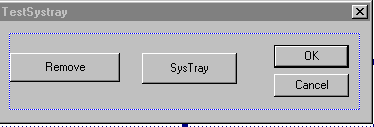
Привет

Ты не знаешь как сделать, чтобы диалоговое приложение под MFC при запуске не

открывала диалог а появлялась в нижнем правом углу в виде иконки?

andy

Итак, если я не ошибаюсь, то нижнее правое окошко это область имеющая название **SysTray**. Туда можно помещать иконки программ. Создаем приложение **MFC AppWizard** на основе диалогового окна. Вырубаем все лишнее. Теперь на диалоговое окно помещаем две кнопки и создаем функции их обработки. Вот так.



Пора делать код для кнопок. Этот для кнопки **SysTray**, которая поместит значек в эту область.

void CTestSystrayDlg::OnButton1()

{

NOTIFYICONDATA nf;

nf.hWnd = m\_hWnd;

nf.uID = NULL;

nf.uFlags = NIF\_ICON;

nf.uCallbackMessage = NULL;

HICON hIcon;

hIcon=AfxGetApp()->LoadIcon(IDR\_MAINFRAME);

nf.hIcon = hIcon;

Shell\_NotifyIcon(NIM\_ADD,&nf);

}

А этот код для кнопки **Remove**.

void CTestSystrayDlg::OnButton2()

{

NOTIFYICONDATA nf;

nf.hWnd = m\_hWnd;

nf.uID = NULL;

nf.uFlags = NIF\_ICON;

nf.uCallbackMessage = NULL;

nf.hIcon = NULL;

Shell\_NotifyIcon(NIM\_DELETE,&nf);

}

Мы используем функцию **Shell\_NotifyIcon**, которая добавляет, модифицирует и удаляет иконы из панели задач. Вот её описание.

WINSHELLAPI BOOL WINAPI Shell\_NotifyIcon(

DWORD dwMessage,

PNOTIFYICONDATA pnid

);

Нам нужно указать команду и заполнить структуру типа **NOTIFYICONDATA**. В ней указать наш **HWND**, установить флаги, указать ссылку на икону, что я и делаю.

Давайте испытаем. Нажмите кнопку **SysTray** и значек появится, нажмите **Remove** и значек удалится.



[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/107.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?106) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?108) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc3.html)  
Автор **Каев Артем**.

## Шаг 108 - Развиваем SysTray

Вы обращали внимание на то, что у всех икон в **SysTray** существует подсказка ? Для её существования необходимо в структуру заполнить поле посказки, правильно выставить флаги.

void CTestSystrayDlg::OnButton1()

{

NOTIFYICONDATA nf;

nf.hWnd = m\_hWnd;

nf.uID = NULL;

nf.uFlags = NIF\_ICON | NIF\_MESSAGE | NIF\_TIP;

nf.uCallbackMessage = WM\_MYICONNOTIFY;

strcpy(nf.szTip,"Hello Systray");

HICON hIcon;

hIcon=AfxGetApp()->LoadIcon(IDR\_MAINFRAME);

nf.hIcon = hIcon;

Shell\_NotifyIcon(NIM\_ADD,&nf);

}

Я хочу и получать сообщения в окно при двойном щелчке на кнопке. Для этого необходимо завести свое сообщение.

//{{NO\_DEPENDENCIES}}

// Microsoft Developer Studio generated include file.

// Used by TestSystray.rc

//

........

#define IDC\_BUTTON2 1001

#define WM\_MYICONNOTIFY WM\_USER+1

// Next default values for new objects

//

.......

Передать его в структуру.

nf.uCallbackMessage = WM\_MYICONNOTIFY;

Описать реакцию на него в карте сообщений

BEGIN\_MESSAGE\_MAP(CTestSystrayDlg, CDialog)

......

//}}AFX\_MSG\_MAP

ON\_MESSAGE(WM\_MYICONNOTIFY,OnIcon)

END\_MESSAGE\_MAP()

Описать функцию, которая будет вызвана в ответ на это сообщение

/////////////////////////////////////////////////

// CTestSystrayDlg dialog

class CTestSystrayDlg : public CDialog

{

// Construction

public:

void OnIcon(WPARAM wp, LPARAM lp);

......

};

И её реализацию проверим **LPARAM** на тип сообщений:

void CTestSystrayDlg::OnIcon(WPARAM wp, LPARAM lp)

{

if (lp==515) AfxMessageBox("Click");

}

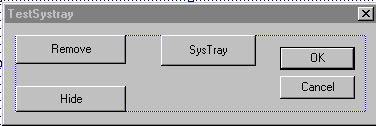
Код этой функции будет вызываться на любое сообщение. А вот 512 - это двойной щелчок.

Собирайте приложение. Создайте иконку в **SysTray** щелкните на неё два раза и ваше диалоговое окно среагирует.

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/108.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?107) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?109) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc3.html)  
Автор **Каев Артем**.

## Шаг 109 - Прячем окно в SysTray

Итак, чтобы получить нормальное приложение, которое прячется в **SysTray** нам просто надо его убрать с экрана. Создадим кнопку **HIDE** на диалоговом окне.



Напишем к ней код.

void CTestSystrayDlg::OnButton3()

{

ShowWindow(SW\_HIDE);

}

Изменим код по нажатию на на иконку.

void CTestSystrayDlg::OnIcon(WPARAM wp, LPARAM lp)

{

if (lp==515) ShowWindow(SW\_RESTORE);

}

Всё можете испытывать. Запустите. Нажмите кнопку **SysTray** - появится иконка. Нажмите кнопку **Hide** - окно пропадет вообще. Щелкните по иконке - окно появится. Теперь эти процедуры Вы можете комбинировать в произвольном порядке.

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/109.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?108) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?110) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc3.html)  
Автор **Каев Артем**.

## Шаг 110 - Готовим данные для записной книжки

-----Исходное сообщение-----

От: Leonid

Кому: kaev@yandex.ru

Дата: 12 февраля 2000 г. 20:34

Тема: MFC by step

>MFC by step безусловно - вещь, а особенно защита от скачивания ;)

>хотелось бы посмотреть пример какого нибудь законченй простенький проект например

>для работы с базой данных.

>Подойдет для начала даже база Access - что нибудь типа записной книжии

>--

>Best regards,

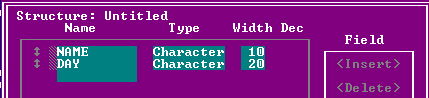
> Leonid mailto:leo@softincom.ru

>

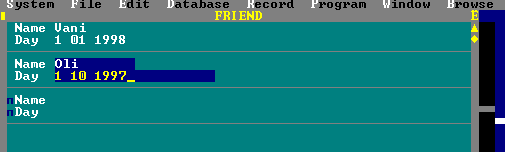
>

Это конечно хорошо, но я уже расказывал. Давайте сделаем полный проект скажем имена и дни рождения друзей на основе **DBF** формата для **DBASE**.

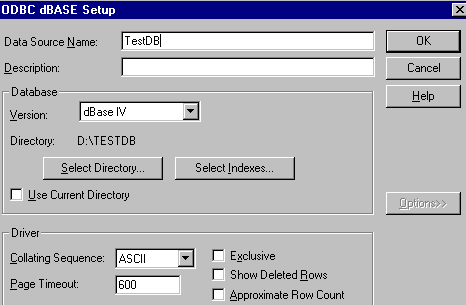
Первое, что надо сделать это подготовить данные. Я это сделаю в **FoxPro**. Если не знаете как смотрите **FoxPro 2.0** на сайте. Имя файла будет **Friend.dbf** со следующей структурой.



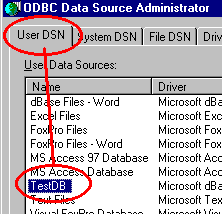
Теперь занесем две записи.



Этот файл будет идти в проекте. Но **ODBC** придется настроить. Я уже писал как это делать, так что кратко. В "Мой компьютер" -> "панель управления" -> "ODBC 32 Source" -> "User DSN", там есть кнопка **Add**, жмите её. Выбирите **Microsoft dBase Driver** и скажите готово. Появится окно. В **Version** поставьте **dBase IV**, отключите опцию **Use Current Directory**. Теперь надо решить, где будет лежать файл. У меня на диске **D:\TestDB**. Вот теперь нужно выбрать директорию, жмите **Select Directory**. Теперь нужно дать имя источнику данных **Data Source Name**. Дайте имя **TestDB**. И жмите **OK**.



Вот теперь вроде всё готово к работе. То есть к созданию программы. Посмотрите еще раз **ODBC**, должно быть так.



[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/110.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?109) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?111) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc3.html)  
Автор **Каев Артем**.

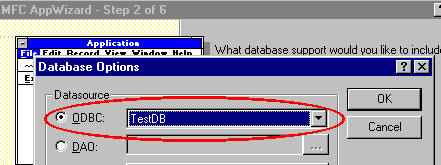
## Шаг 111 - Создаем программу

Запускайте **VC**. Выбираем **MFC AppWizard**, имя проекту даем **TestMemo**

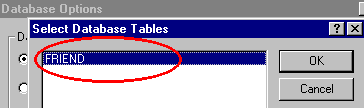
MCF AppWizard Step 1 - Single Document

MCF AppWizard Step 2 - Database view without file support

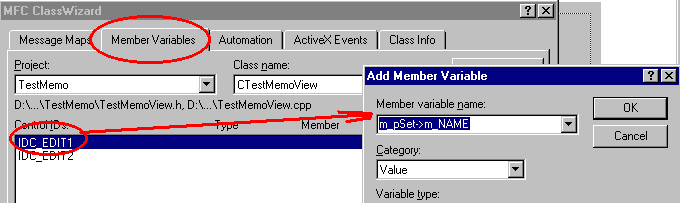
Теперь нам нужно установить источник записей. Жмите на **Data Source**. Появится окно **Database Options**, выберите в нем в **ODBC** источник данных **TestDB**.



Появится окно с предложением выбрать таблицу. Выберите нашу таблицу.

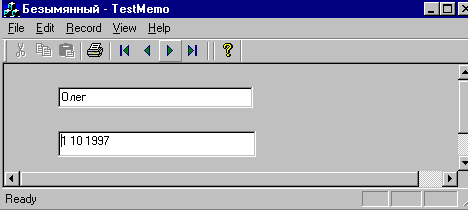


Теперь жмите **Finish**. У вас сгенирируется приложение с пустой формой. Разметим на ней два элемента **Edit** теперь нам надо связать их с полями таблицы. Выделите первый элемент и в меню **View** выберите **Class Wizard**. Нам нужна вкладка **Member Variables**. Жмите на **Add Variable** и выберите поле.



Жмите **OK**, теперь то же самое сделайте и для второго **EDIT**, только с другим полем.

Компилируйте и запускайте.



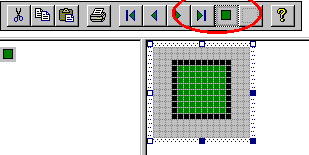
Теперь Вы можете редактировать поля. Отредактируйте и выйдите из приложения. Откройте **Friend.dbf** скажем в **Excel**.

Мы умеем редактировать поля, но не умеем заносить новые и удалять кстати. Исправим !!!

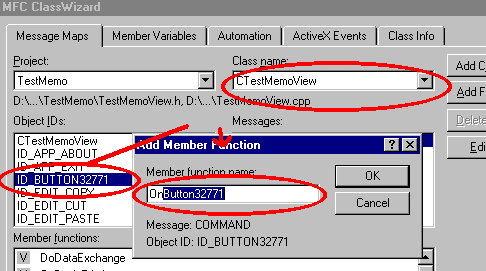
[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/111.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?110) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?112) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc3.html)  
Автор **Каев Артем**.

## Шаг 112 - Добавление записи

Продолжим изменять наш проект. Идем в ресурсы и смотрим там **ToolBar** и **IDR\_MAINFRAME**. Здесь нам нужно сделать кнопочку. Вот так. При нажатии на ней будет вставляться запись.



Закрываем **ResourceView** и идем в **ClassWizard**. Здесь нам нужна карта сообщений **Message Maps** для класса **CTestMemoView**. У нас там есть идентификатор кнопки. Вот по нему и надо создать функцию для реакции на нажатие.



У вас должна появиться функция.

////////////////////////////////////

// CTestMemoView message handlers

void CTestMemoView::OnButton32771()

{

// TODO: Add your command handler code here

}

Вот в неё мы и добавим, код создания новой записи у нас есть ссылка на объект **CRecordSet** её создал **AppWizard** и имя её **m\_pSet**.

////////////////////////////////////////

// CTestMemoView message handlers

void CTestMemoView::OnButton32771()

{

m\_pSet->AddNew();

m\_pSet->m\_NAME="Hello";

m\_pSet->Update();

m\_pSet->Requery();

UpdateData(FALSE);

}

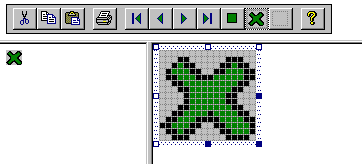
Вызываем функцию **AddNew**, которая умеет добавлять записи. После этого присваиваем записи начальное значение. Функция **Update()** физически вносит изменения в базу, а функция **Requery** перечитывает набор данных. То есть уже с нашей обновленной функцией. **UpdateData** говорит виду, что произошли изменения.

Можно испытывать. Запускаем программу. Щелкаем на нашей кнопке. Идем в конец. Появляется новая запись. Заносим в неё информацию. Работает !

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/112.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?111) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?113) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc3.html)  
Автор **Каев Артем**.

## Шаг 113 - Удаление записи

Продолжим изменять наш проект. Опять идем в ресурсы и делаем кнопочку удаления.



Создаем для этой функции код. На данный момент он пока такой.

void CTestMemoView::OnButton32772()

{

// TODO: Add your command handler code here

}

Нам надо удалить текущую запись. Вот давайте это и закодируем.

void CTestMemoView::OnButton32772()

{

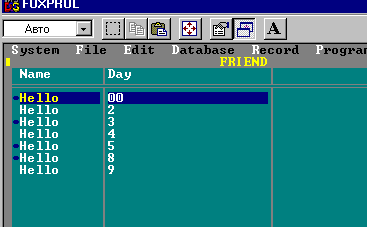
m\_pSet->Delete();

m\_pSet->Requery();

UpdateData(FALSE);

}

Код очень похож, удаляем запись, перечитываем выборку, обновляем вид. Но есть один нюанс. Эта тонкость есть и при работе с базами данных на **FoxPro** или **DBase**. Записи физически не удаляются, а только помечаются на удаление. Поработайте с программой и посмотрите файл в **FoxPro**

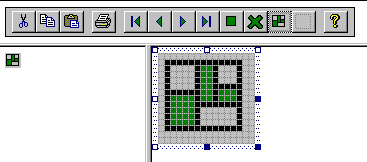


Видите точки ??? Это помеченные записи на удаление. То есть они не видны. Но они удалены физически.

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/113.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?112) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?114) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc3.html)  
Автор **Каев Артем**.

## Шаг 114 - Фильтр

Итак, выбрать необходимые записи можно многими путями, один из таких путей это фильтр. Давайте создадим для него кнопку и код к ней.



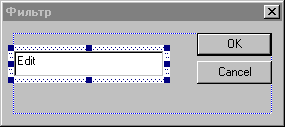
void CTestMemoView::OnButton32773()

{

// TODO: Add your command handler code here

}

Теперь надо решить как вводить фильтр. Давайте сделаем для него диалоговое окно из элемента **Edit**. Идея такова ввел слово и по полю **Name** отфильтровали записи. Создаем диалоговое окно.



Теперь для этого окна нужен класс. Выделите диалогвое окно и запустите **ClassWizard**. Вам в ответ должны предложить создать класс, а вы согласиться. Когда спросят имя дайте ему имя **СFiltrDialog**. Теперь нужно создать переменную для элемента **Edit**. Опять выделите её и запустите **ClassWizard**. Переключитесь на **Memder Variable**, щелкните на **Add Variable** и дайте имя **m\_Filtr**. Теперь в файле реализации **TestMemoView.cpp** необходимо подключить заголовок диалогового окна. Вот так.

.........

#include "TestMemoView.h"

#include "FiltrDialog.h"

#ifdef \_DEBUG

.......

Вот теперь все приготовления сделаны и можно кодировать.

void CTestMemoView::OnButton32773()

{

CFiltrDialog cf;

if (cf.DoModal()==IDOK)

{

CString filt="NAME = '";

filt=filt+cf.m\_Filtr;

filt= filt +"' ";

m\_pSet->m\_strFilter= filt;

m\_pSet->Requery();

UpdateData(FALSE);

}

}

Сначала мы собираем строку для фильтра, она должна быть такая

ПОЛЕ = 'ЗНАЧЕНИЕ'

Устанавливаем **m\_strFilter**, а дальше как обычно перечитать и обновить.

Когда будете проверять сделайте несколько групп одинаковых записей по полю **NAME**. У меня работает.

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/114.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?113) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?115) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc3.html)  
Автор **Каев Артем**.

## Шаг 115 - Напечатаем ???

При обычном предварительном просмотре изображение формируется функцией **OnDraw**. В данном случае так сделать мы не можем. Ну не сможем и ладно. Можно по другому. Для начала установим количество страниц. У нас она будет одна. Я уже писал как это делать.

void CTestMemoView::OnBeginPrinting(CDC\* pDC, CPrintInfo\* pInfo)

{

pInfo->SetMaxPage(1);

};

Куда же нам поместить печать карточки ???? Есть куда !!! В **OnPrint** мы спокойно можем поместить. Там тоже можно печатать и не только колонтитулы. Помещаем.

void CTestMemoView::OnPrint(CDC\* pDC, CPrintInfo\* pInfo)

{

pDC->Rectangle(10,10,1000,1000);

pDC->TextOut(40,100,m\_pSet->m\_NAME);

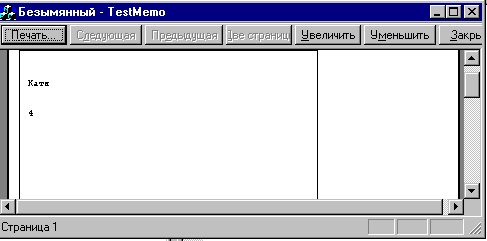
pDC->TextOut(40,200,m\_pSet->m\_DAY);

CRecordView::OnPrint(pDC, pInfo);

};

Как видите всё оказалось по идиотски просто.... Мы ссылаемся на наш **m\_pSet** и получаем поля текущей записи. Вся сложность была в том где печатать.

Это практически всё :-) Запускаем и смотрим.



А под конец маленький прикол в плане того, что в **Microsoft** работают тоже люди. Давайте проведем эксперимент. Закоментируйте установку количества страниц.

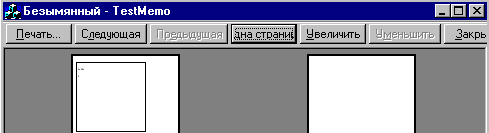
void CTestMemoView::OnBeginPrinting(CDC\* pDC, CPrintInfo\* pInfo)

{

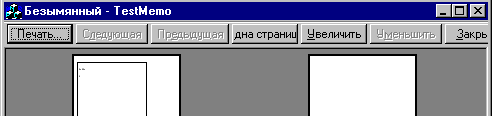
// pInfo->SetMaxPage(1);

}

Запустите приложение и скажите, что будете смотреть две страницы.



Всё нормально. Количество страниц не определенно. Так теперь раскоментируйте установку, скомпилируйте и запустите приложение снова. Посмотрите на предварительном просмотре.



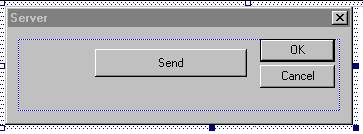
Каково ???? :-)))) Если Вы нажмете одна страница, то она погаснет навсегда. Это доказывает, что было задумано не так, а данный момент просто проспали ;-)))

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/115.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?114) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?116) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc3.html)  
Автор **Каев Артем**.

## Шаг 116 - Посылка сообщений между программами

Итак, наша задача создать две программы. Одна будет сервер и будет посылать сообщения, а вторая будет клиент естественно и будет эти сообщения принимать. Итак, общая идея такова. Можно посылать сообщение используя **HWND** другого окна. Для этого есть функция **SendMessage**. Она пошлет любое сообщение любому окну, которому скажете. Отловить сообщение можно используя карту сообщения **ON\_MESSAGE**. Давайте делать сервер.

Запускайте **AppWizard**, теперь создавайте приложение на базе диалогового окна отключив всё кнопки для простоты. На диалоговом окне давайте сделаем кнопку. И процедуру по нажатию на неё тоже надо.



Определим наше сообщение.

...........

#define WM\_MYMESSAGE WM\_USER+1

//////////////////////

// CServerDlg dialog

............

Теперь можно создавать код по нажатию на кнопку.

void CServerDlg::OnButton1()

{

HWND hwnd;

hwnd=::FindWindow(NULL,"Client");

if (hwnd != NULL)

{

::SendMessage(hwnd,WM\_MYMESSAGE,0,0);

}

}

Первое, что я делаю это использую функции **API**, а не **MFC**. Для этого ставлю две точки.

// ЭТО ФУНКЦИЯ API !!!! а не перегруженная MFC

::SendMessage(hwnd,WM\_MYMESSAGE,0,0);

Для посылки сообщения мне нужно найти окно клиента - это делает **FindWindow**. В первом параметре необходимо указать имя класса окна, которое указывается в **CreateStruct** при создании окна. Я его не указал. Второй параметр имя заголовка окна. Это грозит некоторой опасностью в случае, если имена заголовка окна будут совпадать, т.к. сообщение пошлется всем окнам с таким заголовком. Но это можно повернуть и в свою пользу. В общем я так сделал для простоты. А вы сами решайте. Если окно найдено ему можно послать сообщение **SendMessage**, указав кому мы какое сообщение посылаем. Я посылаю своё сообщение и без параметров. Вот и все теперь клиент.

Для клиента создадим тоже проект **MFC AppWizard**, но только без кнопки. Точно такой же. Опишем реакцию на наше сообщение.

BEGIN\_MESSAGE\_MAP(CClientDlg, CDialog)

//{{AFX\_MSG\_MAP(CClientDlg)

ON\_WM\_PAINT()

ON\_WM\_QUERYDRAGICON()

//}}AFX\_MSG\_MAP

ON\_MESSAGE(WM\_MYMESSAGE,OnMyMessage)

END\_MESSAGE\_MAP()

И функцию соотвественно.

class CClientDlg : public CDialog

{

// Construction

public:

void OnMyMessage();

//......

};

И её реализация естественно.

void CClientDlg::OnMyMessage()

{

MessageBeep(-1);

}

Запускайте два приложения. Щелкайте по кнопке на сервере и клиент будет вам отвечать звуком. Вот они и нашли общий язык.

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/116.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?115) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?117) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc3.html)  
Автор **Каев Артем**.

## Шаг 117 - Обмен данными с использованием WM\_COPYDATA

Мы с Вами создавали приложение клиента и сервера, которые обмениваются сообщениями. Теперь мы модернизируем данные приложения для обмена данными. В операционной системе **Windows** есть большое количество возможностей для обмена данными. Один из таких методов использование сообщения **WM\_COPYDATA**. Это сообщение в **Windows NT** даже может передать данные через границу процессов.

Давайте изменим наш сервер. Сделаем еще одну кнопку для посылки этого сообщения. По нажатию на кнопку, пока будем посылать просто сообщение без параметров.

void CServerDlg::OnButton2()

{

HWND hwnd;

hwnd=::FindWindow(NULL,"Client");

if (hwnd != NULL)

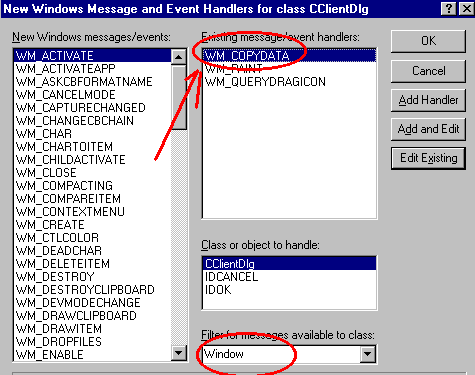
{

::SendMessage(hwnd,WM\_COPYDATA,0,0);

}

}

Теперь клиент должен научиться принимать это сообщение. А здесь нам может помочь **Class Wizard**, он уже умеет создавать процедуру приема данного сообщения. Нажмите правую кнопку на **CClientDlg** и выберите пункт меню **Add Windows Message Handles**.



В результате появится вот такая функция. **MessageBeep** вписал уже я для того, чтобы почувствовать сообщение :-)))

BOOL CClientDlg::OnCopyData(CWnd\* pWnd, COPYDATASTRUCT\* pCopyDataStruct)

{

MessageBeep(-1);

return CDialog::OnCopyData(pWnd, pCopyDataStruct);

}

Уже сейчас можно запустить и проверить, что данный механизм работает. Ну что теперь надо идти на сервер и готовить данные к отправке. Пошли !!!

Изменяем код отправки сообщения.

void CServerDlg::OnButton2()

{

HWND hwnd;

hwnd=::FindWindow(NULL,"Client");

COPYDATASTRUCT cd;

char buffer[25];

strcpy(buffer,"Hello Client");

cd.lpData=&buffer;

cd.cbData= sizeof(buffer);

if (hwnd != NULL)

{

::SendMessage(hwnd,WM\_COPYDATA, 0, (LPARAM) &cd);

}

}

Как видите нам нужно заполнить структуру **COPYDATASTRUCT**, указав адрес буфера и размер. После чего адрес структуры передать в сообшение.

Теперь нужно изменить клиента.

BOOL CClientDlg::OnCopyData(CWnd\* pWnd, COPYDATASTRUCT\* pCD)

{

MessageBeep(-1);

char buffer[25];

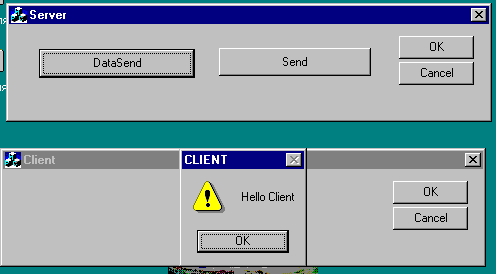
strcpy(buffer,(char \*)pCD->lpData);

AfxMessageBox(buffer);

return CDialog::OnCopyData(pWnd, pCD);

}

То же просто. Копируем строку из буфера и выводим на экран в виде сообщения. Ниже приведена картина неизвесного автора - "Программа в разгар тестирования":

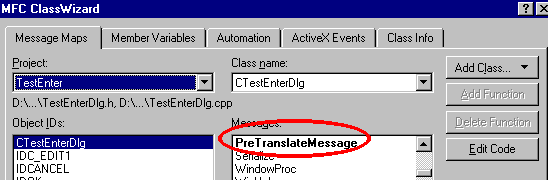


[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/117.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?116) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?118) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc3.html)  
Автор **Каев Артем**.

## Шаг 118 - Как вести лог файл сообщений

Данная задача возникает часто при отладке. Хочется видеть список сообщений, которые поступают скажем в диалоговое окно. При том посмотреть их в виде лог файла. Или еще чего. Есть ряд средств, например, **SPY**, которые позволяют это сделать. Только зачем, если у нас есть исходный код.

Создавайте проект **MFC AppWizard** на основе диалогового окна по максимуму отключив все кнопки. Теперь давайте создадим функцию **PreTranslateMessage**, на ней просто надо щелкнуть два раза и она появится в исходном коде.



Мы хотим вести лог файл только при отладке. Поэтому все, что относится к логу мы заключим в следующие операторы

#ifdef \_DEBUG

// это включится только в режиме отладки.

#endif

Включаем библиотеку файловых потоков

#include "TestEnterDlg.h"

**#ifdef \_DEBUG**

**#include "fstream.h"**

**#endif**

В конструкторе создаем поток.

CTestEnterDlg::CTestEnterDlg(CWnd\* pParent /\*=NULL\*/)

: CDialog(CTestEnterDlg::IDD, pParent)

{

//{{AFX\_DATA\_INIT(CTestEnterDlg)

// NOTE: the ClassWizard will add member initialization here

//}}AFX\_DATA\_INIT

// Note that LoadIcon does not require a subsequent DestroyIcon in Win32

**#ifdef \_DEBUG**

**ofs.open("log.txt");**

**#endif**

m\_hIcon = AfxGetApp()->LoadIcon(IDR\_MAINFRAME);

}

Во время закрытия окна закрываем. Для этого необходимо создать функцию **DestroyWindow**. Её тоже можно создать в **ClassWizard**.

BOOL CTestEnterDlg::DestroyWindow()

{

// TODO: Add your specialized code here and/or call the base class

**#ifdef \_DEBUG**

**ofs.close();**

**#endif**

return CDialog::DestroyWindow();

}

Ну и теперь то, что нас интересует.

BOOL CTestEnterDlg::PreTranslateMessage(MSG\* pMsg)

{

// TODO: Add your specialized code here and/or call the base class

**#ifdef \_DEBUG**

**ofs << pMsg->message << endl;**

**#endif**

return CDialog::PreTranslateMessage(pMsg);

}

Теперь если запустить именно отладочную версию **Debug**, а не реализ, то при работе программы будет создан лог файл. Что в нем ???? Вот что

....

160

160

512

512

512

512

512

...

Что это за сообщения ? Это сообщения в десятичном формате. Их надо перевести в шестнадцатеричный. Для этого можно воспользоваться обычным калькулятором. Поставьте научный вид. Переключитесь на **DEC**, введите десятичное число, а потом на **Hex** и увидите число в шестнадцатеричном виде.

Для интерпретации осталось немного. Первое, открыть файл **WinUser.h**. Там есть сообщения, они начинаются в **WM\_**. Так давайте посмотрим, что это за **512**.

512->200

#define WM\_MOUSEFIRST 0x0200

#define WM\_MOUSEMOVE 0x0200

Это двигалась мышка !!!

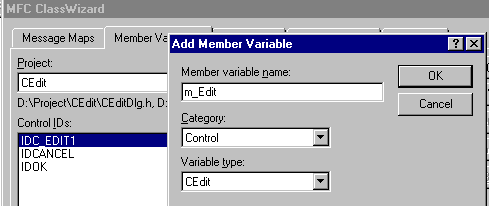
[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/118.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?117) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?119) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc3.html)  
Автор **Каев Артем**.

## Шаг 119 - Изучаем CEdit

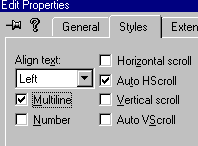
С 23 февраля всех ......

Мол кто не служил тому повезло, кто служил уже вернулся.

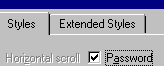
Давайте создадим приложение **MFC AppWizard** как диалоговое окно и имя ему дадим **CEdit**. Поместим на него элемент **Edit Box** и создадим для него класс.



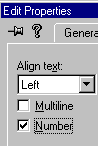
Наша задача в этом шаге поиздеваться над этим классом и посмотреть чего он может. Если Вы сечас запустите приложение, то сможете ввести только одну строку. Для ввода нескольких строк в свойствах элемента управления **CEdit** необходимо установить **MultiLine**.



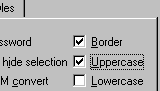
Только здесь есть маленькая засада. Создать новую строку можно только используя **CTRL- ENTER**. Просто **Enter** закроет диалоговое окно. Установка флага **Password** позволяет быстро создать окно ввода пароля.



Свойство **Number** разрешает вводить в окно только числа.



Установка свойства **UPPCASE** изменит окно в том направлении, что все вводимые симолы будут только большие, а **LowerCase** все вводимые символы будут маленькие.



[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?118) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?120) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc3.html)  
Автор **Каев Артем**.

## Шаг 120 - Дальше о CEdit

Давайте к созданному проекту в предыдущем шаге постепенно приделывать кнопочки, которые будут вызывать методы класса **CEdit**. Итак, создаем первую кнопку и с названием **LineLenght**. И код к ней.

void CCEditDlg::OnButton1()

{

char buffer[10];

itoa(m\_Edit.LineLength(),buffer,10);

AfxMessageBox(buffer);

}

Функция **LineLength()** возвращает количество символов в строке. Сроку можно не указывать, если не стоит **MultiLine**. Создаем еще одну кнопку с именем **LimitText** и код к ней.

void CCEditDlg::OnButton2()

{

m\_Edit.LimitText(5);

}

Функция **LimitText** позволяет ограничить количесто символов на ввод. Делаем кнопку **SetReadOnly** и код.

void CCEditDlg::OnButton3()

{

m\_Edit.SetReadOnly();

}

Эта функция запретит редактирование элемента. Создадим рядом кнопку для разрешения редактирования с именем **Edit** и код к ней.

void CCEditDlg::OnButton4()

{

m\_Edit.SetReadOnly(FALSE);

}

Код приведенный выше разрешит редактирование элемента. Давайте создадим новую кнопку с именем **Undo** и код к нему.

void CCEditDlg::OnButton5()

{

m\_Edit.Undo();

}

Создайте текст, а потом удалите последний символ. После этого нажмите нашу кнопку. Создаем кнопку с именем **Cut** и код к ней.

void CCEditDlg::OnButton6()

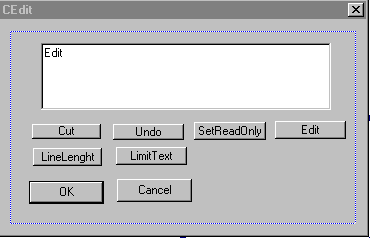
{

m\_Edit.Cut();

}

В ходе работы выделите текст и нажмите эту кнопку, текст будет вырезан и помещен в буфер.

На данный момент наш проект вот какой !!!



[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/120.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?119) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?121) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc3.html)  
Автор **Каев Артем**.

## Шаг 128 - CWinThread

**CWinApp** порожден от **CWinThread**:

....

class CWinApp : public CWinThread

{

.....

Класс **CWinThread** - это потоки выполняемые в приложении. Главный поток - это обычно **CWinApp**. Вообще-то потоки делятся на два типа:

* *рабочие* - нет цикла обработки сообщений
* *интерфейсные* - есть цикл обработки сообщений

Как раз **CWinApp** порожден с помощью интерфейсного потока. Создается поток функцией **AfxBeginThread**, которая имеет две версии в соотвествии с типами потоков.

CWinThread\* AfxBeginThread( AFX\_THREADPROC pfnThreadProc,

LPVOID pParam, int nPriority = THREAD\_PRIORITY\_NORMAL,

UINT nStackSize = 0, DWORD dwCreateFlags = 0,

LPSECURITY\_ATTRIBUTES lpSecurityAttrs = NULL );

CWinThread\* AfxBeginThread( CRuntimeClass\* pThreadClass,

int nPriority = THREAD\_PRIORITY\_NORMAL, UINT nStackSize = 0,

DWORD dwCreateFlags = 0,

LPSECURITY\_ATTRIBUTES lpSecurityAttrs = NULL );

Видите разницу ??? В первом случае передается адрес процедуры, а во втором адрес класса. Поток может быть создан и из конструктора класса, от которого он унаследован только в таком случае необходимо вызвать **CreateThread**. Вообще любое приложение **Win32** состоит из одного или нескольких процессов. Процесс в простом виде это выполняемое приложение. Операционная система распределяет время в соотвествии со статусами процесса. Именно процесс обеспечивает программу ресурсами, как ,например, адресное пространство, указатели, переменные операционной среды и так далее. Каждый процесс может породить дополнительные процессы.

Давайте создадим свой процесс на нашем диалоговом окне.

UINT MyFunction( LPVOID pParam )

{

for (int x=1;x<10;x++)

{

Sleep(1000);

MessageBeep(-1);

}

return 0;

}

void CTestAppDlg::OnButton6()

{

CWinThread\* myWinThread = AfxBeginThread(\*MyFunction, NULL);

}

Итак Вы нажимаете на кнопку и процес запускается. Вы можете дальше продолжать свою работу в то время как производится выполнение процесса и компьютер бибикает. Как же работает **AfxBeginThread** ??? А вот пример.

CWinThread\* pThread = DEBUG\_NEW CWinThread(pfnThreadProc, pParam);

ASSERT\_VALID(pThread);

if (!pThread->CreateThread(dwCreateFlags|CREATE\_SUSPENDED, nStackSize,

lpSecurityAttrs))

Как видите создается новый объект **CWinThread** и вызывается его функция **CreateThread**. Соотвественно можно делать и так. Например, поменять код в функции:

void CTestAppDlg::OnButton6()

{

//CWinThread\* myWinThread = AfxBeginThread(\*MyFunction, NULL);

CWinThread\* myTh;

myTh=new CWinThread(\*MyFunction,NULL);

myTh->CreateThread();

}

Сам же **MFC** использует функции **\_beginthread** из **Win32 API** для создания потоков.

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?127) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?129) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc3.html)  
Автор **Каев Артем**.

## Шаг 131 - Знакомимся CMemFile

Класс **CMemFile** хорош тем, что позволяет Вам создавать файл точную копию дискового файла в смысле работы и функционирования в оперативной памяти, что позволяет значительно увеличить скорость работы. Давайте попробуем. Создадим приложение **MFC AppWizard** на основе диалогового окна и одной кнопки. По нажатию на неё будет создаваться файл, потом в него записываться и считываться информация. Этот класс унаследован от **CFile**. Вы вдумайтесь в смысл. Это очень хороший способ использования оперативной памяти. Никаких проблемм !!! типа **Read** или **Write** или **Seek** без размера. Круто !!!

void CTestMemDlg::OnButton1()

{

CMemFile cm(1024);

char buffer[100];

strcpy(buffer,"Hello MemFile");

cm.Write(buffer,sizeof(buffer));

cm.Read(buffer,sizeof(buffer));

AfxMessageBox(buffer);

cm.Close();

}

Все начинается с конструктора. Их у него несколько.

CMemFile( UINT nGrowBytes = 1024 ); // начальный размер

CMemFile( BYTE\* lpBuffer, UINT nBufferSize, UINT nGrowBytes = 0 ); // на основе буфера

Файл автоматически открывается и поэтому использовать **Open** не нужно. Что будет происходить при записи в файл ?? Выделиться память стандартными средсвами, если памяти не хватит она увеличится на тот размер, который Вы установили. Выделенная память автоматически вернется системе после удаления объекта **CMemFile**. Второй вид конструктора немного сложнее. Вы сами выделяете память и передаете её для использования по типу файловых операций. Это тоже может быть удобно, только вся ответственость за память ляжет на Вас. Есть функции

Attach

Detach

Они позволяют динамично разрешать и запрешать доступ к данному файлу. Вот еще пример с использованием **Seek**.

void CTestMemDlg::OnButton1()

{

CMemFile cm(1024);

char buffer[100];

char buffer2[20];

strcpy(buffer,"Hello MemFile");

strcpy(buffer2,"Hello Buffer 2");

cm.Write(buffer,sizeof(buffer));

cm.Write(buffer2,sizeof(buffer2));

cm.Read(buffer,sizeof(buffer));

AfxMessageBox(buffer);

cm.Read(buffer2,sizeof(buffer2));

AfxMessageBox(buffer2);

cm.SeekToBegin();

cm.Read(buffer2,sizeof(buffer2));

AfxMessageBox(buffer2);

cm.Close();

}

Смотрите **SeekToBegin** устанавливает указатель в начало и потом зачитывает данные уже в **buffer2** работает как с файлом.

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/131.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?130) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?132) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc3.html)  
Автор **Каев Артем**.

## Шаг 132 - Знакомимся CStdioFile

Еще один класс порожденный от **CFile**. Этот класс позволяет пользоваться стандартными стредствами **C**,**C++** для ввода/вывода. При использовании этого класса важно понятие открытия и закрытия файла, как и при обычном открытии файла в **C** в отличии от расмотренного ранее **CMemFile**.

Давайте создадим приложение **AppWizard** на основе диалогового окна и поставим кнопку с кодом. Как делали в прошлые разы.

void CTestStdioDlg::OnButton1()

{

CStdioFile File("c:\\windows\\Wordpad.ini",CFile::modeRead);

CString str;

File.ReadString(str);

AfxMessageBox(str);

File.Close();

}

Любой класс начинается с конструктора. Вот какие конструкторы у этого класса.

CStdioFile( );

CStdioFile( FILE\* pOpenStream );

CStdioFile( LPCTSTR lpszFileName, UINT nOpenFlags );

throw( CFileException );

При укзании имени файла производится его открытие. Открытие может быть в текстовом и бинарном формате. Текстовый режим реализует стандартную обработку возврата коретки при чтении данных. Конец строки работает интересно. Вот так.

Посылаем конец стоки 0хА в файл пишется 0хА 0хD

Читаем из файла конец строки 0хА 0хD, читается из файла 0хA

Как видно из конструктора при невозможности операции с файлом появится исключение. Сделайте вот так.

void CTestStdioDlg::OnButton1()

{

CStdioFile File("c:\\windows\\NotFile.not",CFile::modeRead);

CString str;

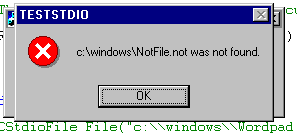
File.ReadString(str);

AfxMessageBox(str);

File.Close();

}

И вы получите исключение такого вида.



красивое сообщение, но его неполохо бы обработать хоть просто, вот например так хотя и не совсем корректно, потому что будут отлавливаться все типы исключений.

void CTestStdioDlg::OnButton1()

{

try

{

CStdioFile File("c:\\windows\\NotFile.not",CFile::modeRead);

CString str;

File.ReadString(str);

AfxMessageBox(str);

File.Close();

}

catch(...)

{

AfxMessageBox("Error Open");

}

}

К классу **CFile** есть ряд дополнительных функций:

ReadString читает строку текста

WriteString записывает строку текста

И указатель на открытый файл:

m\_pStream

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/132.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?131) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?133) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc3.html)  
Автор **Каев Артем**.

## Шаг 133 - Рисуем по диалоговому окну

Ваш проект VC++ first steps просто восхетительный!

Но учитывая что я полный чайник в VC++ ,без вопросов необошлось.

Вчем у меня дело ,нерисует ,хотя ошибок ненаходит?

Мне нужно ,шоб он рисовал при нажатии на Button1

void CMytestprogDlg::OnButton1()

{

CPaintDC dc(this);

dc.TextOut( 20, 20,"TextOut Samples");

}

Help me please ...

ostrongi@takas.lt

Значит так, я предложу свой вариант, как это сделать и если у Вас будут предложения или мысли как сделать по другому пишите. Может быть мой вариант совершенно не правильный.

Итак, создаем приложение на базе **AppWizard** как диалоговое окно. В окно вставляем кнопку. Кнопке нужен код. Вот он.

void CTestPaintDlg::OnButton1()

{

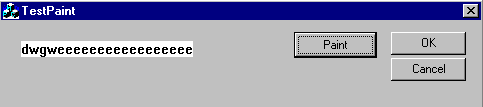
CDC\* dc;

dc=GetDC();

dc->TextOut(20,20,"dwgweeeeeeeeeeeeeeeee");

}

А вот результат работы.



У этого метода куча недостатков. Например, при передвижении окна или перерисовке эта надпись будет пропадать. Ну и ладно. Если этот код поместить в **OnPaint**, то ничего пропадать не будет.

void CTestPaintDlg::OnPaint()

{

...........

}

else

{

CDialog::OnPaint();

}

CDC\* dc;

dc=GetDC();

dc->TextOut(20,20,"dwgweeeeeeeeeeeeeeeee");

}

Основа этих действий функция **GetDC()**:

CDC\* GetDC( );

Эта функция получает контекст окна и находится она в **CWnd**, но этот класс есть в предках диалогового окна и поэтому мы им можем пользоваться. Получаем контекст устройства, который мы можем использовать для функций **GDI**. Эту функцию можно использовать если определены в структуре класса окна следующие стили.

CS\_CLASSDC

CS\_OWNDC

CS\_PARENTDC

Есть и похожие фунции, например, **GetWindowDC**

CDC\* dc;

dc=GetWindowDC();

dc->TextOut(20,20,"dwgweeeeeeeeeeeeeeeee");

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/133.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?132) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?134) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc3.html)  
Автор **Каев Артем**.

## Шаг 134 - Смена курсора

----Исходное сообщение-----

От: Yegor A. Blackheel

Кому: kaev@yandex.ru

Дата: 7 марта 2000 г. 13:31

Тема: MFCByStep

Привет, Артем! Классный сайт!

Альтернативный способ смены курсора вида:

1. В любом нужном месте вида вставляем строки

/\*наши курсоры\*/

HCURSOR newcursor, oldcursor;

.................................

/\* \*/

newcursor = AfxGetApp()->LoadCursor(IDC\_ВАШКУРСОР);

/\* если надо - сохраняем старый \*/

oldcursor = ::GetCursor();

/\*функция SetClassLong - принадледит глобальному

пространству имен - используем :: \*/

/\*GCL\_HCURSOR означает, что мы меняем именно

курсор, принадлежащий окну вида \*/

::SetClassLong(this->m\_hWnd,GCL\_HCURSOR,(long)newcursor);

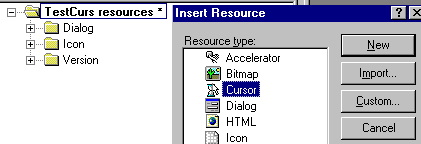
2. Все!

Конечно, чем именно пользоваться, дело вкуса :) Но мне кажется,

что использование SetClassLong достаточно интересная тема для нового ШАГА. :)

С уважением, Yegor A. Blackheel

Ну что, давайте попробуем и этот вариант. Делаем приложение на базе диалогового окна. На это окно помещаем кнопку и код к ней. В ресурсы добавляем наш курсор.



Нарисуйте что-нибудь на курсоре, например, фигу :-))) Давайте напишем код и посмотрим результат работы.

void CTestCursDlg::OnButton1()

{

HCURSOR cr;

HCURSOR old;

cr= AfxGetApp()->LoadCursor(IDC\_CURSOR1);

old = ::SetCursor(cr);

::SetClassLong(this->m\_hWnd,GCL\_HCURSOR,(long)cr);

}

Итак, создаем указатели на курсоры. Один на новый, второй на старый. Из ресурсов зачитываем курсор (**LoadCursor**). Дальше получаем указатель на старый курсор и устанавливаем новый (**SetCursor**). А вот теперь эта функция **SetClassLong**, которая помогает нам установить курсор. Результаты тестирования меня немного удивили :-))). Курсор устанавливается при перемещении над кнопкой в обычный вид, уберешь с кнопки - нормальный, который установили.

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/134.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?133) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?135) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc3.html)  
Автор **Каев Артем**.

## Шаг 136 - Новое в отладке VC 6.0

Появление **VC++ 6.0** принесло много нового в процесс создания приложений с использованием **C++**. Как известно, самой большой проблемой при отладке у компиляторов была необходимость перекомпилировать весь проект, даже при незначительных исправлениях. То есть процесс отладки выглядел так.

отладка <--------------

нахождение ошибки |

правка и --------------

Каждый раз приходилось прерывать выполнение программы, после чего производить изменения и опять искать то место. В **VC 6.0** появилась новая возможность, которая позволяет править и компилировать текст не прерывая программы. Давайте создадим простую программу, как **Win 32 Console** со следующим кодом.

// testLoad.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

#include "windows.h"

#include "iostream.h"

void Step1();

void main()

{

cout << " Module 1" << endl;

Step1();

}

void Step1()

{

cout << "Step1" << endl;

}

Скомпилируем и начнем отладку. Делайте пожалуйста за мной. Нажимаем **F10** пока курсор не установится на строку.

void main()

{

cout << " Module 1" << endl;

-> Step1();

}

Теперь, чтобы зайти внутрь процедуры нужно нажать **F11**. Давайте теперь изменим код.

void Step1()

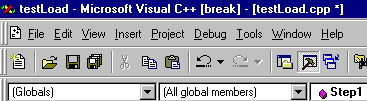
{

cout << "Step1" << endl;

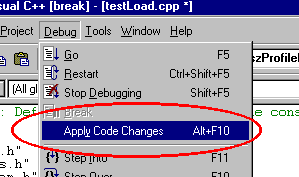
cout << "News " << endl;

}

Заголовок с именем файла должен измениться, около имени файла должна появиться точка.



По идее, в предыдущих версиях необходимо было начать программу заново и идти к этому же месту для продолжения отладки. Но мы работаем на **VC 6.0**. И у него есть пункт меню **Apply Code Change**.



Давайте применим этот пункт меню и посмотрим, что произойдет. А произойдет перекомпиляция программы, но само выполнение программы не прервется. Вы должны будете в окне отладки **Build** увидеть примерно следующее.

Compiling...

testLoad.cpp

Applying Code Changes...

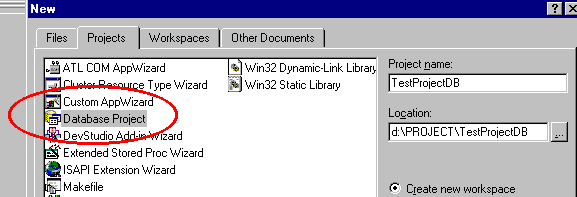
Edit and Continue - 0 error(s), 0 warning(s)

Ну как удобно ??? Мне кажется для мелких исправлений это классно.

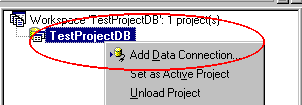
[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?135) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?137) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc3.html)  
Автор **Каев Артем**.

## Шаг 137 - Как среда VC работает с файлами БД

Запускайте **VC**. Выбирайте меню **File -> New**, выбирайте тип проекта как **Database Project**. Дайте имя проекту **TestProjectDB**.



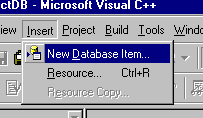
Тут же появится окно, которое предложит Вам выбрать драйвер **ODBC** для работы с базами нажмите пока на отмену. Для того, чтобы добавить соединение с данными нужно щелкнуть правой кнопкой мыши на имени проекта в правом окне (вкладка **File**) и выбрать пункт меню **Add Data Cоnnection**.



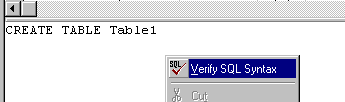
В ответ на этот пункт меню появится окошко с целью выбора драйвера **ODBC** для работы. Выберите **DBASE Files** и нажмите кнопочку **OK**. Дальше Вам необходимо выбрать каталог где будут находиться файлы баз данных. Установите Ваш рабочий каталог. В результате у Вас появится вкладка **DataView** с именем подключенного источника данных.



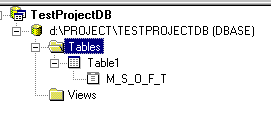
Давайте создадим пустую таблицу. Итак, нам нужен пункт меню **Insert -> New Database item**.



После вопроса ответьте, что у Вас будет локальный запрос. У Вас изменится левая часть и появится поле для ввода **SQL** оператора. Давайте введем оператор создания таблицы.



Теперь этот оператор нужно проверить на соответствие синтаксису языка. Нажмите правую кнопку и выберите **Verify SQL**. У вас должна появиться первая панель, которая должна сообщить, что оператор правильный и второе диалоговое окно, что якобы что-то в драйвере **ODBC** не так. По крайней мере у меня появляется, ну и черт с ним. Ответьте **Yes**. Теперь можно пойти в меню **Tools** и выбрать **RUN**. По выбору этого пункта меню оператор выполнится, о чем Вас известят. Два раза щелните на **Tables**, вы должны увидеть, что таблица появилась.



[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/137.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?136) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?138) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc3.html)  
Автор **Каев Артем**.

## Шаг 138 - Галочки рядом с меню

-----Исходное сообщение-----

От: Radovets Konstantin

Кому: kaev@yandex.ru

Дата: 9 марта 2000 г. 10:14

Тема: mfcbystep

Привет Артем!!!

Такого клевого сайта я еще нигде не видел.

У меня к тебе есть небольшой вопрос:

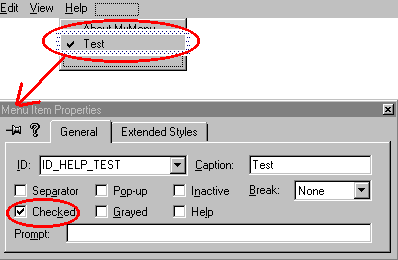
Ты не знаешь как можно добавлять галочки или ярлычки в меню.

Например, в меню "Вид" есть подменю "Cтрока состояния"

Искренне благодарен за ответ, Константин.

Итак, галочки так галочки. Нет бы кто написал, мол отдохни немного мы поможем, а тут галочки :-))) И прямо после 8 марта на больную голову :-)))

Делаем проект как **MFC AppWizard** лучше как **SDI** с именем **MyMenu**, остальное не важно. У этого проекта будет меню честное слово. Так вот давайте переключимся на вкладку **Resource**. Зайдем в меню и добавим пункт **test** в меню **Help**. Для того, чтобы вообще-то галочка была всего-то нужно установить флаг **checked**.



Теперь давайте привяжем к нему код. Лучше через **Class Wizard**. Если вы сделаете так же, как я то процедура обработки пункта меню должна быть такая, а код естественно сами допишите или возьмете проект.

void CMainFrame::OnHelpTest()

{

CMenu\* cm=GetMenu();

if (cm->GetMenuState(ID\_HELP\_TEST,MF\_BYCOMMAND) & MF\_CHECKED)

cm->CheckMenuItem(ID\_HELP\_TEST,MF\_UNCHECKED);

else

cm->CheckMenuItem(ID\_HELP\_TEST,MF\_CHECKED);

}

Итак, есть функция у класса **CWnd**, который является дальним, но прямым родственником нашего класса, и который наложил существенный отпечаток на наш класс. Эта функция **GetMenu**

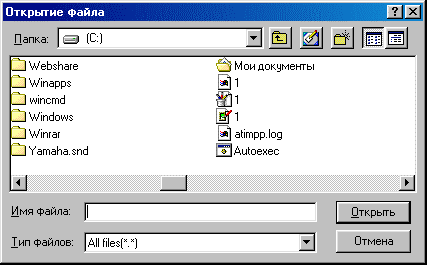
CMenu\* GetMenu( ) const;

Она возвращает указатель на меню приложения. Вообще-то она может быть и **NULL**, но я то знаю что меню у меня есть вроде, поэтому проверять не стал, а наверно стоит :-). А дальше я пользуюсь функцией **GetMenuState** уже класса **CMenu** дабы убедиться, что меню на основе идентификатора **ID\_HELP\_TEST** с галочкой **MF\_CHECKED** и меняю ситуацию на обратную используя функцию, которая умеет это делать **CheckMenuItem**. Во как.

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/138.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?137) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?139) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc3.html)  
Автор **Каев Артем**.

## Шаг 139 - Стандартные диалоги Открытия и Сохранения файлов

Стандартный диалог открытия(сохранения) файла выглядит следующим образом:



Не будет преувеличением сказать, что мы его используем несколько десятков раз на дню. Давайте создадим обычное приложение на базе диалогового окна. Добавим в него кнопку и элемент **ListBox**. Для элемента **ListBox** еще опишем переменную типа **Control** с именем **m\_List** (**MFC Class Wizard -> Member Variables -> Add Variable**). Я задался целью помучить диалог открытия файла, который ничем принципиально не отличается от диалога сохранения файла, и даже класс для реализации этих диалогов существует один на двоих - **CFileDialog**.

Итак, самый простой и стандартный вызов диалога открытия файла при нажатии на нашу кнопку:

void CTestDlg::OnButton1()

{

CFileDialog fd(true);

if (fd.DoModal()==IDOK)

{

m\_List.ResetContent();

m\_List.AddString(fd.m\_ofn.lpstrFile);

}

}

Конструктор класса перегружает все параметры, кроме первого, по умолчанию и поэтому, в принципе, их можно не указывать. Вызовы конструктора в строке

CFileDialog fd(true);

идентичен по смыслу вызову:

CFileDialog fd(true,NULL,NULL, OFN\_HIDEREADONLY+

OFN\_OVERWRITEPROMPT,NULL,NULL);

Итак, конструктор класса:

CMyFileDialog(

BOOL bOpenFileDialog, // TRUE - диалог "Открыть", FALSE - диалог "Сохранить как"

LPCTSTR lpszDefExt = NULL, // расширение (по умолчанию)

LPCTSTR lpszFileName = NULL, // имя файла (по умолчанию)

DWORD dwFlags = OFN\_HIDEREADONLY | OFN\_OVERWRITEPROMPT, //возможные флаги диалога

LPCTSTR lpszFilter = NULL, // фильтр(ы) для вывода определенной группы файлов

CWnd\* pParentWnd = NULL // указатель на родителя, вызвавшего диалог

);

После объявления класса запускаем его на выполнение с помощью **DoModal**. Если была нажата кнопка **OK**, то элемент типа **ListBox** сначала очищается (на всякий случай) с помощью **ResetContent**, а потом в него добавляем выбранный файл вместе с путем с помощью **AddString**. Вполне естественно возникает вопрос насчет аргумента **AddString - fd.m\_ofn.lpstrFile**. Что такое **fd** я думаю понятно, а **m\_ofn** является встроенной переменной класса **CFileDialog** и имеет тип **OPENFILENAME** (структура из **WinAPI**). Член структуры (не подумайте чего плохого!) **lpstrFile** содержит имя выбранного файла с путем. Структура **OPENFILENAME** содержит в себе около 20 членов. Все параметры конструктора (кроме первого), которые мы задаем при инициализации **fd**, можно легко изменить с помощью этой встроенной переменной **m\_ofn**OPENFILENAME можно найти в разделе **MFC** (WinAPI).

На следующем шаге я рассмотрю более продвинутый вариант работы с диалогом Открытия файлов.

### Открываем несколько файлов

Давайте немного усложним задачу и рассмотрим случай, когда можно выбрать как один файл, так и группу файлов, используя **Ctrl** или **Shift**. Берем проект из предыдущего шага и усложняем процедуру реакции на нажатие на кнопку.

Вот новый код кнопки:

void CTestDlg::OnButton1()

{

CWnd\* TheWindow=GetActiveWindow();

CFileDialog fd(true,NULL,NULL,OFN\_ALLOWMULTISELECT+

OFN\_HIDEREADONLY,NULL,TheWindow);

fd.m\_ofn.lpstrTitle="My own OpenFile Dialog";

fd.m\_ofn.lpstrFilter="Bitmap files(\*.bmp)\0\*.bmp\0All files(\*.\*)\0\*.\*\0\0";

fd.m\_ofn.nFilterIndex=2;

fd.m\_ofn.lpstrInitialDir="c:";

if (fd.DoModal()==IDOK)

{

m\_List.ResetContent();

m\_List.AddString(fd.m\_ofn.lpstrFile);

int i,st;

CString file,qq;

st=fd.m\_ofn.nFileOffset;

qq=fd.m\_ofn.lpstrFile[st-1];

//Checking if one file or more...

if (qq=="")

{

file="";

for (i=st;true;i++)

{

qq=fd.m\_ofn.lpstrFile[i];

if (qq=="")

{

m\_List.AddString(file);

file="";

qq=fd.m\_ofn.lpstrFile[i+1];

if (qq=="") break;

}

else

file=file+qq;

}// for loop

}//if more than 1 file

} // if (fd.DoModal==IDOK)

}

А теперь пояснения к приведенному коду.

Получаем указатель на текущее окно с помощью функции **GetActiveWindow** и помещаем его в переменную **TheWindow**. Далее инициализируем переменную **fd**. Я не стал пока особо извращаться и установил второй, третий и пятый параметр как **NULL**. Первый параметр **true** говорит о том, что используем диалог открытия файла. Флаги устанавливаем по минимуму. Они указывают, что можно выбирать несколько файлов сразу и элемент "Только для чтения" диалога не будет показан. В последнем параметре вызова конструктора определил указатель на текущее активное окно. Даже если бы мы проставили в последнем параметре **NULL**, то все равно хозяином диалога стало бы текущее окно приложения. Но так работает **MFC**. Я же немного побаловался с соответствующими функциями **WinAPI**. Если не указывал этот параметр, то диалог становился бесхозным.

Далее демонстрирую то, что с помощью встроенной переменной **m\_ofn** (тип **OPENFILENAME**) можно изменить многие параметры стандартного диалога. Изменяем заголовок диалога (параметр **lpstrTitle**); описываем 2 вида фильтров (параметр **lpstrFilter**); указываем, что при создании диалога вызывается второй фильтр(параметр **nFilterIndex**); указываем начальный путь диалога (параметр **lpstrInitialDir**). Полное описание структуры **OPENFILENAME** можно найти в разделе **MFC(WinAPI)**. Далее по тексту еще не раз встретятся параметры из этой структуры (т.е. **fd.m\_ofn.<параметр>**). За разъяснениями обращайтесь в указанный раздел.

Вызываем диалог с помощью **DoModal** и если нажата кнопка **OK**, то анализируем полученные данные (путь и имена файлов). Если выбираем один файл, то в элемент **ListBox** добавляется имя файла с путем к нему. Если выбрано несколько файлов, то в первой строке элемента указывается путь к файлам, а в последующих перечисляются имена файлов. В параметре **lpstrFile** хранится путь и имя(имена) выбранных файлов.

Смотрим вариант, когда выбрано несколько файлов.

Проблема заключается в том, что путь и имена файлов разделены нулевым ("\0" или "") символом. Это сделано для того, чтобы корректно воспринимать пробелы в именах файлов. Таким образом, напрямую из параметра **lpstrFile** можно получить только путь к файлам. После пути к файлам стоит нулевой символ, что обычно воспринимается как конец строки. Но мы то знаем, что дальше должны быть перечислены файлы! После последнего имени файла стоит двойной нулевой символ.. Параметр **nFileOffset** показывает сдвиг имени файла от начала строки. Если предыдущий символ **fd.m\_ofn.lpstrFile[st-1]** имеет нулевое значение, значит последует перечисление нескольких файлов, если же любой другой символ - выбран всего один файл и последующий код не нужен.

Начинаем по порядку просматривать все символы параметра **lpstrFile** начиная с первого символа перечисления имен файлов (параметр **nFileOffset**) и до тех пор, пока не найдем двойной нулевой символ. В переменной **file** сохраняем имя текущего файла. Как только находим нулевой символ, добавляем строку из переменной **file** в элемент **ListBox** и обнуляем переменную **file**. При этом проверяем, если следующий символ имеет значение **NULL**(нулевой), то прекращаем поиск и выходим из цикла.

Прислал [Valeri Khromov](mailto:lerick@mail.ru).

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?138) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?140) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc3.html)  
Автор **Каев Артем**.

## Шаг 140 - Cтруктура OPENFILENAME

Для вызова стандартных диалогов открытия и сохранения файла в **MFC** есть класс **CFileDialog**. В **WinAPI** существуют соответствующие функции вызова этих диалогов.

Вызов стандартного диалога для открытия файла осуществляется с помощью функции **GetOpenFileName**

BOOL GetOpenFileName(

LPOPENFILENAME lpofn

// адрес структуры с данными

//для инициализации диалога

);

Вызов стандартного диалога для сохранения файла осуществляется с помощью функции **GetSaveFileName**

BOOL GetSaveFileName(

LPOPENFILENAME lpofn

// адрес структуры с данными для инициализации диалога

);

где:

**lpofn** - это указатель на структуру **OPENFILENAME**, в которой содержится информация, необходимая для инициализации диалога. Когда функция выполнена, эта структура содержит информацию о том, какие файлы были выбраны пользователем. Эти две функции похожи как близнецы-братья.

Если пользователь указал какое-либо имя файла(несколько имен) и нажал кнопку **OK**, то возвращается ненулевое значение. При этом, выбранные имена, путь и т.д. и т.п. можно почерпнуть из все той же структуры типа **OPENFILENAME**.

Если же пользователь жмет на кнопку "Отмена", закрывает окно или происходит какая-то ошибка в ходе выполнения функции, то возвращаемое значение будет **0**. Для получения дополнительной информации об ошибках в процессе выполнения функции используется функция **CommDlgExtendedError**

А теперь, пожалуй, самое интересное и важное - сама структура **OPENFILENAME**:

typedef struct tagOFN { // ofn

DWORD lStructSize;

HWND hwndOwner;

HINSTANCE hInstance;

LPCTSTR lpstrFilter;

LPTSTR lpstrCustomFilter;

DWORD nMaxCustFilter;

DWORD nFilterIndex;

LPTSTR lpstrFile;

DWORD nMaxFile;

LPTSTR lpstrFileTitle;

DWORD nMaxFileTitle;

LPCTSTR lpstrInitialDir;

LPCTSTR lpstrTitle;

DWORD Flags;

WORD nFileOffset;

WORD nFileExtension;

LPCTSTR lpstrDefExt;

DWORD lCustData;

LPOFNHOOKPROC lpfnHook;

LPCTSTR lpTemplateName;

} OPENFILENAME;

Рассмотрим составляющие этой структуры по отдельности (приводится частичное описание параметров):

**lStructSize** - определяет длину структуры в байтах.

**hwndOwner** - идентифицирует окно, которое является "владельцем". Это может быть как указатель на окно, так и **NULL** значение (если нет владельца).

**hInstance** - принимается во внимание только если установлены флаги **OFN\_ENABLETEMPLATEHANDLE** или **OFN\_ENABLETEMPLATE** и идентифицирует модуль, в котором находится шаблон формы диалога. Имя шаблона указывается в параметре **lpTemplateName** структуры.

**lpstrFilter** - указатель на буфер, в котором находятся пары нуль-терминированных (null-terminated) строк для фильтра. Последняя строка в этом буфере должна заканчиваться двумя **NULL** символами. Первая строка пары содержит описание фильтра (например, "Text Files"), а вторая - сам шаблон фильтра (например, "\*.TXT"). Можно указать несколько шаблонов для одного фильтра. Для отделения одного шаблона от другого (для одного и того же фильтра) используется знак ";"(точка с запятой), например, "\*.TXT;\*.DOC;\*.BAK". Шаблон должен состоять из комбинации разрешенных символов и знака (\*) звездочки. В шаблоне не используются пробелы.

Операционная среда не меняет порядок фильтров, а показывает их в том порядке, в котором шаблоны были заданы в **lpstrFilter**. Если **lpstrFilter** имеет значение **NULL**, то диалог не показывает фильтров.

Пример строки для этого параметра из **MFC** (3 шаблона):

"Bitmap files(\*.bmp)\0\*.bmp\0JPEG files(\*.jpg)\0\*.jpg\0All files(\*.\*)\0\*.\*\0\0"

**lpstrCustomFilter** - указатель на статический буфер, в котором находится пара ноль-терминированных(null-terminated) строк для определения шаблона пользователя. Первая строка описывает шаблон, вторая - сам шаблон. Первый раз, когда Ваше приложение создает диалоговое окно, вы определяет первую строку(не должна быть пустой). Когда пользователь выбирает файл, диалог копирует текущий шаблон фильтра во вторую строку. Этот шаблон может быть как одним из шаблонов, определенных в буфере **lpstrFilter**, так и может быть введен пользователем. Система использует эти строки для определения шаблона, указанного пользователем, в следующий раз, когда диалог будет вызван. Если параметр **nFilterIndex** равен нулю, то диалог использует пользовательский шаблон. Если этот параметр имеет значение **NULL**, то диалог не поддерживает использование пользовательского шаблона. Если значение этого параметра не равно **NULL**, то параметр **nMaxCustFilter** должен указывать размер буфера **lpstrCustomFilter**: для **ANSI** в байтах, а для **Unicode** - в символах.

**nMaxCustFilter** - определяет размер, в байтах, буфера, описанного в **lpstrCustomFilter**. Этот буфер должен быть длиной как минимум в 40 символов. Этот параметр игнорируется, если **lpstrCustomFilter** имеет значение **NULL** или указывает на строку со значением **NULL**.

**nFilterIndex** - определяет выбранный в настоящий момент фильтр для поля "Тип файлов" диалога. Параметр указывает на пару строк, которые определены в параметре **lpstrFilter**. Первая пара строк параметра **lpstrFilter** имеет индекс 1, вторая - 2 и т.д. Индекс 0 указывает на то, что выбран фильтр, определенный в **lpstrCustomFilter**. Можно указать необходимый индекс фильтра при вызове диалога. Когда пользователь выбирает файл, **nFilterIndex** возвращает индекс текущего фильтра. Если **nFilterIndex** равен 0 и **lpstrCustomFilter** имеет значение **NULL**, то система использует первый фильтр из буфера **lpstrFilter**. Если все три параметра равны 0 или **NULL**, то система не использует никаких фильтров и не показывает никаких файлов в списке файлов диалога.

**lpstrFile** - указатель на буфер, в котором находится имя для инициализации поля "Имя файла" диалога. Если инициализация не нужна, то первый символ буфера должен быть **NULL**. Если функции GetOpenFileName или **GetSaveFileName** успешно выполнены, то буфер содержит полный путь, имя и разрешение выбранного файла(ов).

Если флаг **OFN\_ALLOWMULTISELECT** установлен и пользователь выбирает несколько файлов, то буфер содержит текущую папку и затем следуют имена выбранных файлов. Для "Explorer-style" диалога, папка и имена файлов разделены символом **NULL**. После последнего имени файла добавляется еще один символ **NULL**. В диалогах старого типа строки разделены пробелами и функции используют короткие имена для файлов с пробелами в имени. Функция **FindFirstFile** может быть использована для преобразования длинных имен в короткие. Если буфер слишком мал, то функция (**GetOpenFileName** или **GetSaveFileName**) вернет **FALSE** и функция **CommDlgExtendedError** вернет значение **FNERR\_BUFFERTOOSMALL**. В этом случае первые два байта параметра **lpstrFile** будут содержать требуемый размер (в байтах или символах).

**nMaxFile** - определяет размер буфера, на который указывает параметр **lpstrFile**, в байтах (**ANSI**) или в символах (**Unicode**). Функции **GetOpenFileName** и **GetSaveFileName** вернут значение **FALSE** если буфер будет слишком маленьким, чтобы вместить информацию о файлах. Буфер должен быть как минимум длиной в 256 символов.

**lpstrFileTitle** - указатель на буфер, в котором находятся имя и расширение(без пути) выбранного файла. Параметр может иметь значение **NULL**.

**nMaxFileTitle** - определяет размер буфера, на который указывает параметр **lpstrFileTitle**, в байтах (**ANSIUnicode**). Этот параметр игнорируется, если **lpstrFileTitle** имеет значение **NULL**.

**lpstrInitialDir** - указатель на строку, в которой определяется стартовая папка при открытии диалога. Если параметр имеет значение **NULL**, то система использует текущую папку как стартовую для этого диалога.

**lpstrTitle** - указатель на строку, которая будет помещена в заголовке окна диалога. Если этот параметр имеет значение **NULL**, то система использует стандартные заголовки (соответственно, "Сохранить как" или "Открыть")

### Flags (флаги)

Набор флагов используется при инициализации диалога. Они определяют то, как будет происходить и результат взаимодействия пользователя и системы. Этот параметр может быть представлен как комбинация флагов (например, **OFN\_ALLOWMULTISELECT + OFN\_EXPLORER**): Флаги и их расшифровки:

**OFN\_ALLOWMULTISELECT** - Разрешает выбирать несколько файлов сразу (с помощью клавиш **Shift** и **Ctrl**) в открывшемся диалоге. Если при этом указан флаг **OFN\_EXPLORER**, то диалог будет использовать Explorer-style интерфейс диалога; иначе будет использован старый стиль диалога. Если пользователь вибирает более, чем один файл, то параметр **lpstrFile** возвращает путь к текущему каталогу, а за ним следуют имена выбранных файлов. Параметр **nFileOffset** указывает сдвиг(смещение) первого имени файла, а параметр **nFileExtension** вообще не используется. В диалогах типа Explorer-style путь к файлам и имена файлов разделены нулевым (**NULL**) символом с дополнительным нулевым символом после последнего имени файла. Использование нулевых символов для разделение пути и имен файлов позволяют возвращать диалогу типа Explorer-style имена файлов с пробелами. В диалогах старого образца путь и имена файлов разделены пробелами и функции используют короткие (8.3) имена для файлов с пробелами в имени. Для преобразования длинных имен в короткие можно использовать функцию **FindFirstFile**.

**OFN\_CREATEPROMPT** - Если пользователь указал файл, которого не существует, то диалог запросит разрешение создать новый файл с указанным именем. Если пользователь выберет создание нового файл, то диалог закроется и вернет указанное имя, в противном случае диалог останется открытым.

**OFN\_ENABLEHOOK** - Разрешает применение перехватывающей процедуры, определенной в параметре **lpfnHook**.

**OFN\_ENABLETEMPLATE** - показывает, что параметр **lpTemplateName** указывает на ресурс шаблона диалога в модуле, который указан в параметре **hInstance**.

**OFN\_ENABLETEMPLATEHANDLE** - показывает, что параметр **hInstance** определяет блок данных, который является пред-загружаемым шаблоном.

**OFN\_EXTENSIONDIFFERENT** - показывает, что пользователь ввел расширение, которое отличается от расширения, используемого в параметре **lpstrDefExt**. Этот флаг не используется, если флаг **lpstrDefExt** имеет значение **NULL**.

**OFN\_FILEMUSTEXIST** - определяет, что пользователь может ввести только имена существующих файлов в поле ввода "Имя файла" диалога. Если указан этот флаг и пользователь ввел некорректное имя файла, то выдается предупреждающее сообщение. Если указан этот параметр, то также используется флаг **OFN\_PATHMUSTEXIST**.

**OFN\_HIDEREADONLY** - скрывает поле "Только для чтения" диалога.

**OFN\_LONGNAMES** - Для диалогов старого образца. Этот флаг определяет использование длинных имен файлов. Если этот флаг не установлен или **OFN\_ALLOWMULTISELECT** флаг также выбран, диалоговый окна старого образца используют формат коротких имен (8.3) для файлов с пробелами. Explorer-style диалоги игнорируют этот флаг и всегда показывают длинные имена.

**OFN\_NOCHANGEDIR** - устанавливает текущей папкой первоначальное значение в случае, если пользователь переходит из одной папки в другую при поиске фалов.

**OFN\_NODEREFERENCELINKS** - указывает диалогу возвращать путь и имя файла ярлыка (**.LNK**). Если этот флаг не установлен, то диалог возвращает путь и имя файла, на которые указывает ярлык.

**OFN\_NOLONGNAMES** - для диалогов старого типа. Этот флаг приводит к тому, что в диалоге используются короткие имена (формат 8.3). Explorer-style диалог игнорирует этот диалог и всегда показывает длинные имена.

**OFN\_NONETWORKBUTTON** - убирает и выключает кнопку "Сеть" диалога.

**OFN\_NOREADONLYRETURN** - определяет, что диалог не содержит выбранным поле "Только для чтения" и возвращаемый файл не находится защищенной от копирования папке.

**OFN\_NOTESTFILECREATE** - определяет, что файл не создается до тех пора, когда диалог не будет закрыт. Этот флаг должен быть определен, если приложение сохраняет файл при наличии прав "создать но не модифицировать" в сети (create-nonmodify network sharepoint). При использрвании этого флага не производится проверки на наличие защиты от записи, полный диск, не вставлен диск или защиты от сетевого доступа. При этом, приложение должно очень осторожно должно использовать эту операцию для открытия файла, т.к. файл не может быть открыт заново, если он был закрыт.

**OFN\_NOVALIDATE** - позволяет использовать некорректные символы при вводе имени файла. Обычно, перехватывающая(hook) процедура вызывающего приложения проверяет корректность имени файла используя вызов FILEOKSTRING. Если поле ввода имени файла диалог пусто или заполнено пробелами, то список файлов и папок обновляется. Если поле ввода содержит что-то другое, то значения параметров nFileOffset и nFileExtension устанавливаются путем грамматического разбора текста. При этом к тексту не добавляются ни расширение по умолчанию, ни текстовый буфер, определенный в параметре lpstrFileTitle. Если значение параметра nFileOffset меньше 0, то имя файла некорректно. В противном случае имя считается корректным, nFileExtension и nFileOffset используются как будто не был указан флаг OFN\_NOVALIDATE (т.е. как обычно).

**OFN\_OVERWRITEPROMPT** - определяет, что диалог "Сохранить как" выдаст предупреждающее сообщение, если файл уже существует. Пользователь должен подтвердить перезапись файла.

**OFN\_PATHMUSTEXIST** - определяет, что пользователь может указать только существующий путь и имена файлов. Если пользователь ввел неправильное имя файла или путь, то появится предупреждающее окно.

**OFN\_READONLY** - использование этого флага ведет к тому, что "Только для чтения" поле для выбора диалога будет выбрано при создании диалога. Когда диалог закрыт, этот флаг говорит о состоянии поля "Только для чтения" диалога.

**OFN\_SHAREAWARE** - определяет, что если вызов функции **OpenFile** потерпел неудачу из-за отсутствия сетевого доступа, то эта ошибка будет проигнорирована и диалог вернет имя выбранного файла. Если этот флаг не установлен, диалог сообщит перехватывающей(**hook**) процедуре, о произошедшей ошибке доступа к сетевым ресурсам(файлу).

**OFN\_SHOWHELP** - выводит в окне диалога кнопку "Справка". Параметр **hwndOwner** должен определять окно, которому получит сообщение **HELPMSGSTRING**, посылаемое диалогом, когда пользователь нажимает на кнопку "Справка".

**nFileOffset** - определяет сдвиг (основанный на нуле) от начала пути на имени файла в строке, на которую указывает параметр **lpstrFile**. К примеру, если **lpstrFile** указывает на следующую строку, "c:\dir1\dir2\file.ext", то этот параметр равен 13, что обозначает сдвиг строки "file.ext" от начала параметра **lpstrFile**.

**nFileExtension** - определяет сдвиг(основанный на нуле) начиная с первого символа пути до расширения файла, на который указывает параметр **lpstrFile**. Например, если **lpstrFile** указывает на строку "c:\dir1\dir2\file.ext", то этот параметр равен 18. Если пользователь не указал расширение и **lpstrDefExt** имеет значение **NULL**, то этот параметр указывает сдвиг до последнего(терминирующего) символа. Если в имени файла пользователь указал последним символом ".", то этот параметр равен 0.

**lpstrDefExt** - указатель на буфер, в котором находится расширение по умолчанию. Если пользователь не указал расширение файла, то это расширение будет добавлено к имени файла. Строка может быть любой длины, но присоединяются только 3 первых символа. Строка не должна содержать точки (.). Если этот параметр имеет значение **NULL**, то, при отсутствии расширения, ничего к имени файла не добавляется.

**lCustData** - В этом параметре хранятся данные, которые передаются перехватывающей процедуре, указанной в параметре **lpfnHook**.

**lpfnHook** - указатель на перехватыватывающую процедуру. Этот параметр игнорируется, если не установлен флаг **OFN\_ENABLEHOOK**.

**lpTemplateName** - указатель на нуль-терминированную строку, в которой описаны имена(указатели) ресурсов, определенные в параметре **hInstance**.

Вызов других стандартных диалогов (выбор цвета, поиск и т.д.) возможен как через соответствующие классы **MFC**, так и с помощью соответствующих функций **WinAPI**.

Прислал [Valeri Khromov](mailto:lerick@mail.ru).

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?139) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?141) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc3.html)  
Автор **Каев Артем**.

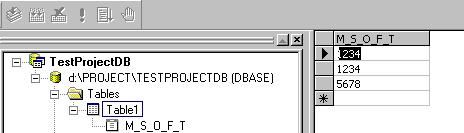
## Шаг 141 - Дальше о том как среда VC работает с файлами БД

Итак, в прошлый раз мы с Вами создавали таблицу. Теперь давайте к созданной таблице добавим записи. Открывайте наш проект. Выбирайте пукнт меню **Insert** и далее **New Database Item** для ввода **SQL** операторов. Итак. Вот сам оператор.

INSERT INTO `Table1`

VALUES (5678)

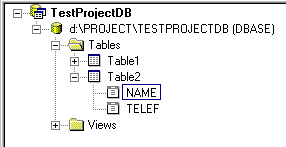
Он переводится примерно так - Вставить в таблицу строку с значением 5678. Попробуйте поэкспериментировать и получить такие результаты как на экране.



Для того, чтобы создать таблицу с заранее создаными полями нужно указать их спецификацию в **CREATE TABLE**

CREATE TABLE Table2(Name char(20), telef char(20))

Вот результат этой операции.



Как вы заметили все указанные мною операции являются отдельными командами языка управления базами данных **SQL**. Вы можете отдельно почитать о языке **SQL** и поэксперементировать не только по описанным возможностям, но и больше.

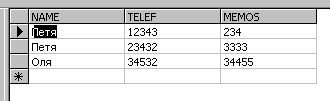
К созданой таблице можно добавить колонки, но при условии, что в ней нет данных, вот так например.

ALTER TABLE Table2 ADD Memos char(20)

Теперь давайте заполним эту таблицу значениями вот примерно так.

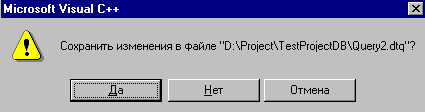
INSERT INTO `Table2`

VALUES ('Оля', '34532', '34455')

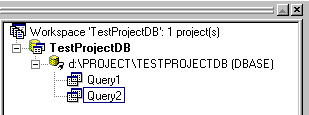


Следует отметить, что возможности самого языка зависят от типа драйвера **ODBC** и физической базы данных. Например, понятие вида (**View**) не существут для **DBF** и поэтому использовать допустим оператор **CREATE VIEW** не получится.

Так же интересен тот момент, что все созданные **SQL** операторы можно сохранять. При закрытии правого окна появится вопрос о сохранении запроса. Отвечайте да.



Вводите имя. Все запросы будут помещены на вкладку **FileView**. Выбрав запрос и щекнув на нем два раза Вы сможете быстро повторить операцию.



[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/141.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?140) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?142) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc3.html)  
Автор **Каев Артем**.

## Шаг 142 - Обрабатываем нажатия клавиш

Добрый день Артем!

Спасибо за ответ на предыдущий вопрос!

Напишите пожалуйста как пользоватся OnKeyDown (если ето та функция

что я имею ввиду), их там немало OnKeyUp , OnKeyDownEvent ,

OnKeyPress , ...

Например при нажатии на J ...

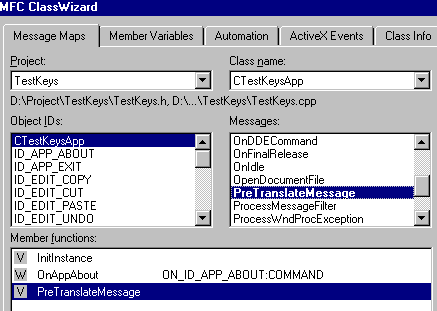
И при нажатии на такие кнопки как ВВЕРХ , ВНИЗ,TAB, F1 .......

Thanx !!!

ostrongi@takas.lt

Ну что, давайте поговорим о том, как вообще сообщения **Windows** в том числе и нажатие клавиш появляются, и как они обрабатываются **MFC**. Давайте создадим приложение **MFC AppWizard** как окно **MDI**, то есть с многими окнами и назовем его **TestKeys**.

Как известно, самый главный класс у нас **CTestKeysApp**, он порожден от **CWinApp** и поэтому сообщения от **Windows** поступают сначала туда. Давайте перегрузим функцию первоначальной обработки сообщения **PreTranslateMessage**



BOOL CTestKeysApp::PreTranslateMessage(MSG\* pMsg)

{

// TODO: Add your specialized code here and/or call the base class

MessageBeep(-1);

return CWinApp::PreTranslateMessage(pMsg);

}

Запустите это приложение и Вы увидите, что на любое Ваше действие комп будет пищать !!! В том числе и на нажатие клавиш !!!. То есть обработать при желании можно и здесь. Куда дальше будет передаваться сообщение ?? В **CMainFrame**. Здесь тоже сообщение можно отловить еще до обработки. Вот так.

BOOL CMainFrame::PreTranslateMessage(MSG\* pMsg)

{

// TODO: Add your specialized code here and/or call the base class

MessageBeep(-1);

return CMDIFrameWnd::PreTranslateMessage(pMsg);

}

Запустите и увидите то же самое. На каждое действие будет писк. Дальше сообщение предлается Виду **CTestKeysView**. Здесь мы той же функцией можем его отследить.

BOOL CTestKeysView::PreTranslateMessage(MSG\* pMsg)

{

MessageBeep(-1);

return CView::PreTranslateMessage(pMsg);

}

Всё то же самое, писк на все сообщения. Ну давайте произведем отсев сообщений и будем ловить только те, которые приходят от нажатых клавиш. Мы можем отпеределить нажатие клавиши **KEYDOWN** и её отпускание **KEYUP**. Давайте обработаем нажатие.

void CTestKeysView::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)

{

// TODO: Add your message handler code here and/or call default

MessageBeep(-1);

CView::OnKeyDown(nChar, nRepCnt, nFlags);

}

Как же определить какая клавиша нажата ? Для этого есть параметр **bChar**, который можно сравнить в виртуальной клавишей, например, так:

if (nChar==VK\_F1) MessageBeep(-1);

Список их можно найти в **winuser.h**. Так же можно перегрузить сообщение **WM\_CHAR** извещающее о нажатии клавиши. Управление данной функции будет передано только если нажат конкретный символ. Например, на нажатие на клавишу **F1** она не реагирует.

void CTestKeysView::OnChar(UINT nChar, UINT nRepCnt, UINT nFlags)

{

MessageBeep(-1);

CView::OnChar(nChar, nRepCnt, nFlags);

}

Несколько отличается обработка нажатия клавиш в классе **COleControl**. Здесь появляется функция **OnKeyPressEvent** и функция **FireKeyDown** и так далее. Но об этом нужно разговаривать отдельно.

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/142.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?141) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?143) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc3.html)  
Автор **Каев Артем**.

## Шаг 143 - Читаем растр с диска

Очень интересно, но в **VC** нет такого простого класса, который может прочитать растровую картинку с диска, скажем **BMP**, и вывести её на экран. Ну нет и ладно, давайте создадим. Создавайте приложение **AppWizard** на основе **SDI** c именем **TestDib**.

Давайте теперь создадим новый класс через меню **Insert -> New Class** и имя ему дадим **CDib**. Вас спросят о его типе, давайте **Generic Class**. Вот каким он станет.

class CDib

{

public:

CDib();

virtual ~CDib();

};

Давайте изменим его так, чтобы в конструктор передавали имя файла и при создании объекта класса сразу создавался файл.

class CDib

{

public:

CDib(CString filename);

virtual ~CDib();

void LoadFile(CString filename);

};

Создаем код конструктора:

CDib::CDib(CString filename)

{

LoadFile(filename);

}

А теперь **LoadFile** код:

void CDib::LoadFile(CString filename)

{

CFile file(filename,CFile::modeRead);

BITMAPFILEHEADER bmf;

file.Read((void \*)&bmf,sizeof(bmf));

if (bmf.bfType != 0x4d42)

{

AfxMessageBox("Not Bmp File");

}

else

{

DWORD sizebmp=file.GetLength()-sizeof(bmf);

BYTE\* ptrBmp = (BYTE\*)GlobalAlloc(GMEM\_MOVEABLE,sizebmp);

file.Read((void\*)ptrBmp,sizebmp);

file.Close();

lpInfo=(LPBITMAPINFO)ptrBmp;

}

}

Мы добавили описание переменной в класс **CDib**:

class CDib

{

public:

LPBITMAPINFO lpInfo;

........

};

И код деструтора для уничтожения выделенной памяти.

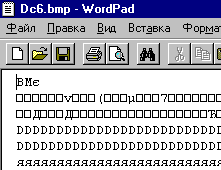
CDib::~CDib()

{

GlobalFree(lpInfo);

}

Это пока только болванка, но в следующих шагам мы её расширим до необходимого рабочего примера. Как видите самое интересное это функция **LoadFile**. Для начала мы открываем файл используя класс **CFile** и передавая в конструктор имя файла для открытия. Открываем его для чтения. Дальше мы зачитываем заголовок файла в структуру **BITMAPFILEHEADER**. Первое, что мы проверим это то, что этот файл именно **BMP**. Если Вы откроете файл **BMP** в текстовом редакторе, то увидите, что он начитается с **BM**, что гарантирует вроде как, что это именно то.



Эта информация будет находится в поле **bfType** структуры. Вот это мы и проверяем. После этого мы можем расчитать память отняв от размера файла размер заголовка. Выделить память и прочитать туда файл. После чего файл закрываем. Для сохранения указателя на файл мы завели новую переменную и присвоили её значения выделенной памяти. Кроме этого добавили в деструктор освобождение памяти при закрытии объекта класса.

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/143.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?142) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?144) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc3.html)  
Автор **Каев Артем**.

## Шаг 144 - Дальше Читаем растр с диска

Но то, что мы сделали в прошлый раз еще не все. Нам теперь надо завести несколько переменных и вычислить их положение в растре.

class CDib

{

.....

LPBITMAPINFOHEADER lpHeader;

RGBQUAD\* lpRGB;

BYTE\* lpBMP;

.......

};

А теперь код для вычисления этих переменных.

void CDib::LoadFile(CString filename)

{

........

// NEW !!!

lpHeader=(LPBITMAPINFOHEADER)ptrBmp;

lpRGB = (RGBQUAD\*)(ptrBmp+lpHeader->biSize);

int num=GetNumColor();

DWORD sizetable=num\*sizeof(RGBQUAD);

lpBMP=ptrBmp+lpHeader->biSize+sizetable;

}

}

Мы объявили процедуру для вычисления количества цветов.

class CDib

{

.....

UINT GetNumColor();

};

И её реализация.

UINT CDib::GetNumColor()

{

if ((lpHeader->biClrUsed == 0 ) && (lpHeader->biClrUsed <9))

return (1biClrUsed;

}

Вот давайте посмотрим на наши деяния. Мы объявили новую переменную, как указатель на структуру информации о растре. Сразу за ней начинается таблица цветов. Указатель на неё мы и вычисляем. Теперь надо вычислить размер таблицы. Для этого нужно количество цветов. Если мы умножим количество цветов на размер одной записи и сложим с началом записей, то получим указатель уже на конкретные данные.

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?143) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?145) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc3.html)  
Автор **Каев Артем**.

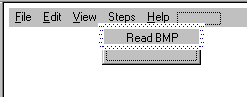
## Шаг 145 - Дальше Читаем растр с диска

Суббота 9.00 такое классное утро. Так хорошо спится. Только вот

телефон рядом с диваном :-( так поспать и не дали :-((((

Суббота 19:50 вот и новые шаги готовы :-)))

Ну что, попробуем наш еле живой класс на предмет чтения и помещения изображения на экран. Давайт создадим пункт меню типа **Read BMP**. Вот где.



Объявляем указатель на описание нашего класса.

// TestDibView.h : interface of the CTestDibView class

//

/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX\_TESTDIBVIEW\_H\_\_DE26E62D\_F772\_11D3\_B050\_000001260696\_\_INCLUDED\_)

#define AFX\_TESTDIBVIEW\_H\_\_DE26E62D\_F772\_11D3\_B050\_000001260696\_\_INCLUDED\_

#if \_MSC\_VER > 1000

#pragma once

#endif // \_MSC\_VER > 1000

#include "dib.h"

......

Ссылку на класс в классе вида.

class CTestDibView : public CView

{

...........

// Operations

public:

CDib \*cbmfile;

..........

}

Теперь конструктор и присвоение этой перменной значения **NULL**.

CTestDibView::CTestDibView()

{

// TODO: add construction code here

cbmfile=NULL;

}

По нажатию на пункт меню

void CTestDibView::OnStepsReadbmp()

{

if (cbmfile!=NULL) delete cbmfile;

cbmfile = new CDib("c:\\windows\\my.bmp");

RedrawWindow();

}

И функция рисования.

////////////////////////////////////////////////////

// CTestDibView drawing

void CTestDibView::OnDraw(CDC\* pDC)

{

CTestDibDoc\* pDoc = GetDocument();

ASSERT\_VALID(pDoc);

if (cbmfile!=NULL)

{

StretchDIBits(pDC->m\_hDC,10,10,100,100,0,0,100,100,cbmfile->lpBMP ,cbmfile->lpInfo,DIB\_RGB\_COLORS,SRCCOPY);

}

// TODO: add draw code for native data here

}

Из-за того, что когда я писал шаг меня отвлекли я проспал распределение памяти в **CDib.cpp**, вот необходимые изменения.

#include "windowsx.h"

.......

void CDib::LoadFile(CString filename)

{

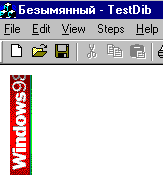
.....

BYTE\* ptrBmp = (BYTE\*)GlobalAllocPtr(GMEM\_MOVEABLE,sizebmp);

.....

}

Проверим ??? Только не забудьте указать имя своего **BMP** файла.



С нашим классом просто. Создаем его и выводим на экран в функции **OnDraw** с помощью **StretchDIBits**.

Ну наконец получилось. Если Вы заметили я очень быстро пробежался по многим вопросам, которым нужно уделить больше внимания. Я попробую немного попозже. Несмотря на то, что сегодня суббота меня честно говоря достали. Всё-таки работать тоже надо иногда. То одному чего-то надо, то другому чего-то надо. Вообщем караул. Так что за ошибки извеняйте. Я в свою очередь обещаю описать все мутные вопросы, которые встречались здесь в подробностях.

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/145.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?144) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?146) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc3.html)  
Автор **Каев Артем**.

## Шаг 147 - Отладка и TRACE

В каком-то шаге я рассказывал о ведении **LOG** файла для отладки или еще каких-то нужд. Но есть и другая возможность - это использование макросов **TRACE**. Это отладочные макросы и они не включаются в код при подготовке варианта **Relase**.

Давайте создадим приложение на базе диалогового окна с одной кнопкой. Вот код для неё.

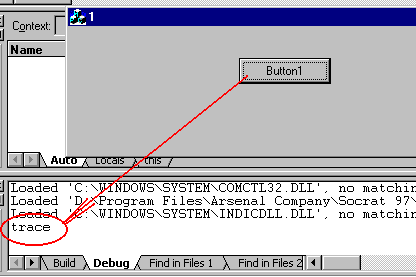
void CMy1Dlg::OnButton1()

{

TRACE("trace\n");

}

Вот теперь, если мы запустим приложение в отладочном варианте через **F5**, то при работе приложения при нажатии на кнопку в окне отладки появится наша запись.



Макросов **TRACE** много:

TRACE

TRACE0

TRACE1

TRACE2

TRACE3

Логика здесь такая. Цифра в конце указывает количество параметров. А сам макрос примерно тоже самое, что и функция **printf**. Вот так. Почему их так много это просто. Отладчик экономит место в сегменте данных. Ну и ладно. Вы можете пользоваться любым.

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?146) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?148) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc3.html)  
Автор **Каев Артем**.

## Шаг 148 - WAV файлы в ресурсах

по мотивам материалов присланных

---Исходное сообщение-----

От: Oleg

Кому: kaev@yandex.ru

Дата: 8 марта 2000 г. 20:40

Тема: MFCByStep

Проигрывание Wave-файла в виде ресурса

..........................

-------------------------

К сожалению материал был взят с какого то сайта

и его пришлось переработать.

Итак, задача следующая. Вы хотите создать программу, которая будет снабжена звуковым сопровождением. Вы решили использовать **WAV** файлы. У Вас есть два варианта, первый это хранить звуковые файлы отдельно, скажем в виде файлов. Это плохо, так как их могут стереть или испортить. Наиболее разумно поместить их внутри ресурсов.

Создаем приложение, как **MFC AppWizard** на базе диалогового окна. На окно помещаем кнопку для проигрывания файла. Имя проекта **TestWav**. Давайте поместим звуковой файл в ресурсы. Выбираем пункт меню **Insert -> Resource**. На появившемся окне есть кнопка **Import**. Нажмите её и у вас появится диалоговое окно выбора файла для импорта. В окне типа файлов надо выбрать **WAV** и найти файлы, например, в папке **Windows\Media**.

Посмотрите вкладку **Resource**, вы увидите папочку **WAVE**. Там и будет наш ресурс. Теперь код для кнопки, пока он такой:

void CTestWavDlg::OnButton1()

{

// TODO: Add your control notification handler code here

}

Итак, первое нам необходимо объявить заголовочный файл с функциями проигрывания **WAV** файлов

// TestWavDlg.cpp : implementation file

//

.......

#include "TestWavDlg.h"

#include "mmsystem.h"

.......

Незабудьте в **Project Setting** добавить **winmm.lib**. Ну, а теперь код.

void CTestWavDlg::OnButton1()

{

HRSRC hWav;

hWav=FindResource(AfxGetResourceHandle(),MAKEINTRESOURCE(IDR\_WAVE1),"WAVE");

if (hWav!=NULL)

{

HGLOBAL hPtrDataWav;

hPtrDataWav=LoadResource(AfxGetResourceHandle(), hWav);

if (hPtrDataWav!=NULL)

{

LockResource(hPtrDataWav);

sndPlaySound((LPCTSTR)hPtrDataWav,SND\_MEMORY|SND\_ASYNC );

FreeResource(hPtrDataWav);

}

else AfxMessageBox(" Not Load");

}

else AfxMessageBox("Not Resource");

}

Вот так это делается. Ищем ресурс, дальше читаем ресурс, блокируем его, а потом проигрываем как уже рассказывалось и разблокируем.

Поместить можно в файл ресурсов всё, что угодно и в последствии использовать !!!

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?147) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?149) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc3.html)  
Автор **Каев Артем**.

## Шаг 151 - Еще раз о Документ - Вид - Диалог

Hi!

Вот тут вопрос хотел задать, если можно. В шагах

искал, искал, и не нашел чего-то. Как реализовать

в однодокументном приложении диалог, который будет

выполнять ввод и вывод данных? Причем числовых

(Везде все почему-то на примере строк).

Может есть уже где такое?

Т.е. проблемы две в принципе... Как получить

из этого диалога доступ к документу и как

обработать введенные циферки.

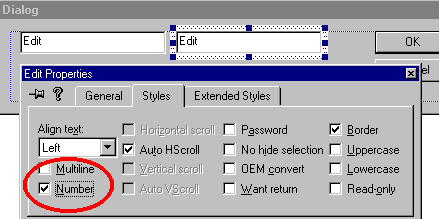
Заранее благодарен...

with best regards,

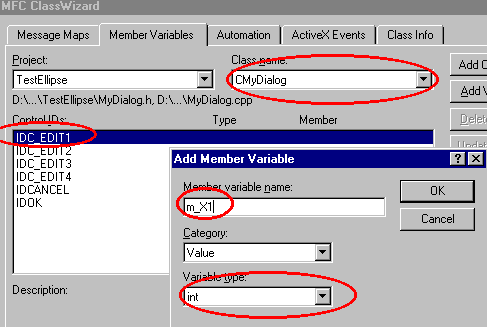
Vladimir V. Kvassov

Ну давайте созадим полный пример диалога. Задача такая. Наша программа будет рисовать эллипс. Параметры эллипса будут задаваться из диалогового окна. Эллипс будет записываться на диск и читаться. Вобщем полное короткое приложение.

Создаем приложение **MFC AppWizard**, как **SDI** и даем ему имя **TestEllipse**. Переходим на вкладку **Resoure View** и создаем новое диалоговое окно. Размещаем на этом диалоге 4 элемента редактирования (**CEdit**). Для того, чтобы в этот элемент управления вводились числа необходимо указать это в свойствах.



Теперь давайте из этого диалогового окна создадм класс. Это просто. Выделите диалог и вызовите **ClassWizard** об этом я точно писал. Дайте имя классу **CMyDialog**. Теперь нам надо к элементам диалога привязать переменные. Об этом я тоже писал. Только тип переменных надо дать **int**.



Вот что примерно должно быть у Вас написано в классе диалога.

class CMyDialog : public CDialog

{

..............

// Dialog Data

//{{AFX\_DATA(CMyDialog)

enum { IDD = IDD\_DIALOG1 };

int m\_X1;

int m\_Y1;

int m\_X2;

int m\_Y2;

//}}AFX\_DATA

.............

Класс диалога теперь готов к эксплуатации. Теперь нам нужно создать переменные в классе документа **CTestEllipseDoc**. Давайте с такими же именами, чтобы не путаться. В них и будут храниться параметры эллипса.

class CTestEllipseDoc : public CDocument

{

........

public:

int Y2;

int X2;

int Y1;

int X1;

........

};

Вот нам теперь их нужно иницилизировать. Лучше это сделать в функции **CTestEllipseDoc::OnNewDocument()**. Так как эта функция всегда вызывается при создании нового документа и можно задавать значения по умолчанию.

BOOL CTestEllipseDoc::OnNewDocument()

{

if (!CDocument::OnNewDocument())

return FALSE;

X1=10;

Y1=10;

X2=100;

Y2=100;

return TRUE;

}

Вот есть. Давайте теперь сделаем, чтобы этот эллипс рисовался, а это можно сделать в функции **Draw** класса вида.

void CTestEllipseView::OnDraw(CDC\* pDC)

{

CTestEllipseDoc\* pDoc = GetDocument();

ASSERT\_VALID(pDoc);

pDC->Ellipse(pDoc->X1,pDoc->Y1,pDoc->X2,pDoc->Y2);

}

Я хочу, что в этот диалог вызывался при нажатии правой кнопкой мыши на окне. Вот это событие я и превращу в функцию. Писал об этом.

void CTestEllipseView::OnRButtonDown(UINT nFlags, CPoint point)

{

// TODO: Add your message handler code here and/or call default

CView::OnRButtonDown(nFlags, point);

}

Для вызова класса диалога необходимо объявить заголовок.

// TestEllipseView.cpp : implementation of the CTestEllipseView class

//

........

#include "TestEllipseView.h"

#include "MyDialog.h"

........

Теперь сам вызов диалога и организация обмена данными.

void CTestEllipseView::OnRButtonDown(UINT nFlags, CPoint point)

{

CTestEllipseDoc\* pDoc = GetDocument();

CMyDialog cm;

cm.m\_X1 = pDoc->X1;

cm.m\_Y1 = pDoc->Y1;

cm.m\_X2 = pDoc->X2;

cm.m\_Y2 = pDoc->Y2;

if (cm.DoModal()==IDOK)

{

pDoc->X1=cm.m\_X1;

pDoc->Y1=cm.m\_Y1;

pDoc->X2=cm.m\_X2;

pDoc->Y2=cm.m\_Y2;

pDoc->UpdateAllViews(NULL);

}

CView::OnRButtonDown(nFlags, point);

}

Пишем на диск. То есть организуем процедуру записи на диск.

void CTestEllipseDoc::Serialize(CArchive& ar)

{

if (ar.IsStoring())

{

ar << X1;

ar << Y1;

ar << X2;

ar << Y2;

}

else

{

ar >> X1;

ar >> Y1;

ar >> X2;

ar >> Y2;

}

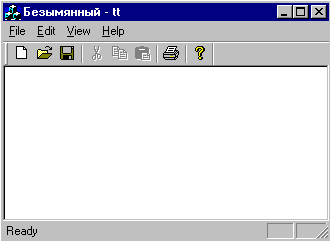
}

Ну вот и всё, можно тестировать. Попробуйте нажимайте правую кнопку. Появится диалоговое окно с параметрами по умолчанию. Поменяйте цифры нажмите **Ok**. Эллипс изменится. Только делайте изменения большие скажем со 100 на 200.

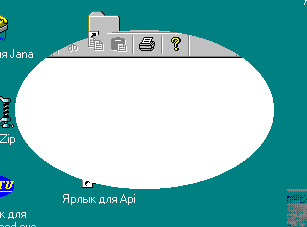
[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/151.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?150) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?152) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc4.html)  
Автор **Каев Артем**.

## Шаг 155 - Борьба с однообразием, круглое окно

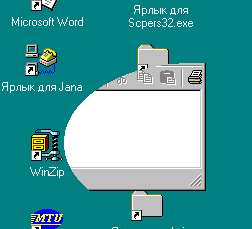
Сначала посмотрите как работает. Я просто в шоке. Сначало обычный запуск.



Теперь нажмем правую кнопку мыши.



При некоторой доли сноровки можно добиться и такого эффекта.



Нажмете левую кнопку и окно вернется к начальному виду.

Итак, создаем приложение как **MFC AppWizard**. Пусть оно будет **SDI** и имя у него будет **tt**. В классе приложения заведем переменную типа регион.

class CTtApp : public CWinApp

{

public:

HRGN rgn;

CTtApp();

........

};

И сохраним регион окна при инициализации окна.

////////////////////////////////////

// CTtApp initialization

BOOL CTtApp::InitInstance()

{

.......

GetWindowRgn(m\_pMainWnd->m\_hWnd,rgn);

return TRUE;

}

Вот теперь окно можно будет востановить. Код при нажатии на правую кнопку мыши.

void CTtView::OnRButtonDown(UINT nFlags, CPoint point)

{

AfxGetApp()->m\_pMainWnd->SetWindowRgn( CreateEllipticRgn(40, 40, 300, 200),true);

CView::OnRButtonDown(nFlags, point);

}

И левую при востановлении.

void CTtView::OnLButtonDown(UINT nFlags, CPoint point)

{

CTtApp\* tt=(CTtApp\*)AfxGetApp();

AfxGetApp()->m\_pMainWnd->SetWindowRgn(tt->rgn,true);

CView::OnLButtonDown(nFlags, point);

}

Вот и всё. Смотрите на код. Основой этого года является понятие регион. Это давайте договоримся что это область окна. Её можно получить использовав функцию **GetWindowRgn**. И поменять **SetWindowRgn** помня, что все эти функции находятся в классе **CWinApp**. Регионы бывают разные. Например, эллипсоидные. Вот такой регион мы и делаем. Чем же регион отличается от класса **CRect**. Только тем, что **CRect** всегда прямоугольник, а регион может принимать разные формы.

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/155.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?154) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?156) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc4.html)  
Автор **Каев Артем**.

## Шаг 156 - О регионах (CRgn)

Мы все с Вами привыкли, что Windows прямоугольный. Вроде к этому ведет всё. Формы, окна все они прямоугольные. Единственный способ бороться с этим - это применять регионы. Использование регионов позволит Вам создавать окна любой формы.

Регион в **Windows** - это замкнутая фигура, обычно для региона применяют английское слово **RGN**. В **MFC** есть соотвествующий класс **CRgn**. Как всегда этот класс соответствует чему-то в **Windows**. В **Windows** же есть понятие **HRGN**, то есть указатель на регион.

У каждого приложения есть регион. Вы его можете представить как область, за пределами которой окна не существует. То еть оно может быть, но пока есть регион он обрезает все, что выходит за его область. Любой класс порожденный от **CWnd** имеет регион. Для работы с регионами в этом классе предусмотрены две функции.

int SetWindowRgn( HRGN hRgn, BOOL bRedraw ); // установить регион

int GetWindowRgn( HRGN hRgn )const; // получить регион

Для того, чтобы использовать регион его нужно создать, регионы могут быть разной формы и создаются они по разному.

BOOL CreateRectRgn( int x1, int y1, int x2, int y2 );

// прямоугольные регион

BOOL CreateRectRgnIndirect( LPCRECT lpRect );

// тоже самое только входящие данные структура CRect

BOOL CreateEllipticRgn( int x1, int y1, int x2, int y2 );

// элипсоидный регион

BOOL CreateEllipticRgnIndirect( LPCRECT lpRect );

// тоже самое только входящие данные структура CRect

BOOL CreatePolygonRgn( LPPOINT lpPoints, int nCount, int nMode );

// регион из массива точек

BOOL CreatePolyPolygonRgn( LPPOINT lpPoints, LPINT lpPolyCounts, int nCount, int nPolyFillMode );

// регион из набора регионов

BOOL CreateRoundRectRgn( int x1, int y1, int x2, int y2, int x3, int y3 );

// регион с закруглеными краями

Но на этом все не заканчивается, регионы можно комбинировать и получать более новые и сложные фигуры.

int CombineRgn( CRgn\* pRgn1, CRgn\* pRgn2, int nCombineMode );

// комбинирование регионов.

У этой функции есть несколько параметров, которые отражают основные математические операции только с площадными объектами.

RGN\_AND пересечение регионов

RGN\_COPY копия

RGN\_DIFF не пренадлежащая часть

RGN\_OR объединение двух

RGN\_XOR объединение без пересечений

И теперь самое важное - это функции, которые определяеют попадание точки внуть региона. Это очень радует, так как регион может быть довольно сложной формы и круглый например. Меня данная возможность радует потому, что мне совсем не хочется код попадания в круг писать самому или в эллипс или в сложную фигуру, например, звезду.

BOOL PtInRegion( int x, int y ) const;

BOOL PtInRegion( POINT point ) const;

Для чего же можно использовать регионы ??? Ну во-первых различные формы окон и областей рисования. Именно с помощью регионов вы можете ограничивать области рисования придавая им самые немыслимые формы. Второе это определение попадания на в объект на экране. Так как регион может быть любой сложности, то есть возжность определения попадая мышью даже в очень сложные фигуры.

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?155) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?157) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc4.html)  
Автор **Каев Артем**.

## Шаг 158 - Обновление вида из панели диалога

-----Исходное сообщение-----

От: Denis Lazebnik

Кому: kaev@yandex.ru

Дата: 18 марта 2000 г. 11:52

Тема: Идея о новом шаге в MFC

Привет!

При написании программы на Visual C++ 6.0 я наткнулся на небольшую

проблемму и подумал что для её решения можно уделить целый шаг на сайте.

Проблемма такова:

Скажем у меня самая простая программа SDI и нажатием левой кнопки крысы у

меня появляется Modal Dialog в котором введя какую-то строку текста и

нажав OK этот текст у меня появляется в окне View. В мой Dialog я хочу

добавить кноку Update Now при нажатии которой, текст в окне View обновляется,

но сам Dialog не закрывается.

Спасибо.

Задача мне понравилась. Спасибо Денис. Вот и давайте её реализуем. Ну во-первых делаем приложение **SDI** c именем **TestOutDialog** c помощью **AppWizard**. Теперь обычные вещи. Делаем диалоговое окно с полем редактирования **CEdit** и кнопкой **Update**. Для диалогового окна длеаем класс. Об этом много раз писал как. Называем этот класс **CMyDialog**. Этот класс унаследует все функции класса **CDialog**. Теперь некоторые нюансы. Данные должны храниться в классе документа. У нас класс документа называется **CTestOutDialogDoc**, кроме того класс документа умеет обновлять все виды связанные с ним. Поэтому для реализации необходимой нам кнопки вот чего надо. Надо, чтобы в классе диалога была ссылка на класс документа и тогда ...... . Вот давайте конструктор класса диалога и изменим, и еще сделаем ссылку на класс документа.

#include "TestOutDialogDoc.h"

/////////////////////////////////////////

// CMyDialog dialog

class CMyDialog : public CDialog

{

// Construction

public:

CMyDialog(CTestOutDialogDoc\* pD,CWnd\* pParent = NULL); // standard constructor

protected:

CTestOutDialogDoc\* pDoc;

.........

};

Код конструктора тоже надо изменить. Вот так, чтобы сохранить в диалоге ссылку на документ.

CMyDialog::CMyDialog(CTestOutDialogDoc\* pD,CWnd\* pParent)

: CDialog(CMyDialog::IDD, pParent)

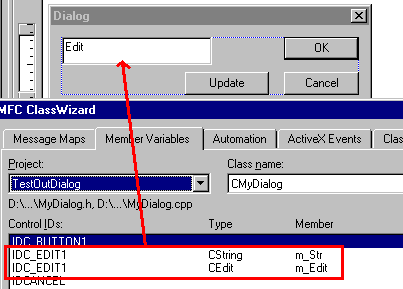
{

pDoc=pD;

......

}

С этим разобрались. Теперь в любой момент мы будем из класса диалога иметь доступ к классу документа. Давайте займемся нашим полем редактирования. Вы когда-нибудь пробовали к полю редактирования привязать две переменных типа строки и элемента управления. Не пробовали тогда сделайте вот так.



Зачем я это делаю ? Проблема в получении текста из строки редактирования. Единственный гарантированный способ получить текст если завести переменную типа элемент управления и воспользоваться функцией **GetWindowText** не дожидаясь механизма **DDX**. А со строкой работать удобнее во всех других случаях. Давайте посмотрим код к кнопке.

void CMyDialog::OnButton1()

{

char buffer[100];

m\_Edit.GetWindowText(buffer,sizeof(buffer)); ;

pDoc->str=buffer;

pDoc->UpdateAllViews(NULL);

}

При нажатии на кнопку получаем текст, потом передаем его в класс документа и вызываем функции обновления видов. Теперь нужно создать пункт меню для вызова диалога. Делали, писать не буду. Имя ему **New Str**. Код нужно, чтобы обрабатывался в классе вида. Тоже писалось. А вот и код.

void CTestOutDialogView::OnTestNewstr()

{

CTestOutDialogDoc\* pDoc = GetDocument();

CMyDialog cm(pDoc);

cm.m\_Str=pDoc->str;

cm.DoModal()

}

Передаем класс документа. Правда перед выводом на экран передаем переменную из класса документа, в которой хранится строка. Вот её описание.

class CTestOutDialogDoc : public CDocument

{

.......

public:

CString str;

......

};

И начальная инициализация при создании нового документа.

BOOL CTestOutDialogDoc::OnNewDocument()

{

if (!CDocument::OnNewDocument())

return FALSE;

str="Hello MFC";

return TRUE;

}

В классе вида вывод на экран.

void CTestOutDialogView::OnDraw(CDC\* pDC)

{

CTestOutDialogDoc\* pDoc = GetDocument();

ASSERT\_VALID(pDoc);

pDC->;TextOut(10,10,pDoc->str);

}

Идея была рассказана следующая. В класс диалога был передан указатель на класс документа. С помощью этого указателя изменялись данные и обновлялся экран без закрытия диалога.

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/158.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?157) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?159) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc4.html)  
Автор **Каев Артем**.

## Шаг 159 - Некоторые обобшения - Виды ввода

Весна... Тянет на философию и общение.

Особенно перед выборами президента.

Мы с Вами довольно долго работаем с разными сообщениями, которые являются следствием ввода. Пора глобально взглянуть откуда берутся сообщения.

Сообщения могут идти от клавиатуры. Вот такие.

WM\_KEYDOWN Пользователь нажал клавишу

WM\_KEYUP Пользователь отпустил клавишу

WM\_SYSKEYDOWN Пользователь нажал системную клавишу

WM\_SYSKEYUP Пользователь отпустил системную клавишу

Ввод символа это не тоже самое, что нажал кнопку хотя принцип тот же. Для того, чтобы получить символ нужно нажать и отпустить кнопку. В данном случае эти сообщения транслируются в одно сообщение, что в случае самостоятельной реализации ввода символов в окно удобно. Например, легко запретить вводить ряд символов :-). Кодировка символов **ANSI**

WM\_CHAR Пользователь ввел символ

WM\_SYSCHAR Пользователь ввел системный символ

Да системные клавиши это все кроме букв и цифр, то есть **Alt** или, например, **F10** разбивка сообщений на эти два вида тоже искусственная. Но удобна.

От мыши.

WM\_MOUSEMOVE Пользователь перемещает курсор мыши в/через окно.

WM\_LBUTTONDOWN Пользователь нажимает левую кнопку.

WM\_LBUTTONUP Пользователь отпускает левую кнопку.

WM\_LBUTTONDBLCLK Пользователь нажимает, отпускает и

вновь нажимает левую кнопку (в рамках отпущенного системой времени).

WM\_MBUTTONDOWN Пользователь нажимает среднюю кнопку.

WM\_MBUTTONUP Пользователь отпускает среднюю кнопку.

WM\_MBUTTONDBLCLK Пользователь нажимает, отпускает и

вновь нажимает среднюю кнопку(в рамках

отпущенного системой времени).

WM\_RBUTTONDOWN Пользователь нажимает правую кнопку.

WM\_RBUTTONUP Пользователь отпускает правую кнопку.

WM\_RBUTTONDBLCLK Пользователь нажимает, отпускает и

вновь нажимает правую кнопку (в рам-

ках отпущенного системой времени).

Сообщение от мыши Вы получите только в том случае, если мышка внутри окна. Ради справедливости можно сказать, что можно ловить сообщения и вне окна, для этого нужно использовать **SetCapture**, тогда весь ввод от мыши будет в окне. Мышь будет у Вас пока вы её не освободите с помощью **ReleaseCapture**. Окно, которое ухватило мышь можно вычислить с помощью **GetCapture**. Двойной щелчок окно может получать не всегда. Только если стиль его **CS\_DBLCLKS**. **MFC** заботится об этом и поэтому он уже установлен.

От таймера

WM\_TIMER Таймер сработал

Скорость посылки сообшений таймером устанавливается в милисекундах. В одной секунде 1000 милисекунд. Расчет простой считайте.

От полос прокрутки

WM\_HSCROLL Пользователь жмет горизонтальную прокрутку

WM\_VSCROLL Пользователь жмет вертикальную прокрутку

От меню

WM\_COMMAND Пользователь выбрал пункт меню

WM\_SYSCOMMAND Пользоваитель выбрал пункт системного меню

Ну и на всякий случай как их все отловить. Нажмите правую кнопку на классе и выберите пункт меню **Add Windows Message Handle**. Все они будут перед вами.

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?158) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?160) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc4.html)  
Автор **Каев Артем**.

## Шаг 160 - Заполняем список из БД

Задача такая. Создать простое приложение на основе диалоговой панели с окном списка, которое заполнится из файла БД. За основу возьмем базу данных **Access** и механизм **DAO**. В базе данных будет одна таблица с именем **test**. Эта таблица взята из реального проекта поэтому там первое поле ключевое, второе фамилия (**family**), а третье отчество. Не надо критики. Идет разговор о базе данных по материалам работ 1868-2000 года. Так вот в геологических отчетах младше 1950 годов отчество уже не восстановишь. Вот поэтому-то и поля совмещенные.

Создавайте приложение на базе диалогового окна и помещайте на него элемент управления список. К этому элементу управления нужно добавить переменную типа **Control**, чтобы можно было с ней работатб. Вот так.

class CTestBaseDlg : public CDialog

{

......

enum { IDD = IDD\_TESTBASE\_DIALOG };

CListBox m\_ListBox;

.....

};

Список будет заполняться при запуске диалоговой панели и поэтому наиболее удобно его заполнить в функции **OnInitDialog()**.

BOOL CTestBaseDlg::OnInitDialog()

{

..........

// TODO: Add extra initialization here

ListFullDao();

return TRUE; // return TRUE unless you set the focus to a control

}

Всё заполнение я решил вынести в отдельную функцию дабы не загромождать инициализацию. А вот и она.

void CTestBaseDlg::ListFullDao()

{

AfxDaoInit();

CDaoWorkspace cw;

CDaoDatabase cd(&cw);

CDaoTableDef cf(&cd);

CDaoRecordset cr;

cw.Open();

if (cw.IsOpen())

{

cd.Open("d:\\TestDB\\test.mdb");

if (cd.IsOpen())

{

cf.Open("test");

if (cd.IsOpen())

{

cr.Open(&cf);

while (!cr.IsEOF())

{

COleVariant val;

cr.GetFieldValue("family",val);

char buf[256];

sprintf(buf,"%s",val.bstrVal);

m\_ListBox.AddString(buf);

cr.MoveNext();

}

}

else AfxMessageBox("Error Open Table");

}

else AfxMessageBox("Error Open File");

}

else AfxMessageBox("Not Open WorkSpace");

cr.Close();

cf.Close();

cd.Close();

cw.Close();

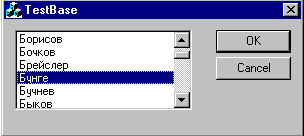
AfxDaoTerm();

}

Итак, как видите все тривиально. Инициализируем **DAO** функцией **AfxDaoInit()**, открываем рабочее пространство **CDaoWorkspace** это просто имя области работы можно оставить **NULL**. Дальше открываем саму базу **CDaoDatabase**, из базы открываем таблицу **CDaoTableDef**, на основе которой делаем набор записей **CDaoRecordset** постоянно проверяя на корректность функцией **IsOpen()**. После открытия записей мы можем бежать по записям используя **MoveNext()**, когда дойдем до конца получим **IsEOF**.

Немного сложнее с получением самого значения. Для этого используя функцию **GetFieldValue** и имя поля (можно его номер) получаем содержимое в переменную **COleVariant**. Чтобы в поле не находилось оно поместится туда. Такая переменная хорошая. Так как тип её я знаю, то и извлеку её в строку. Вот.

Давайте протестируем. Только не забудьте указать Ваш путь к БД.



Вы видите эту фамилию (Бунге). Этот мужик в 1886 году от Российской Академии Наук изучал Новосибирские острова.

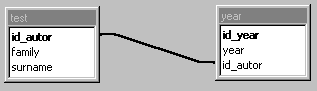
[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/160.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?159) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?161) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc4.html)  
Автор **Каев Артем**.

## Шаг 161 - Дальше о заполнении списка из БД, связанные таблицы

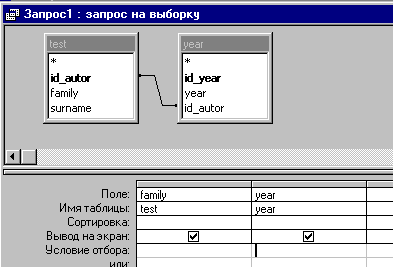
В реальных проектак конечно все сложнее. Существуют взаимосвязанные таблицы. Только вот на чьи плечи возложить ответственность за взаимосвязь таблиц ??? **Microsoft** предлагает разделить любую программу логически на 3 слоя.

1. данные
2. логика
3. интерфейс

Правда я нигде не встречал подробного объяснения, что понимается под логикой. Так вот, если у нас две связанные таблицы, то кто будет ответственен за связь ??? БД или приложение ??? Давайте попробуем возложить ответственность на БД. Раз уж она называется **Access**, что вроде означает "доступ", так пусть и работает :-))) Вот схема данных.



Мы договаривались, что ответственность за связь возмет БД. Поэтому создадим в БД запрос.



Имя ему дадим **Query1**.

Вот пора делать программу. Будет изменять предыдущий проект. Итак, теперь мы будет работать не на основе таблицы, а на основе запроса. Значит надо поменять тип класса.

CDaoQueryDef cf(&cd);

// CDaoTableDef cf(&cd);

Идея простая, с таблицами работают одни классы, а с запросами другие. Но класс **CDaoRecordset** можно строить и на основе таблицы и на основе запроса. Поэтому всё остается похоже. Только при открытии набора записей нужно указать имя запроса.

if (cd.IsOpen())

{

cf.Open("Query1");

if (cd.IsOpen())

Что произойдет при выполнении запроса ??? Две таблицы превратятся в одну. Вот так. Теперь мы можем выводить и фамилию и год.

char buf[256];

COleVariant val;

cr.GetFieldValue("family",val);

COleVariant valyear;

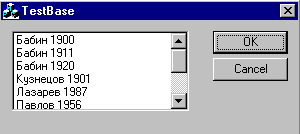
cr.GetFieldValue("year",valyear);

sprintf(buf,"%s %s",val.bstrVal,valyear.bstrVal);

m\_ListBox.AddString(buf);

cr.MoveNext();

Давайте запустим и посмотрим как это будет выглядеть.



Видите фамилия и год рядом, т.е. таблицы слились. Некрасиво скажете Вы. Ну это дело поправимое. Чего хотели того добились !!!

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/161.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?160) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?162) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc4.html)  
Автор **Каев Артем**.

## Шаг 162 - Дальше о заполнении списка из БД , улучшения

То, что мы сделали в [прошлом шаге](http://www.firststeps.ru/mfc/steps/r.php?161) это конечно круто. Но было бы еще круче, если бы мы нажимали на авторе и рядом бы появлялся список с годами. Почти как по настоящему :-). Ну давайте берите прошлый проект. Помещайте рядом элемент список и имя ему давайте дадим **m\_ListBoxExt** связывая с переменной типа элемент управления.

class CTestBaseDlg : public CDialog

{

..............

// Dialog Data

//{{AFX\_DATA(CTestBaseDlg)

enum { IDD = IDD\_TESTBASE\_DIALOG };

CListBox m\_ListBoxExt;

CListBox m\_ListBox;

//}}AFX\_DATA

..............

};

Так как пользоваться классами БД мы будет несколько раз, наиболее общие надо бы вынести в объявления на уровне класса.

class CTestBaseDlg : public CDialog

{

// Construction

public:

void ListFullDao();

CTestBaseDlg(CWnd\* pParent = NULL); // standard constructor

CDaoWorkspace cw;

CDaoDatabase\* cd;

.............

};

Запуск **DAO** надо бы перенести в инициализацию диалога и инициализацию основных классов туда же.

///////////////////////////////////////////////

// CTestBaseDlg message handlers

BOOL CTestBaseDlg::OnInitDialog()

{

..................

// TODO: Add extra initialization here

AfxDaoInit();

cd = new CDaoDatabase(&cw);

cw.Open();

if (cw.IsOpen())

{

cd->Open("d:\\TestDB\\test.mdb");

if (cd->IsOpen())

{

ListFullDao();

return TRUE;

}

}

return FALSE;

}

Это знакомо. Соотвественно функция **ListFullDao()** упростится.

void CTestBaseDlg::ListFullDao()

{

CDaoTableDef cf(cd);

CDaoRecordset cr;

cf.Open("test");

if (cf.IsOpen())

{

cr.Open(&cf);

while (!cr.IsEOF())

{

char buf[256];

COleVariant val;

cr.GetFieldValue("family",val);

sprintf(buf,"%s",val.bstrVal);

m\_ListBox.AddString(buf);

cr.MoveNext();

}

}

else AfxMessageBox("Error Open Table");

cr.Close();

cf.Close();

}

Закрывать глобальные классы и останавливать выполнение **DAO** мы будет путем перегрузки функции **DestroyWindow**.

BOOL CTestBaseDlg::DestroyWindow()

{

// TODO: Add your specialized code here and/or call the base class

cd->Close();

delete cd;

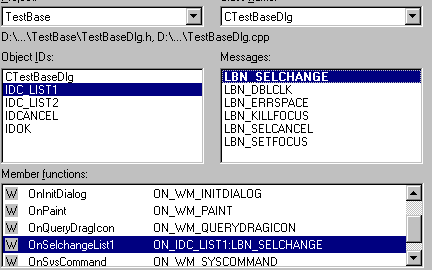
cw.Close();

AfxDaoTerm();

return CDialog::DestroyWindow();

}

Для реализации выборки по выбору фамилии нам необходимо перехватить сообщение от списка. Сделать это можно использовав **ClassWizard**.



Ну и код естественно.

void CTestBaseDlg::OnSelchangeList1()

{

CString s;

char SQLbuff[256];

m\_ListBox.GetText(m\_ListBox.GetCurSel(),s);

for (int x=0;x<(m\_ListBoxExt.GetCount()+1);x++)

m\_ListBoxExt.DeleteString(x);

CDaoQueryDef cf(cd);

CDaoRecordset cr;

cd->DeleteQueryDef("MFCtemp");

sprintf(SQLbuff,"SELECT \* FROM Query1 WHERE family='%s';",s);

cf.Create("MFCtemp",SQLbuff);

cf.Append();

cf.Close();

cf.Open("MFCtemp");

if (cf.IsOpen())

{

cr.Open(&cf);

if (cr.IsOpen())

{

while (!cr.IsEOF())

{

char buf[256];

COleVariant val;

cr.GetFieldValue("year",val);

sprintf(buf,"%s",val.bstrVal);

m\_ListBoxExt.AddString(buf);

cr.MoveNext();

}

}

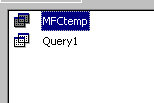
}

cr.Close();

cf.Close();

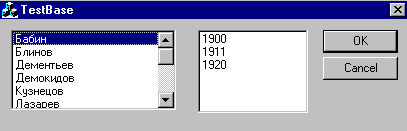
}

Логика такая. Мы делаем набор строк **CDaoRecordset** на основе запроса. Только вот формировать запрос будете сами ;-) Смотрите код. Для начала мы удалим запрос из БД, а потом создадим его заново. Он физически будет добавлен в БД **ACCESS**. Вот он.



Итак. Удаляем старый **cd->DeleteQueryDef("MFCtemp");** и создаем новый **cf.Create("MFCtemp",SQLbuff);** этот новый запрос помещаем в БД **cf.Append();** закрываем его и открываем снова для того, чтобы он явился источником строк. Вот и вся идея. А дальше как всегда побежали по полям по делам.

Можно тестировать.



Работает !!! А куда он денется :-). Новая версия **MDB** в проекте.

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/162.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?161) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?163) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc4.html)  
Автор **Каев Артем**.

## Шаг 163 - Где хранить данные ???

Hi! Классный сайт!!!

По мере изучения материала у меня накопилась масса вопросов.

Если не очень трудно и вопросы покажутся не детскими, то ответьте пожалуйста.

1. Ситуация: в диалоговом окне несколько Edit`ов. При инициализации диалога

в них нужно поместить цифры. Пользователь может сам назначить какие цифры

он хочет там видеть. Я сохраняю эти цифры в файле (текстовом). Где и как

еще можно их запомнить. Остальные вопросы потом, чтобы сильно не загружать.

Заранее благодарен, Борис (bor1999@mail.ru)

Хранить можно в самом простом варианте вот где.

1. Текстовый файл
2. INI файл

Давайте рассмотрим каждый вариант. Начнем с текстового файла. Создавайте приложение на базе диалогового окна с именем **TestWrite** и помещайте на него два элемента редактирования **CEdit**. Дайте им имена и свяжите с переменной.

class CTestWriteDlg : public CDialog

{

.......

CString m\_M1;

CString m\_M2;

//}}AFX\_DATA

........

};

Теперь нужно решить где инициализировать и сохранять данные. Можно поступить так. Самое удобно производить инициализацию в классе **OnInitDialog()**, а вот запись можно по разному. Например, перегрузить нажатие на кнопку **OK**. Я решил по другому, что введенные данные будут сохраняться не в зависимости от нажатия на кнопки, а в момент разрушения окна и поэтому перегрузил **OnDestroy()**. Ну, а вообще это Вам решать.

Итак, давайте посмотрим какой код нужно создать в случае данных в текстовом формате. Кстати хранение в текстовом файле и в бинарном очень схожи по принципу. Так как все данные будут у нас строковые, то с бинарным форматом и возиться не будем. Код я поместил в функции **ReadDataFile** и **WriteDataFile**. Так вот когда будем смотреть следующую возможность, то вызовы функций этих я закоментирую, но в проекте они будут все.

Вот текст этих двух функций

void CTestWriteDlg::ReadDataFile()

{

try

{

CStdioFile cf;

cf.Open("test.txt", CFile::modeRead);

cf.ReadString(m\_M1);

cf.ReadString(m\_M2);

cf.Close();

}

catch(...)

{

AfxMessageBox("Error Read ");

}

UpdateData(FALSE);

}

void CTestWriteDlg::WriteDataFile()

{

UpdateData(FALSE);

try

{

CStdioFile cf;

cf.Open("test.txt",CFile::modeCreate | CFile::modeWrite );

cf.WriteString(m\_M1+"\n");

cf.WriteString(m\_M2);

cf.Close();

}

catch(...)

{

AfxMessageBox("Error Write");

}

}

Хитрого нет ничего. Вот только **UpdateData(FALSE)** вызывается для того, чтобы заставить силовым методом обменяться данными.

С использованием **INI** файлов проще. Вот код.

void CTestWriteDlg::ReadIniData()

{

m\_M1=AfxGetApp()->GetProfileString("DATA","M1");

m\_M2=AfxGetApp()->GetProfileString("DATA","M2");

UpdateData(FALSE);

}

void CTestWriteDlg::WriteIniData()

{

UpdateData(FALSE);

AfxGetApp()->WriteProfileString("DATA","M1",m\_M1);

AfxGetApp()->WriteProfileString("DATA","M2",m\_M2);

}

А еще можно в реестре и в базе данных. Но это сложнее. Мне кажется два эти варинта наиболее простые. Если данные сложные, то их можно организовать в структуры и писать на диск сразу структуру. Когда будете тестировать работу с файлом незабудьте переключиться в режим **Release** для обрабоки исключений программным путем.

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/163.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?162) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?164) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc4.html)  
Автор **Каев Артем**.

## Шаг 165 - Диалог - добавим меню

Использование диалогового окна в качестве главного окна программы иногда бывает очень удобно, особенно в случаях, когда программа имеет простой интерфейс и выполняет узкий круг задач, например, утилиты конвертации файлов - ну что там сложного: спроси юзера где входной файл, пару вопросов о типе конвертации и можно формировать выходной файл. Но иногда и в таких программах хочется иметь привычные элементы интерфейса: меню, акселераторы, изменяемый размер окна...Ладно, хватит болтать, лучше жева... то-есть, писать.

Итак, быстренько делаем заготовку для творчества: запускаем Visual Studio, File/New.../MFC AppWizard. Project Name: AdvDlg, OK. Dialog based, Finish, OK. Теперь можно поизменять размеры будущего диалога, потаскать кнопочки, добавить что-нибудь от себя. Я подредактировал поле "static" с текстом "Todo..." (заменил текст на "File:") и добавил строку редактирования **IDC\_EDIT** и кнопку **IDC\_BROWSE**, потаскать кнопки тоже не забыл, в итоге получилась такая симпатяжка:



Можно откомпилировать и запустить. Работает? Продолжим. Задача: добавить меню, пока что с единственным пунктом - "Help/About..." Я не буду рассказывать, как создать меню - пора бы уже научиться, скажу лишь, что само меню у меня имеет идентификатор **IDR\_MENU**, пункт "About..." - **ID\_ABOUT**. Теперь будем приклеивать меню к диалогу: на закладке с ресурсами проекта находим ресурс нашего главного диалога (у меня - **IDD\_ADVDLG**) и двойным кликом вновь открываем редактор диалога. Один левый тычок мыши в заголовок диалога, чтобы выбрать именно диалог, теперь правый тык и пункт контекстного меню "Properties". В открывшемся окне "Dialog properties", на закладке "General" нужно найти указать ресурс меню для нашего диалога - **IDR\_MENU**. Компиляция, контрольный запуск:



Неплохо, но, во-первых, в нашем случае меню выглядит не слишком красиво из-за отсутствия подчеркивающей его линии (чтобы было понятнее, посмотрите на меню Visual Studio), а во-вторых, отсутствует какая-либо реакция на выбор пункта **About**. Для решения первой проблемы я просто вставил в наш диалог еще один "контрол" - статическая картинка, изменил ее размеры до горизонтальной линии и оттащил на самый верх окна. Затем немного поигрался со свойствами созданной разделительной линии и получил во такой результат:



Ну, справиться со вторым уже гораздо проще, особенно если учесть, что само окно "About" для нас уже давно создал Волшебник (как еще можно назвать **AppWizard**). Итак, **ClassWizard**(Ctrl+W), для объекта с идентификатором **ID\_ABOUT** назначаем обработчик сообщения "COMMAND". После чего добавляем в обработчик пару строк.

void CAdvDlgDlg::OnAbout()

{

// TODO: Add your command handler code here

CAboutDlg dlgAbout;

dlgAbout.DoModal();

}

Собственно, все. Ну, а чтобы Вам не казалось, что диа4логи имеют слишком много ограничений, в одном из следующих шагов мы научимся использовать в диалоге акселераторы...

Шаг прислал Галицкий Игорь ([ig@ntvplus.ru](mailto:ig@ntvplus.ru))

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?164) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?166) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc4.html)  
Автор **Каев Артем**.

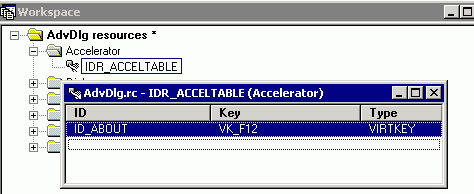
## Шаг 166 - Диалог - добавим акселераторы

Акселераторы (accelerators) позволяют существенно упростить пользователю работу с Вашей программой за счет быстрого доступа к ее элементам. Нажатием комбинации клавиш "Ctrl + O" в Microsoft Word Вы мгновенно попадаете в диалог открытия документа, минуя потери времени на перебирание пунктов меню. Для человека, знакомого хотя бы с одной программой Windows, многие комбинации клавиш других программ уже известны, например:

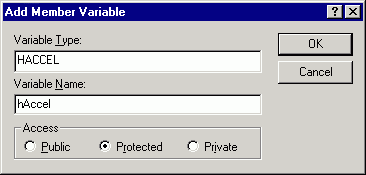
|  |  |
| --- | --- |
| Alt+X | Выход из программы |
| Ctrl+C (Ctrl+Ins) | Копирование данных в буфер обмена (clipboard) |
| Ctrl+X (Shift+Del) | Удаление данных с копированием их в буфер обмена |
| Ctrl+V (Shift+Ins) | Вставка данных из буфера обмена |
| Ctrl+Z (Alt+BackSpace) | Отказ от последней операции (UNDO) |

К счастью, чтобы использовать в программе акселераторы, не требуется почти никаких затрат, поскольку поддержка акселераторов встроена в операционную систему. Но, как обычно, не может быть все столь хорошо. Дело в том, что сначала таблицу акселераторов необходимо загрузить из ресурсов Вашей программы (впрочем, Вы можете создать ее в памяти программно, но сейчас об этом не будем), после чего вызывать функцию **TranslateAccelerator** в цикле обработки сообщений главного окна программы. Вот здесь-то и кроется засада, поскольку в модальном диалоге цикл обработки сообщений скрыт где-то с недрах операционки и в явном виде недоступен. Конечно, можно было бы создать безмодальный диалог, но это потребует дополнительных усилий программиста - например, в случае вложенных безмодальных диалогов придется изрядно попотеть, стараясь недопустить переключения между диалогом-родителем и диалогом-потомком. К счастью, программеры из одной чрезвычайно крупной компании при разработке MFC поступили весьма интересно, поскольку в MFC нет модальных диалогов в чистом виде - при использовании функции **DoModal** в наследнике **CDialog** создается типичный безмодальный диалог с циклом выборки сообщений, а "модальность" такого диалога эмулируется программно. Благодаря такому решению у нас есть доступ к циклу сообщений, а для реализации акселераторов большего нам и не надо.

За основу возьмем все тот же проект (см. Шаг 165). Переходим на закладку "ResourceView", выбираем **AdvDlg resources** и давим правую кнопку. В контекстном меню необходимо выбрать пункт **Insert...**, а в появившемся диалоге создать новый ресурс - таблицу акселераторов. Вновь созданной таблице назначим идентификатор **IDR\_ACCELTABLE**, после чего дважды кликаем на первой строчке таблицы. В диалоге создания акселератора поле **ID**, как обычно, задает идентификатор сообщения, которое будет посылаться окну при нажатии заданной комбинации клавиш. Поле **Key** можно заполнить и вручную, но для этого необходимо, как минимум, знать принятые в Windows обозначения клавиш, поэтому мы пойдем простым путем и нажмем кнопку "Next key typed", после чего кнопку клавиатуры **F12**, поле **Key** будет заполнено автоматически кодом **VK\_F12**. На этом создание таблицы акселераторов закончим.



Как я уже говорил, перед использованием акселераторов они должны быть загружены в память, для чего используется функция **LoadAccelerators**, которая вернет нам хэндл загруженной таблицы акселераторов. Хэндл придется сохранить, поскольку он используется в качестве параметра функции **TranslateAccelerator**, поэтому в Visual Stidio перейдем в на закладку "ClassView" и, после правого тыка на классе основного диалога, выбираем пункт меню **Add member variable...** (добавить переменную - член класса). Чтобы не описывать процесс добавления, лучше я покажу картинку:



Почему я выбрал тип доступа **Protected**? Правила хорошего тона C++: не надо давать доступ к членам класса больше, чем это действительно необходимо, поскольку помогает избавиться от глупой модификации переменных класса вне класса. К сожалению, от неглупой модификации это не спасает :-).

Теперь в конструкторе диалога добавляем код загрузки таблицы акселераторов:

CAdvDlgDlg::CAdvDlgDlg(CWnd\* pParent /\*=NULL\*/)

: CDialog(CAdvDlgDlg::IDD, pParent)

{

//{{AFX\_DATA\_INIT(CAdvDlgDlg)

// NOTE: the ClassWizard will add member initialization here

//}}AFX\_DATA\_INIT

// Note that LoadIcon does not require a subsequent DestroyIcon in Win32

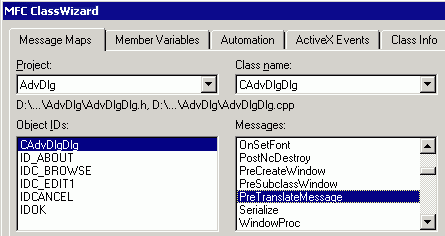
m\_hIcon = AfxGetApp()->LoadIcon(IDR\_MAINFRAME);

**hAccel = LoadAccelerators( AfxGetResourceHandle(), MAKEINTRESOURCE( IDR\_ACCELTABLE ) );**

}

Несколько слов об **MAKEINTRESOURCE()**. Во первых, это не функция, а макрос, что вовсе не уменьшает его полезности. Дело в том, что большинство функций Win32, работающих с ресурсами, для ссылок на ресурсы используют их текстовые имена. Однако в обычной практике программирования для ресурсов используют числовые идентификаторы, поскольку это уменьшает размер программы. Макрос **MAKEINTRESOURCE()** как раз и позволяет выполнить преобразование числового идентификатора ресурса к символьному указателю.

А теперь нам осталось нанести тот самый завершающий штрих, после которого картина оживет... Добавляем виртуальную функцию **PreTranslateMessage**, которая замечательна тем, что вызывается при поступлении очередных (т.е. из очереди) сообщений, но еще до того, как вызвать стандартные обработчики. Так что запускайте **ClassWizard** и творите чудеса. Ну, а если есть сомнения в своих силах - смотрите картинку:



Осталось слегка модифицировать тело **PreTranslateMessage**:

BOOL CAdvDlgDlg::PreTranslateMessage(MSG\* pMsg)

{

// TODO: Add your specialized code here and/or call the base class

**if ( TranslateAccelerator( m\_hWnd, hAccel, pMsg ) ) return TRUE;**

return CDialog::PreTranslateMessage(pMsg);

}

Все. Компиляция, сборка, пробный запуск (на языке акселераторов Visual Studio - F7, Ctrl+F5 :-). Вы еще не забыли, что в качестве акселератора в программе мы выбрали **F12**? Если нет - давите ее и смотрите за результатом...

Шаг прислал Галицкий Игорь ([ig@ntvplus.ru](mailto:ig@ntvplus.ru))

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/166.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?165) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?167) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc4.html)  
Автор **Каев Артем**.

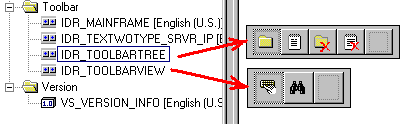
## Шаг 167 - Стиль окна Exploler и панели инструментов

В [шаге 164](http://www.firststeps.ru/mfc/steps/r.php?164) мы с Вами уже создавали стиль **Exploler**, вот теперь давайте к этому проекту добавим свои панели инструментов и посмотрим как их активность будет зависеть от фокуса вида. Напоминаю, что стиль **Exploler** разбивает окно на два вида.

**CLeftView** - это левый вид и порожден этот вид от класса **CTreeView**.

**....View** - это правый вид и он порожден в зависимости от выбранного Вами вида при генерации приложения.

Итак, создаем в ресурсах две новых панели инструментов. Обязательно на них надо нарисовать кнопки. Если просто создать панели и не добавить кнопки, то при компиляции они пропадут.



После добавления панели инструментов надо создать. Для этого в классе **CMainFrame** необходимо объявить дополнительные переменные для каждой панели.

class CMainFrame : public CMDIFrameWnd

{

........

protected: // control bar embedded members

CStatusBar m\_wndStatusBar;

CToolBar m\_wndToolBar;

CToolBar m\_wndToolBarView;

CToolBar m\_wndToolBarTree;

......

};

Ну и сам код в **OnCreate()**:

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)

{

if (CMDIFrameWnd::OnCreate(lpCreateStruct) == -1)

return -1;

if (!m\_wndToolBar.CreateEx(this, TBSTYLE\_FLAT, WS\_CHILD | WS\_VISIBLE | CBRS\_TOP

| CBRS\_GRIPPER | CBRS\_TOOLTIPS | CBRS\_FLYBY | CBRS\_SIZE\_DYNAMIC) ||

!m\_wndToolBar.LoadToolBar(IDR\_MAINFRAME))

{

TRACE0("Failed to create toolbar\n");

return -1; // fail to create

}

if (!m\_wndToolBarTree.CreateEx(this, TBSTYLE\_FLAT, WS\_CHILD | WS\_VISIBLE | CBRS\_TOP

| CBRS\_GRIPPER | CBRS\_TOOLTIPS | CBRS\_FLYBY | CBRS\_SIZE\_DYNAMIC) ||

!m\_wndToolBarTree.LoadToolBar(IDR\_TOOLBARTREE))

{

TRACE0("Failed to create toolbar\n");

return -1; // fail to create

}

if (!m\_wndToolBarView.CreateEx(this, TBSTYLE\_FLAT, WS\_CHILD | WS\_VISIBLE | CBRS\_TOP

| CBRS\_GRIPPER | CBRS\_TOOLTIPS | CBRS\_FLYBY | CBRS\_SIZE\_DYNAMIC) ||

!m\_wndToolBarView.LoadToolBar(IDR\_TOOLBARVIEW))

{

TRACE0("Failed to create toolbar\n");

return -1; // fail to create

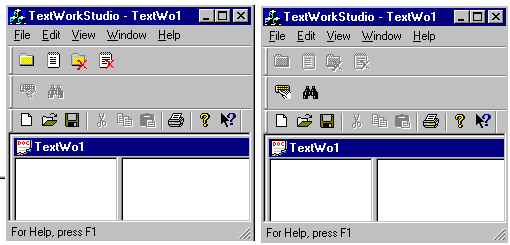
}

........

}

Ну и теперь самое главное. Для каждой кнопки нужно указать в каком виде будет производиться обработка события её нажатия. Сделать это можно используя **ClassWizard**. Свяжите кнопки для панели **IDR\_TOOLBARTREE** c функциями класса **CLeftView**, у нас их четыре. А кнопки панели **IDR\_TOOLBARVIEW** с Вашим классом вида у меня он называеться **CTextWorkStudioView**.

Теперь приложение можно запустить. И вот тут-то и можно посмотреть на что я хочу обратить внимание. То есть На панели инструментов кнопки будут активны, если активен тот вид, в котором производится обработка. Давайте убедимся. Запустите приложение и щелкните на правом виде.



А потом на левом. В зависимости от выбраного вида будут активизироваться панели инструментов. Вот и вся идея.

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?166) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?168) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc4.html)  
Автор **Каев Артем**.

## Шаг 171 - Диалог - обновляем элементы интерфейса

Сегодня я предлагаю Вам научиться (почти) автоматически управлять состояние элементов интерфейса в нашем диалоге. Для нашего случая такими элементами являются пункты меню и "контролы", т.е. элементы управления. Чтобы упростить задачу и не отвлекаться на частности, используем проект из шага **"Диалог - добавим акселераторы..."**.

Поставим себе такую задачу:

1. Кнопка **OK** должна быть активной только в том случае, если поле редактирования (далее - редактор) не пусто.
2. Пункт меню **Help/About** после первого выбора помечается галочкой, после последующих - становится неактивным.

Осталось немного - выполнить поставленную задачу. В принципе, для ее выполнения вовсе не обязательно знание MFC, достаточно иметь начальные знания о Win32 и вперед... Но MFC упрощает нашу работу, правда, обычно платой за простоту является размер исполняемого кода и его производительность.

В MFC для обновления элементов пользовательского интерфейса имеется специальный предназниченный для этого класс **CCmdUI** и обработчики **ON\_UPDATE\_COMMAND\_UI**. Если не слишком углубляться в мрачные недра MFC, то алгоритм для добавления "самообновляющихся" свойств элементам управления несложен: нужно для каждого элемента определить функцию-обработчик со следующим прототипом:

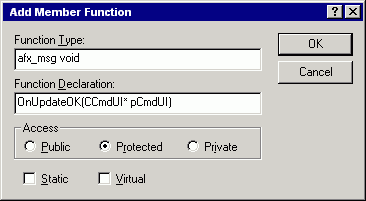
afx\_msg void OnUpdateConrol(CCmdUI\* pCmdUI);

Имя функции не имеет никакого значения, главное, чтобы совпадали тип функции и тип ее единственного аргумента. После этого необходимо добавить в карту сообщений окна (в нашем случае - диалога) макросы со ссылками на обработчики.

ON\_UPDATE\_COMMAND\_UI(ID\_CONTROL, OnUpdateControl)

Наверное, достаточно теории, переходим к практике.

Начнем с первого пункта. Что там у нас? Сделать **OK** неактивным (или, как говорит один мой знакомый "засерить", от слова "серый" :-) при пустом редакторе? Нет проблем... Грузим в Visual Studio наш простенький проект, закладка "ClassView", правый тык на классе **CAdvDlgDlg**, в контекстном меню выбираем "Add member function...".



*Почему класс доступа "Protected"? Как я уже говорил на одном в прошлых шагов, ни к чему давать слишком много свободы...*  
Затем нажатием кнопки **OK** добавляем эту самую функцию.

Определение функции смотрите ниже:

void CAdvDlgDlg::OnUpdateOK(CCmdUI \*pCmdUI)

{

BOOL bEnable = GetDlgItem(IDC\_EDIT)->GetWindowTextLength() != 0;

pCmdUI->Enable( bEnable );

}

Как видите, ничего сложного: получаем указатель на объект окна редактора, спрашиваем у редактора длину введенного текста, если длина ненулевая - делаем кнопку **OK** активной.

Если Вы руками работаете быстрее, чем читаете, и уже успели понажимать заветные комбинации клавиш **F7, Ctrl+F5**, то можете убедиться, что пока что ничего не изменилось. Так и должно быть, ведь надо внести изменения в карту сообщений. Вот так она выглядит у меня (изменения выделены):

BEGIN\_MESSAGE\_MAP(CAdvDlgDlg, CDialog)

//{{AFX\_MSG\_MAP(CAdvDlgDlg)

ON\_WM\_SYSCOMMAND()

ON\_WM\_PAINT()

ON\_WM\_QUERYDRAGICON()

ON\_COMMAND(ID\_ABOUT, OnAbout)

//}}AFX\_MSG\_MAP

**ON\_UPDATE\_COMMAND\_UI(IDOK, OnUpdateOK)**

END\_MESSAGE\_MAP()

Ну что, нетерпеливые, опять не работает? Ну тогда изменим еще одну функцию, уже знакомую нам по прошлому шагу...

BOOL CAdvDlgDlg::PreTranslateMessage(MSG\* pMsg)

{

// TODO: Add your specialized code here and/or call the base class

if ( TranslateAccelerator( m\_hWnd, hAccel, pMsg ) ) return TRUE;

**UpdateDialogControls(this, TRUE);**

return CDialog::PreTranslateMessage(pMsg);

}

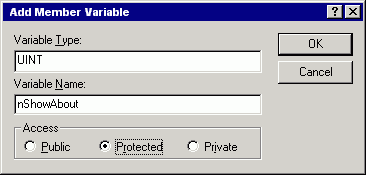
Скажу пару слов про **UpdateDialogControls**. Именно она для всех элементов интерфейса вызывает их обработчики, у нас пока что только **OnUpdateOK**. Первым аргументом в функцию **UpdateDialogControls** передается указатель на оконный объект, элементы которого будут обрабатываться, второй же по своему действию весьма необычен: **TRUE** приводит к тому, что элементы управления, не имеющие обработчиков **ON\_COMMAND**, принудительно будут "засериваться", FALSE - никакой принудиловки не будет.

Нетерпеливые, как Вы там? Заработало? Поздравляю. Правда, нетерпеливые, но внимательные могут заметить, что при запуске программы возможна ситуация, когда редактор пуст, но кнопка **OK** активна. Причина проста - обновление интерфейса происходит при поступлении в окно сообщений, а их после запуска еще не поступало. Стоит при этом провести мышкой по окну, и первое же сообщение **WM\_MOUSEMOVE** наведет порядок во внешнем виде программы. Чтобы избежать этой неоднозначности, можно пойти двумя путями: в функции **CAdvDlgDlg::OnInitDialog** вызвать еще раз **UpdateDialogControls** или послать самому себе какое-нибудь сообщение. Я выбрал второй вариант и добавил в конец **CAdvDlgDlg::OnInitDialog** одну строчку:

PostMessage(WM\_USER, 0, 0);

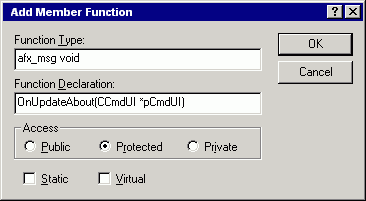
Вот теперь с первым пунктом поставленной задачи мы справились полностью. Кстати, обратите внимание на поведение кнопки 171_2.gif (152 b) ("Browse",идентификатор IDC\_BROWSE), причины которого в последнем параметре **TRUE** функции **UpdateDialogControls**.

Переходим ко второй части задачи. Тут нам, как ни крути, потребуется некий счетчик. Делаем счетчик, добавляя переменную - член (гусары, молчать!:-) класса **CAdvDlgDlg**.



Чтобы не тянуть резину, далее вкратце описываю свои действия: в конструкторе диалога инициализирую наш счетчик нулевым значением, после показа диалога **About** инкрементирую счетчик.

Теперь запускаем Class Wizard, в списке идентификаторов объектов класса **CAdvDlgDlg** находим ID\_ABOUT, дважды кликаем на сообщении **UPDATE\_COMMAND\_UI**, тем самым соглашаясь на создание обработчика.



Текст обработчика я написал такой, хотя можно было бы сделать его более эффективным:

void CAdvDlgDlg::OnUpdateAbout(CCmdUI \*pCmdUI)

{

switch ( nShowAbout )

{

case 0:

pCmdUI->Enable();

break;

case 1:

pCmdUI->SetCheck( 1 );

break;

case 2:

pCmdUI->SetCheck( 0 );

pCmdUI->Enable( FALSE );

break;

default:

break;

}

}

На этом подготовительные работы закончены, впереди самое интересное...

Интересность в том, что ограниченность диалога вновь не позволяет нам "автоматом" управлять состоянием элементов меню. Чтобы обойти это ограничение, надо добавить обработчик сообщения **WM\_INITMENUPOPUP**. Но это тоже не просто, поскольку Class Wizard для диалога не предлагает такого обработчика, но мы обманем волшебника и сделаем по-своему. Запускаем Wizard (Ctrl+W), переключаемся на закладку "Class Info", в списке "Class name" выбираем **CAdvDlgDlg**, а вот в списке "Message filter" вместо **Dialog** ставим **Topmost Frame**, тем самым существенно расширяя список обработчиков сообщений в списке волшебника. Теперь уже спокойно добавляйте нужный нам обработчик и, если хотите, возвращайте фильтр сообщений в прежнее состояние. Идея данного обработчика заимствована (с некоторой модификацией) у [John Wismar](mailto:john.wismar@iname.com), оригинал статьи которого Вы можете прочитать здесь: ["Handling OnUpdate() processing for menu items"](http://www.codeguru.com/dialog/OnUpdate.shtml). А его текст смотрите ниже:

void CAdvDlgDlg::OnInitMenuPopup(CMenu\* pPopupMenu, UINT nIndex, BOOL bSysMenu)

{

CDialog::OnInitMenuPopup(pPopupMenu, nIndex, bSysMenu);

if (!bSysMenu)

{

CCmdUI cmdUI;

cmdUI.m\_nIndexMax = pPopupMenu->GetMenuItemCount();

for (UINT i = 0; i < cmdUI.m\_nIndexMax; i++)

{

cmdUI.m\_nIndex = i;

cmdUI.m\_nID = pPopupMenu->GetMenuItemID(i);

cmdUI.m\_pMenu = pPopupMenu;

cmdUI.DoUpdate(this, TRUE);

}

}

}

В двух словах - обработчик при открывании меню перебирает его пункты и вызывает для каждого пункта функцию обновления. Собственно, на этом все. Единственное, на что хочется обратить Ваше внимание - нажатие кнопки-акселератора **F12** полностью эквивалентно выбору соответствующего пункта меню. Запускайте, проверяйте, пишите о замеченных ошибках...

Шаг прислал Галицкий Игорь ([ig@ntvplus.ru](mailto:ig@ntvplus.ru))

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/171.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?170) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?172) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc4.html)  
Автор **Каев Артем**.

## Шаг 173 - Многоликие иконы

Итак, Ваше приложение может использовать иконы. Иконы бывают следующих видов:

32х32 256 цветов

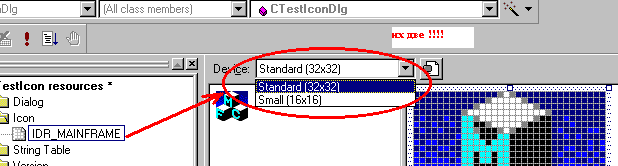
48х48 256 цветов

32х32 монохромный

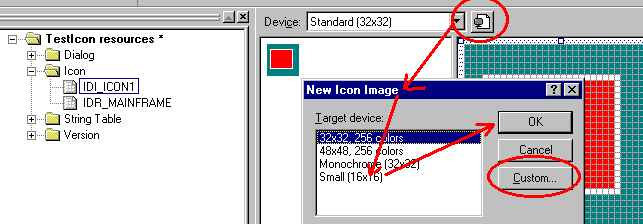
15х15 16 цветов

А ваше приложение может работать в совершенно ужастных условиях, например на черно белом мониторе, или на цветном мониторе, но в черно белом режиме. Кроме того пользователь может поставить совершенно возмутительные режимы типа 16 цветов или разрешение всякое там. Вообщем ужас. Издеваться будут как могут. Так вот икона должна быть всегда на высоте как секретарша в офисе, юбочка ниже колен, макияж и так далее. Выход из этой ситуации есть.

Создайте приложение **MFC AppWizard** на базе диалогового окна. Простое приложение без всяких настроек. В результате работы волшебшика в Вашем приложении будет обязательно создана икона **IDR\_MAINFRAME**, посмотрите на нее в ресурсах. Вам кажется, что она одна ??? Обман ! У меня, например, их две (как минимум это 16X16 и 32X32). Не верите ?? Посмотрите на картинку ниже, можете сами переключаться ради интереса. Под одним идентификатором 2 иконы !!!.



Вывод под одним идентификатором может скрываться нексколько икон. Раз это может волшебник, значит и мы сможем. Создайте свою икону. По умолчанию создастся икона 32Х32. Но мы можем к ней добавить любую, для этого достаточно нажать на кнопку и выбрать нужный режим. Смотрите.



Видите ??? Нажимаем на кнопку и появляются возможные варианты. Но кроме того есть кнопка **Custom**, которая позволит Вам сформировать икону любого типа. На Ваш вкус. Я еще раз напоминаю, что **Варианты одной иконы будут храниться под одним идентификатором**. Извлекаться они будут в зависимости от ситуации.

Вот пример из жизни. Есть иконы размером 32Х32 и 16Х16 под одним идентификатором. Мне надо поместить иконы 16х16 в список изображений. Я создаю список и помещаю туда иконы. И помещаются иконы именно 16Х16.

CLeftView::CLeftView()

{

cImageList.Create(16,16,FALSE,3,0);

InsertImageListIcon();

}

Создал список изображений. И помещаю туда иконы.

void CLeftView::InsertImageListIcon()

{

HICON hIcon;

hIcon=::LoadIcon(AfxGetResourceHandle(),MAKEINTRESOURCE(IDI\_ICON\_DIR));

cImageList.Add(hIcon);

hIcon=::LoadIcon(AfxGetResourceHandle(),MAKEINTRESOURCE(IDI\_ICON\_DOC));

cImageList.Add(hIcon);

hIcon=::LoadIcon(AfxGetResourceHandle(),MAKEINTRESOURCE(IDI\_ICON\_DIR\_OPEN));

cImageList.Add(hIcon);

hIcon=::LoadIcon(AfxGetResourceHandle(),MAKEINTRESOURCE(IDI\_ICON\_DOC\_OPEN));

cImageList.Add(hIcon);

}

В список изображений поместятся иконы именно 16х16. Не верите, в [следующем шаге](http://www.firststeps.ru/mfc/steps/r.php?174) поверите :-)

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?172) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?174) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc4.html)  
Автор **Каев Артем**.

## Шаг 174 - Добавление пунктов к дереву

Итак, продолжаем разговор о стиле **Exploler** и дереве в левом окне. Мы сейчас создадим код, который добавляет к дереву картинки. Все они будут на одном уровне, то есть мы будем создавать корни. Итак, нам нужно наше приложение, которое мы начали делать в ["Шаг 167 - Стиль окна Exploler и панели инструментов"](http://www.firststeps.ru/mfc/steps/r.php?167).

Для того, чтобы в дереве можно было использовать картинки необходимо подготовить класс списка изображений. Для этого объявляем переменную класса списка изображений.

class CLeftView : public CTreeView

{

...

CImageList cImageList;

...

};

Создадим и настроим в конструкторе список изображений.

CLeftView::CLeftView()

{

cImageList.Create(16,16,FALSE,3,0);

InsertImageListIcon();

}

void CLeftView::InsertImageListIcon()

{

HICON hIcon;

hIcon=::LoadIcon(AfxGetResourceHandle(),MAKEINTRESOURCE(IDI\_ICON\_DIR));

cImageList.Add(hIcon);

hIcon=::LoadIcon(AfxGetResourceHandle(),MAKEINTRESOURCE(IDI\_ICON\_DOC));

cImageList.Add(hIcon);

hIcon=::LoadIcon(AfxGetResourceHandle(),MAKEINTRESOURCE(IDI\_ICON\_DIR\_OPEN));

cImageList.Add(hIcon);

hIcon=::LoadIcon(AfxGetResourceHandle(),MAKEINTRESOURCE(IDI\_ICON\_DOC\_OPEN));

cImageList.Add(hIcon);

}

Обратите внимание на то, что я создал массив икон 16х16, так как большие иконы мне показались не очень красиво. Для этого я создал 4 иконы 16х16. Не зря был написан [прошлый шаг](http://www.firststeps.ru/mfc/steps/r.php?173).

Но вот внимание !!! Теперь для дальнейшего использования необходимо связать список изображений с классом дерева. Вот так.

void CLeftView::OnInitialUpdate()

{

.......

GetTreeCtrl().SetImageList(&cImageList,TVSIL\_NORMAL);

}

Вызываем соотвествующую функцию, передаем указатель на класс и опцию. Только функция эта не класса вида а класса **CTRL** соответствующего элементу управления дерево.

Теперь все готово к добавлению элемента к дереву. Вот код.

////////////////////////////////////////////////////

// CLeftView message handlers

void CLeftView::OnAddDir()

{

GetTreeCtrl().InsertItem("Dir",0,2);

}

Как видите опять функция класса **CTRL** позволяет просто вставить пункт. Обратите внимание на цифры. Эти цифры указывают какое изображение будет выводиться в случае, если пункт активный или неактивный.

Итак в глобальном плане. Готовим список изображений, связываем его с деревом, вставляем пункты указывая номера изображений активных и неактивных элементов.

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?173) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?175) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc4.html)  
Автор **Каев Артем**.

## Шаг 175 - Удаление пунктов из дерева и изменение имени

В [прошлом шаге](http://www.firststeps.ru/mfc/steps/r.php?174) мы научились добавлять пункты верхнего уровня к дереву. Теперь нам необходимо научиться их удалять. На панели инструментов у нас есть кнопка удаления, к которой привязана функция реализованная в виде дерева.

175_1.gif (1035 b)

А вот и код для неё в классе вида на основе дерева, но пока пустой.

void CLeftView::OnDeleteDir()

{

// TODO: Add your command handler code here

}

Первое, что нам надо сделать это определить был ли выделен элемент дерева. Для этого надо воспользоваться методом **GetSelectedItem**. Этот метод возвращает указатель на структуру типа **HTREEITEM** или **NULL**, если не было выделения элемента.

void CLeftView::OnDeleteDir()

{

HTREEITEM hi;

hi=GetTreeCtrl().GetSelectedItem();

if (hi!=NULL)

{

}

else AfxMessageBox("Not Selected Item");

}

Если элемент получен, то есть ссылка на **HTREEITEM**, теперь этот элемент можно и удалить.

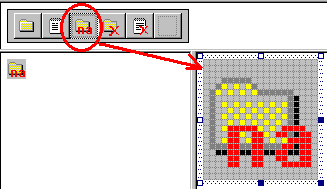
if (hi!=NULL)

{

GetTreeCtrl().DeleteItem(hi);

}

Для переименования пункта нам надо сделать новую кнопку на панели инструментов и связать обработку с видом.



Для начала проверим выделен ли пункт меню.

void CLeftView::OnRanameDir()

{

HTREEITEM hi;

hi=GetTreeCtrl().GetSelectedItem();

if (hi!=NULL)

{

}

else AfxMessageBox("Not Selected Item");

}

Теперь можно получить название элемента дерева. Для этого есть метод **GetItemText**. Вот, например, так.

if (hi!=NULL)

{

AfxMessageBox(GetTreeCtrl().GetItemText(hi));

}

Для установки нового текста есть метод **SetItemText**, вот так им можно воспользоваться.

if (hi!=NULL)

{

AfxMessageBox(GetTreeCtrl().GetItemText(hi));

GetTreeCtrl().SetItemText(hi,"new");

}

Теперь дело за маленьким. Сделать диалоговое окно, которое позволит править текст. И переименование элемента дерева будет готово. Вот общий код.

if (hi!=NULL)

{

CDIalogEdit cd(GetTreeCtrl().GetItemText(hi));

if (cd.DoModal()==IDOK)

{

GetTreeCtrl().SetItemText(hi,cd.m\_Edit);

}

}

Единственное, что мне пришлось создать класс диалога. Дальше на него я поместил строку редактирования и создал в классе диалога элемент данных **m\_Edit**, перегрузил конструктор, чтобы в окне редактирования сразу появлялось то, что и в пункте элемента дерева. Вот конструктор.

CDIalogEdit::CDIalogEdit(CString s,CWnd\* pParent /\*=NULL\*/)

: CDialog(CDIalogEdit::IDD, pParent)

{

//{{AFX\_DATA\_INIT(CDIalogEdit)

m\_Edit = s;

//}}AFX\_DATA\_INIT

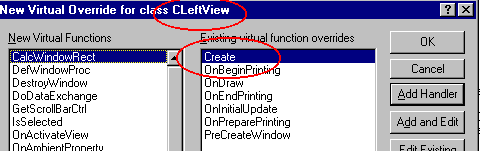
}

Теперь наше приложение умеет удалять пункты меню и их переименовывать.

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?174) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?176) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc4.html)  
Автор **Каев Артем**.

## Шаг 176 - Просмотр связей и подчиненность элемента

Наша задача в этом шаге сделать просмотр дерева так, чтобы были видны связи и нижний элемент можно было переподчинить верхнему того-же уровня. Давайте рассмотрим как Вы можете управлять просмотром дерева. Для этого нам необходимо перегрузить виртуальную функцию **Create** для просмотра.



После этого мы можем с Вами добавить стили вида. Вот как в примере ниже. Мы с переменной **dwStyle** добавляем новые атрибуты управляющие видом дерева.

BOOL CLeftView::Create(LPCTSTR lpszClassName, LPCTSTR lpszWindowName,

DWORD dwStyle, const RECT& rect, CWnd\* pParentWnd, UINT nID,

CCreateContext\* pContext)

{

dwStyle=dwStyle | TVS\_HASLINES | TVS\_LINESATROOT | TVS\_HASBUTTONS;

return CWnd::Create(lpszClassName, lpszWindowName,

dwStyle, rect, pParentWnd, nID, pContext);

}

Следующей нашей задачей будет переподчинение элемента верхнему элементу при условии, что они одного уровня. На панели инструментов я создал кнопку.

176_2.gif (975 b)

Ну давайте посмотрим последовательность действий для подчинения нижнего элемента верхнему. Итак, первое - это проверка на то, что элемент дерева выделен.

void CLeftView::OnGoRigth()

{

HTREEITEM hi;

hi=GetTreeCtrl().GetSelectedItem();

if (hi!=NULL)

{

.......

}

else AfxMessageBox("Not Selected Item");

}

Теперь нам надо получить указатель на элемент дерева того же уровня. Вот так это можно сделать. Функция **GetPrevSiblingItem** получает указатель на элемент того же уровня, но выше указанного элемента.

HTREEITEM his;

his=GetTreeCtrl().GetPrevSiblingItem(hi);

if (his!=NULL)

{

......

}

else AfxMessageBox("Not Move Item")

Ну вот и пришло время подчинить нижний элемент верхнему.

int notselect;

int select;

GetTreeCtrl().GetItemImage(hi,notselect,select);

GetTreeCtrl().InsertItem(GetTreeCtrl().GetItemText(hi),notselect,select,his);

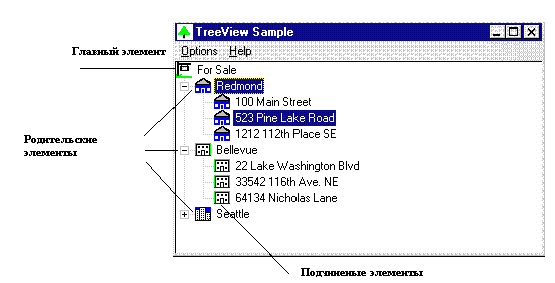
GetTreeCtrl().DeleteItem(hi);

Вот так можно сделать. Смотрите. Первое, что мы делаем это получаем 2 характеристики выделенного элемента. Первые две это картинки когда активный или когда не активный. Командой **InsertItem** мы вставляем элемент как подчиненный для верхнего. Не зря же мы указали **his** в конце. Это указатель на элемент, в подчинение которому пойдет этот элемент. В конце перемещаемый элемент мы удаляем.

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?175) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?177) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc4.html)  
Автор **Каев Артем**.

## Шаг 177 - Общие выводы о работе с деревом

Итак, у дерева есть несколько элементов. Можно считать, что есть корневые элементы, есть ветвления и есть подчиненные элементы. Общее представление можно получить посмотрев картинку внизу.



Важное место при работе с деревом занимает указатель **HTREEITEM**, который указывает на элемент дерева. Именно этот указатель возвращается при многих операциях с деревом, например, получение текущего элемента, получение элемента выше или подчиненных элементов. Имея этот указатель можно получить доступ ко многим операциям над элементами и свойствам элемента. Практически все функции класса **CTreeCtrl** используют этот указатель в качестве параметров. Но это только указатель.

У дерева есть стили, с помощью которых можно управлять аспектами его отображения. Описание каждого стиля можно посмотреть в [подробностях](http://www.firststeps.ru/mfc/detail/38.html).

TVS\_HASLINES

TVS\_LINESATROOT

TVS\_HASBUTTONS

TVS\_EDITLABELS

TVS\_SHOWSELALWAYS

Стили устанавливаются при создании окна. Для этого нужно перегрузить функцию **Create** вида наследника от **CTreeView**. В последствии стили можно изменить используя функции **GetWindowLong** и **SetWindowLong**.

Для добавления пункта необходимо послать сообщение **TVM\_INSERTITEM**, это сообщение вернет указатель **HTREEITEM** однозначно определяющий пункт дерева. К счастью **MFC** скрывает от нас подобные подробности. Давайте посмотрим как это делается. Для вставки элемента мы должны послать сообщение и указать в параметрах структуру **TVINSERTSTRUCT**, которая содержит структуру **TVITEM**. Сложно ??? Класс **CTreeCtrl** скрывает эти подробности позволяя вместо посылки сообщений вызывать функции. Например, вот так это выглядит.

GetTreeCtrl().InsertItem(GetTreeCtrl().GetItemText(hi),notselect,select,hit,0);

В данном коде никаких сообщений посылать не надо.

Внимание !!! Работать с элементами управления **Windows** нужно на основе сообщений. Классы **....CTRL** заменяют посылку сообщений вызывами функций классов. Это значительно упрощает программирование. Но вероятно и ведет к усечению ряда возможностей. Еще раз посылка сообщений заменена вызовом функции. Это очень важно для понимания работы многих элементов управления. Вот так реализована подмена сообщения.

\_AFXCMN\_INLINE BOOL CTreeCtrl::GetItem(TVITEM\* pItem) const

{ ASSERT(::IsWindow(m\_hWnd)); return (BOOL)::SendMessage(m\_hWnd, TVM\_GETITEM, 0, (LPARAM)pItem); }

С этим понятно. Теперь надо разобраться, как получить информацию от элемента дерева. Если без **MFC**, то нужно послать сообщение, а с **MFC** вызвать функцию **GetItem**, что одно и тоже. Вот она.

BOOL GetItem( TVITEM\* pItem );

В эту функцию передается адрес структуры **TVITEM**, вот её описание.

typedef struct tagTVITEM

{

UINT mask;

HTREEITEM hItem;

UINT state;

UINT stateMask;

LPTSTR pszText;

int cchTextMax;

int iImage;

int iSelectedImage;

int cChildren;

LPARAM lParam;

} TVITEM, FAR \*LPTVITEM;

Что здесь главное, ну во-первых, **hItem** должен указывать на элемент дерева, который существует. Дальше **mask**, а вот здесь указывается какие данные мы хотим извлечь.

Обратите внимание на то, что название это **LPTSTR**. То есть нужен буфер, в который она поместится. Вот пример кода извлекающего имя активного элемента.

void CLeftView::OnTest()

{

HTREEITEM hi;

hi=GetTreeCtrl().GetSelectedItem();

if (hi!=NULL)

{

TV\_ITEM pItem; // структура

char szBuffer[20]; // буфер для названия

memset(&pItem,0,sizeof(pItem)); // обнуляем структуру

pItem.hItem=hi; // этого элемента информацию мы хотим получить

pItem.mask= TVIF\_TEXT ; // тип получаемой информации

pItem.pszText = szBuffer; // куда вернуть

pItem.cchTextMax=19; // размер буфера = размер char-1 для завершения строки

GetTreeCtrl().GetItem(&pItem); // получить информацию

AfxMessageBox(pItem.pszText);

}

else AfxMessageBox("Not Selected Item");

}

Итак, воспользовавшись этим можно получить всю информацию о элементе дерева и перевести указатель **HTREEITEM** в реальную структуру **TV\_ITEM**.

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?176) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?178) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc4.html)  
Автор **Каев Артем**.

## Шаг 179 - Немного о классе CString и LPCTSTR

Привет Артем.

Ответь пожалуйста, в письме, как используется

этот чертов оператор LPCTSTR класса CString.

Ну ОЧЕНЬ надо.

С уважением Дмитрий.

Класс **CString** появился как альтернативный метод работы со строками типа **char buffer[100]**. При этом при его создании учитывалось, что многие функции уже созданы и необходимо будет обеспечить поддержку старого подхода основанного на указателях строк. Рассматривая данный указатель давайте посмотрим историю его появления.

typedef LPCSTR LPCTSTR;

typedef CONST CHAR \*LPCSTR, \*PCSTR;

Вот это просто указатель на строку. Только если Вы обратите внимание на буковку **C** в строке **LPCTSTR** этот указатель именно **CONST**. В **MSDN** описано, что **LPCTSTR** это 32 битный указатель на строку для совместимости с **UNICODE** и **DBCS**. **UNICODE** строки отличаются от обычных строк вот чем:

wchar\_t - 16 бит

char - 8 бит

Проще всего воспринимать этот указатель как аналог указателя в **Win32**

LPCSTR (Win32 data type) - LPCTSTR

32 битный указатель на строку. Использовать его можно также как и используется указатель на буфер. Вот пример кода.

char buffer[100];

strcpy(buffer,"Hello World!\n");

LPCTSTR st=buffer;

CString s(st);

AfxMessageBox(s);

Кроме того в классе **CString** есть оператор **LPCTSTR**

operator LPCTSTR ( ) const;

Этот оператор позволяет получить доступ к строке в стиле **C**. Возвращается только указатель. Вот пример кода

CString s;

LPCTSTR lp;

s="Hi";

lp = (LPCTSTR) s;

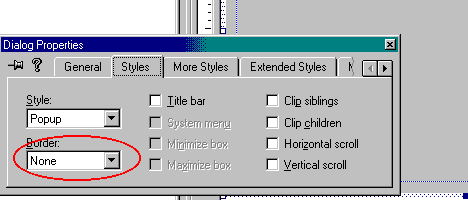
printf(lp);

return 0;

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?178) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?180) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc4.html)  
Автор **Каев Артем**.

## Шаг 184 - Создаем хранитель экрана

Давате сделаем основу для чего-нибудь полезного, например, хранителя экрана. Создаем проект с именем **SaveScreen** на базе диалогового окна. При использовании **AppWizard** отключайте все, что можно. На диалоговом окне удалите все кнопки. При этом диалоговое окно нужно сделать без рамки указав соответствующий стиль.



Теперь нам нужно изменить функцию **InitInstance()**:

BOOL CSaveScreenApp::InitInstance()

{

if (!strcmpi(m\_lpCmdLine,"/s") ||

!strcmpi(m\_lpCmdLine,"-s") ||

!strcmpi(m\_lpCmdLine,"s"))

{

SetDialogBkColor(RGB(0,0,0));

CSaveScreenDlg dlg;

m\_pMainWnd = &dlg;

dlg.DoModal();

}

return FALSE;

}

Итак, функцией SetDialogBkColor устанавливаем цвет фона диалога. А вот изначально проверяем параметры командной строки. Для хранителей экрана есть некоторые правила в том числе и на командные параметры.

c для конфигурирования из конрольной панели

s для запуска в режиме сохранения

Это все хорошо, но диалоговую панель необходимо сделать на весь экран. Это можно сделать в **OnInitDialog**.

BOOL CSaveScreenDlg::OnInitDialog()

{

CDialog::OnInitDialog();

SetIcon(m\_hIcon, TRUE);

SetIcon(m\_hIcon, FALSE);

SetWindowPos(&wndTopMost,0,0,::GetSystemMetrics(SM\_CXSCREEN),

::GetSystemMetrics(SM\_CYSCREEN) ,SWP\_SHOWWINDOW);

return TRUE;

}

Самое главное здесь в функции **SetWindowPos**, которая устанавливает размеры экрана получая их из системы используя **GetSystemMetrics**.

Для завершения работы необходимо при нажатии на клавишу или мышку. Ну это с помощью **ClassWizard** код вот такой:

void CSaveScreenDlg::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)

{

CDialog::EndDialog(IDOK);

}

void CSaveScreenDlg::OnLButtonDown(UINT nFlags, CPoint point)

{

CDialog::EndDialog(IDOK);

}

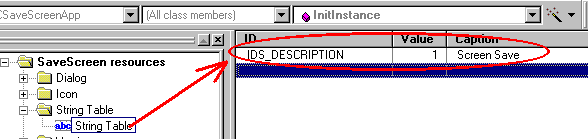
void CSaveScreenDlg::OnRButtonDown(UINT nFlags, CPoint point)

{

CDialog::EndDialog(IDOK);

}

Для того, чтобы название хранителя экрана было в контрольной панели нужно создать строковый ресурс вот такой.



Главное, что значение строкового ресурса равно 1.

И еще одна неприятность, дело в том, что хранитель экрана будет каждый раз запускаться при истечении заданного промежутка времени. Поэтому нам придется эту ситуацию тоже предусмотреть.

Поместить хранитель экрана надо в папку **windows\system** и поменять расширение на **\*.SCR**. По этому расширению **Windows** определяет, что это хранитель экрана.

Выходов из этой ситуации наверно много, но вот я решил просто установить у окна **title** и по нему искать окно если она загружено, то запускать приложение нельзя. Выход конечно не очень хороший, вот он:

if (FindWindow(NULL,"SCREENSAVE")==NULL) // title dialog тоже

{

SetDialogBkColor(RGB(0,0,0));

CSaveScreenDlg dlg;

m\_pMainWnd = &dlg;

dlg.DoModal();

}

Кто чего знает по этому поводу пишите не стесняйтесь.

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?183) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?185) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc4.html)  
Автор **Каев Артем**.

## Шаг 185 - Изобразительные средства CDC

В контексте устройства есть ряд изобразительных средств, которые являются наследниками **CGdiObject**. В **MFC** их шесть. Вот смотрите.

CObject - |-CBitmap

| |-CBrush

CGdiObject ----|-CFont

|-CPallete

|-CPen

|-CRgn

Класс **CPen** аналог дескриптора пера **Win 32 HPEN**, и вообще это глобально каждому классу **MFC** из ряда изобразительных средств есть аналог дескриптор в **Win 32**.

CBitmap HBITMAP

CBrush HBRUSH

CFont HFONT

CPalette HPALETTE

CPen HPEN

CRgn HRGN

Теперь кратко о каждом классе **CBitmap** - это класс, который умеет работать с растровыми изображениями. **CBrush** - это кисть для рисования. **CFond** - класс шрифта текста. **CPallete** - класс умеющий работать с цветовыми палитрами. **CPen** - класс пера и **CRgn** - класс региона, то есть области вывода.

Для того, чтобы этот объект использовать его необходимо создать. Так вот создание объекта связано со связыванием объекта программы с объектом **Windows**. Общий механизм такой: использование объекта состоит из двух шагов

* создание
* сваязывание

Создается объект в момент его инициализации, а вот связывание производится при вызове функции **Create**, параметры которой зависят конкретно от данного класса. При завершении работы должно быть гарантированное удаление объекта, так как в системе количество объектов хоть и большое, но ограниченное.

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?184) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?186) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc4.html)  
Автор **Каев Артем**.

## Шаг 186 - Доступ к элементам управления в диалоговом окне

----- Original Message -----

From: Trionon, Kharkov

To: kaev@yandex.ru

Sent: Saturday, May 06, 2000 2:37 AM

Subject: MFCByStep

Я предпочитаю везде использовать диалоги и никогда не

создавать окон с помощью CreateWindow...

Вот тут я узнал один интересный логический ход при работе с диалоговыми окнами, и думаю с Вами им нужно поделиться. Отчасти к небольшому развитию этой темы меня подвело и это письмо выше. Спасибо за мысль. И так задача в том можно ли как-нибудь еще кроме как через **DDX** и класс элемента получить доступ к элементу на диалоговом окне. Можно имея идентифкатор элемента.

Все элементы управления являются потомками **CWnd** и можно очень просто построить этот класс имея идентифкатор элемента. Вот так с помошью **GetDlgItem**.

CWnd\* pWnd = GetDlgItem(IDC\_....);

Дальше мы можем использовать функции **CWnd**

pWnd->GetWindowRect(...);

Несмотря на всю простоту этого примера он кроет огромный потенциал для автоматизации. Все элементы диалогового окна являются потомками от **CWnd**, в нем находится огромное количество параметров окна, в том числе и местоположение, размер и так далее. Вот. Используя это свойство Вы сможете проводить изменение положения или размера или еще чего-то сразу со всеми элементами диалогового окна вне зависимости от их типа только на основе идентификатора ресурса.

3. Ваш пример по вышеуказанному адресу с сохранением функциональности

мог бы занимать значительно меньше - практически это мог бы быть

единственный вызов DialogBox() + функция диалога Конечно в этом случае

он бы потерял свойство "учебности", но если бы все вместе с "классическим"

примером, демонстрирующим СТРУКТУРУ и ИДЕОЛОГИЮ Win32 публиковали бы

"короткий", сколько людей было бы наставлено на путь истинный!

Действительно самый короткий пример с использованием **MFC** вот такой. Делаем проект **Win 32 Application**. Ставим в настройках **USE MFC** далее код.

#include "stdafx.h"

#include "afxwin.h"

int APIENTRY WinMain(HINSTANCE hInstance,

HINSTANCE hPrevInstance,

LPSTR lpCmdLine,

int nCmdShow)

{

AfxMessageBox("Hello Word");

return 0;

}

Только надо в **stdafx.win** отключить **#include**

......

#define WIN32\_LEAN\_AND\_MEAN // Exclude rarely-used stuff from Windows headers

//#include <windows.h>

// TODO: reference additional headers your program requires here

......

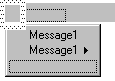
[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?185) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?187) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc4.html)  
Автор **Каев Артем**.

## Шаг 187- Всплывающее меню Popup

Попробуем создать меню, которое частенько появлется при нажатии правой кнопки мыши... Так называемое всплывающее меню.

В файле ресурсов необходимо создать меню. Для этого перейти на вкладку **Resource** и Меню **Insert**, далее **New Resource** из списка выберите меню. Присвоить ему имя **IDR\_MENUPOPUP**.

Вы уже должны знать как нужно создавать меню (см. ["Шаг 6 - Создание меню"](http://www.firststeps.ru/mfc/steps/r.php?6)). Но здесь надо обратить внимание на маленькую тонкость - создаем наше меню как подменю основного выпадающего меню, которое можно оставить пустым:



Возьмем реакцию на щелчок левой кнопкой мыши. И изменим процедуру реакции.

BOOL CMyDlg::OnInitDialog()

{

m\_PopupMenu.LoadMenu(IDR\_MENUPOPUP);

ASSERT(m\_PopupMenu!=NULL);

}

void CMyDlg::OnSelect()

{

CPoint point;

GetCursorPos(&point);

m\_PopupMenu.GetSubMenu(0)->TrackPopupMenu(

TPM\_LEFTALIGN|TPM\_LEFTBUTTON,

point.x, point.y, this);

}

Сейчас вам уже должны быть прозрачны почти все действия. Остановимся на функции **OnSelect()**, которая вызывается в подходящий для нас момент. Первоначально нам надо узнать, в каком месте показывать меню, для этого определяем объект **CPoint point** и берем текущее положение курсора **GetCursorPos(&point)**. Теперь все готово, чтобы показать на экране наше меню. Для этого вызываем функцию **m\_PopupMenu.GetSubMenu(0)**, которая возвращает указатель на наш пустой пункт меню, и тут же его применяем по назначению вызвав функцию **TrackPopupMenu**, первый параметр задает в некотором роде выравнивание, второй и третий - координаты появления, четвертый - родителя, необязательный пятый параметр определяется структурой **CRECT**, и задает область видимости (то есть куда можно ткнуть).

Шаг прислал [sadhan](mailto:sadhan@i.am?SUBJECT=MFCByStep).

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?186) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?188) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc4.html)  
Автор **Каев Артем**.

## Шаг 188 - Наборы символов

На данный момент используется несколько наборов символов. Самым старым и распространеным является набор **ASCII** - этот набор появился еще когда компьютеры были 8 разрядные, и с тех пор для отображения символов используется один байт. Байт состоит из 8 бит и поэтому в этом наборе всего 256 символов.

В операционной системе **Windows** есть еще два набора символов - это **MBCS** и **Unicode**. Последний набор реализован только в **Windows NT**, но не в **Windows 9x**.

Для работы с наборами символов в **MFC** есть специальный заголовочный файл, который реализует основные возможности. Имя у него **TCHAR.h**. Имеются команды препроцессора для работы с кодировками

\_MBCS

\_UNICODE

По умолчанию **MFC** использует **\_MBCS**. Вы можете посмотреть или заменить эту установку в **Project - Setting - C/C++ - Preprocessor header**. Для преобразования символов есть макрокоманды типа **\_TCHAR** или **\_T**. Но на этом все не заканчивается, можно для **UNICODE** переопределить и основную функцию **main** для полноценного использования широких символов.

wmain( int argc, wchar\_t \*argv[ ], wchar\_t \*envp[ ] )

Для работы со строками часто используеться класс **CString** этот класс работает, как с **Unicode** так и с **MBCS**. Кстати поддержка **UNICODE** не устанавливается при автоматической установке. Необходимо выбрать произвольную и указать установку **UNICODE**. Библиотеки **MFC** c поддержкой **UNICODE** отличаются от обычных **LIB** наличием буковки **U**. Например **MFC42U.lib**. Предусмотрены специальные функции для работы с **UNICODE**.

GetTextCharset

......

MultiByteToWideChar

.....

WideCharToMultiByte

Так же есть свои макросы и структуры. Для каждой страны в **UNICODE** предусмотрены смещения.

0 Basic Latin

1 Latin-1 Supplement

2 Latin Extended-A

3 Latin Extended-B

4 IPA Extensions

5 Spacing Modifier Letters

6 Combining Diacritical Marks

7 Basic Greek

......

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?187) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?189) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc4.html)  
Автор **Каев Артем**.

## Шаг 189 - Приложение на основе HTMLVIEW

Создаем приложение **MFC AppWizard** на основе одного окна **SDI** и в шаге 6 базовый класс **Base Class** надо выбрать **CHtmlView** и нажимаем **Finish**. Если Вы посмотрите код, то увидите в функции **CTestHTMLView::OnInitialUpdate** следующий код

void CTestHTMLView::OnInitialUpdate()

{

CHtmlView::OnInitialUpdate();

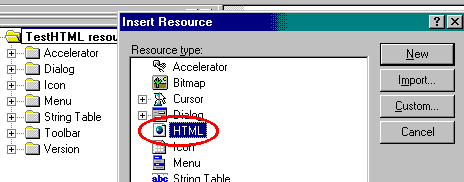
// TODO: This code navigates to a popular spot on the web.

// change the code to go where you'd like.

Navigate2(\_T("http://www.microsoft.com/visualc/"),NULL,NULL);

}

Этот код при запуске ищет страницу в **Internet**. Вы можете либо сами указать страницу в интернете либо создать локальную страницу в ресурсах, с которой и будет производиться старт системы. Эту страницу тоже можно разметить в ресурсах. Для этого перейдите на вкладку ресурсы и выберите **Insert Resource**.



В разделе ресурсов появится **HTML** раздел, в котором будет **HTML** документ и его идентификатор. Давайте создадим простой документ.

<html>

<p> Hello

</html>

Для загрузки в **HTMLVIEW** страницы из ресурсов есть специальная функция **LoadFromResource**. Вот она.

BOOL LoadFromResource( LPCTSTR lpszResource );

BOOL LoadFromResource( UINT nRes );

И реализуется это в той же функции **OnInitialUpdate**.

void CTestHTMLView::OnInitialUpdate()

{

CHtmlView::OnInitialUpdate();

LoadFromResource(IDR\_HTML1);

}

Теперь при старте приложения будет загружаться страница из ресурсов. Но при работе в реальной сети часто возникает ситуация, когда загрузка страницы производится долго и должен быть механизм, который позволит определить то, что загрузка документа закончена. Такая функция есть и она виртуальная **OnDocumentComplete**, если Вы ее перегрузите, то она будет выглядеть так.

void CTestHTMLView::OnDocumentComplete(LPCTSTR lpszURL)

{

// TODO: Add your specialized code here and/or call the base class

CHtmlView::OnDocumentComplete(lpszURL);

}

Сам класс реализует практически все функции по навигации такие как **GoBack**, **GoForward** и так далее. Вам нужно только привязать их к панели инструментов или пунктам меню.

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?188) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?190) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc4.html)  
Автор **Каев Артем**.

## Шаг 190 - WinInet

Для работы с Интернетом есть классы **WinInet - Windows Internet Externsion**. Общая идея этих классов упростить работу с интернетом не вдаваясь в подробности работы с сокетами и протоколом **TCP/IP**. Для работы с классами **WinInet** описание их нужно подключить

#include "afxinet.h"

Основой является класс **CInternetSession**, который используется для установки связи и инициализации соединения. Работа начинается с создания объекта сессии. Параметров у конструктора много, но самое интересное, что они могут быть все по умолчанию :-). Можно указать имя программы, которая производит выхоз как в примере ниже, но можно и этого не делать указав **NULL** тогда имя классов будет получено самостоятельно путем вызова **AfxGetAppName**.

CInternetSession se("My Session");

Следующим шагом будет создание объекта **CStdioFile**.

CStdioFile\* pFile = NULL;

При открытии сессии вернется указатель этого класса. Вернувшийся указатель нужно проверить на **NULL**. Вот как можно это сделать

CStdioFile\* pFile = NULL;

pFile = se.OpenURL("http://www.firststeps.ru/");

if (pFile != NULL)

{

}

А вот теперь можно и читать строки.

{

pFile->ReadString(cs);

pFile->Close();

}

В данном случае появится только одна первая строка **HTML**. При необходимости прочитать весь документ можно сделать так.

while(pFile->ReadString(cs))

{

........

}

После получения файла сесию нужно закрыть.

se.Close();

Вот такой он **WinInet**, но он может намного больше, например, перехватить функцию **OnStatusCallBack** для получения информации о ходе соединения. Это виртуальная функция член класса **CInternetSession**.

virtual void OnStatusCallback( DWORD dwContext,

DWORD dwInternetStatus, LPVOID lpvStatusInformation,

DWORD dwStatusInformationLength );

Так же класс **CInternetSession** умеет устанавливать специфические соединения. Обращения к таким соединениям производится идеологически, также создается файл и ему присваивается указатель соединения, но только для каждого типа соединения свой класс.

GetGopherConnection -> CGopherFileFind

GetHttpConnection -> CHttpFile

GetFtpConnection -> CFtpConnection

Функции связанные с работой **WinInet** находятся в **WININET.DLL**. То есть она должна быть в системе для корректной работы, хотя я сомневаюсь, что есть компьютер, на котором ее нет :-)

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?189) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?191) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc4.html)  
Автор **Каев Артем**.

## Шаг 191 - Кто сказал что в VC сложнее работать с БД

Это будет повтор стандартного создания приложения

принятого в примерах по VB и Delphi

с целью демонстрации той же простоты

Итак, создаем приложение на базе диалогового окна с именем **TestDB**. Не забудьте поставить **ActiveX** в разделе **Support** волшебника. Это для вставки **ActiveX** элементов.

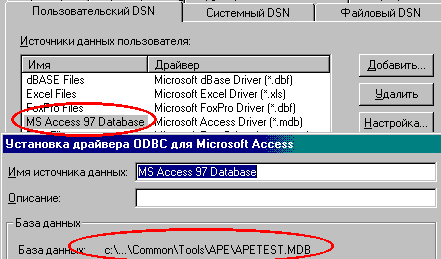
Классически для отображения таблицы в **GRID** необходимо два компонента

|-Источник таблиц

|-Источник строк

Отобразительное стредсво

Источник строк и таблиц обычно представлен в виде одного объекта. Первое, что надо сделать - это настроить доступ **ODBC** пользовательский для **ACCESS**. Писалось об этом много раз. Вот так настроен у меня.

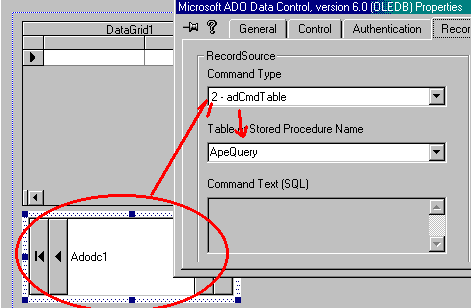


Теперь надо на диалоговое окно поместить элемент управления, который может получать данные. Давайте поставим **Microsoft ADO Data Control**. В нем необходимо настроить

На вкладке Control USE DATA SOURCE NAME - MS Access 97 Database

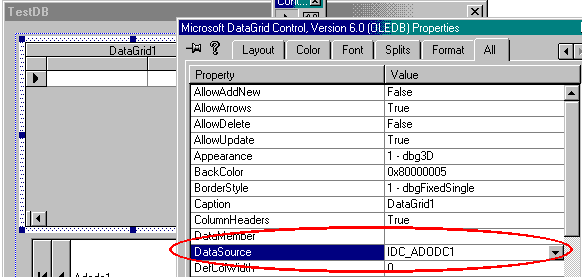
На вкладке General Visible - False

На вкладке Resord Source Command Type - adCmdTable И выбрать таблицу



Невидимым я его делаю потому что нечего на него смотреть :-)

Теперь дорогие мои пора вставить элемент управления сетка. Опять **Insert ActiveX** только теперь **Microsoft DataGrid Control**. И его надо настроить, то есть сослаться на источник данных.



Ну вот и все можно запрускать. У вас в сетке отобразится Ваша таблица. Ничего страшного нет. Ни одной строки кода. Если Вы закомы с **VB** или **Delphi**, то данный процесс должен быть Вам до боли знаком.

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?190) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?192) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc4.html)  
Автор **Каев Артем**.

## Шаг 193 - Попробуем DirectDraw

Мы сейчас просто посмотрим как можно инициализировать режимы **DirectDraw** для работы. **DirectDraw** - это часть **DirectX**.

DirectX API

DirectDraw - прямой доступ к видеопамяти

DirectSound - прямой доступ к звуковой карте

DirectPlay - прямой доступ к сетевым возможностям

для обеспечения multiplayer mode

DirectInput - поддержка игровых устройств ввода

**DirectDraw** обеспечивает прямой доступ к видеопамяти. Реализован **DirectDraw** в виде **COM** объектов.

Давайте создадим простое приложение с помощью **AppWizard**, как **SDI** с именем допустим **TestDraw**. Задача простая по выбору пункта меню инициализировать **DirectDraw**. Поэтому создайте пункт меню **DirectDraw** скажем в **View** и свяжите этот пункт меню с функцией

void CMainFrame::OnViewDirectdraw()

{

// TODO: Add your command handler code here

}

Итак, пора инициализировать. Для начала надо подключить заголовочный файл с описаниями функций

......

#include "TestDraw.h"

#include "MainFrm.h"

#include "ddraw.h"

......

Вот теперь нам нужно получить указатель на основной объект **DirectDraw** давайте это сделаем.

void CMainFrame::OnViewDirectdraw()

{

LPDIRECTDRAW lpDD;

HRESULT hdr;

hdr = DirectDrawCreate( NULL, &lpDD, NULL );

if( hdr != DD\_OK )

{

AfxMessageBox("Not Direct Create");

}

}

Мы создаем объект **DirectDrawCreate**. Но вот если сейчас скомпилировать то вы получите ошибку сборки. Конечно нужен **LIB** файл !!!. Нужно так нужно, как говорится в одной из реклам. Вот он **ddraw.lib**, вы должны поместить его в **Project Setting**.

Теперь мы можем включить режим доступа к монитору

hdr = lpDD->SetCooperativeLevel( m\_hWnd, DDSCL\_EXCLUSIVE | DDSCL\_FULLSCREEN );

if( hdr != DD\_OK )

{

AfxMessageBox("SetCooperativeLevel");

}

Ну теперь пора установить режим вывода на экран.

hdr = lpDD->SetDisplayMode( 800, 600, 8);

if( hdr != DD\_OK )

{

AfxMessageBox("Error SetDisplayMode");

}

При этом происходит физическое переключение, как я понимаю. Мой монитор гаснет, шелкает, и переходит в данный режим. Кстати если Вы обратили внимание, то я установил эксклюзивный режим и поэтому можно работать только с этой программой.

Конечно я понимаю, что это мало рассказанно про **DirectDraw**. Просто его инициализация и о данной теме стоит рассказывать отдельно.

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?192) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?194) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc4.html)  
Автор **Каев Артем**.

## Шаг 194 - Поддержка MAPI вручную

Давайте попробуем реализовать поддержку **MAPI** без всяких волшебников, а вручную. Для этого создадим приложение **AppWizard** на базе диалогового окна и поместим кнопку, к которой и припишем код инициализации **MAPI**.

В первую очередь нам нужно поместить заголовочный файл с описанием функций **MAPI**.

// TestMapiDlg.cpp : implementation file

//

.........

#include "mapix.h"

.........

Вот теперь надо подключить библиотеку с самим функциями в **Project Setting** и имя ее **Mapi32.lib**, ну все можно писать код.

void CTestMapiDlg::OnButton1()

{

HRESULT hR = MAPIInitialize(NULL);

ASSERT(S\_OK == hR);

LPMAPISESSION pS = NULL;

hR = MAPILogonEx(0,NULL,NULL,MAPI\_LOGON\_UI |

MAPI\_ALLOW\_OTHERS | MAPI\_EXTENDED,&pS);

if (S\_OK == hR) AfxMessageBox("Ok !");

else AfxMessageBox("Bad !");

MAPIUninitialize();

}

Смотрим чего там написано. Первая функция **MAPIInitialize** производит инициализацию данных в системных модулях **DLL**. Сама функция вот такая.

HRESULT

MAPIInitialize

(

LPVOID lpMapiInit

);

Но вместо структуры **lpMapiInit** можно передать **NULL**, что и делается. Если все нормально, то функция вернет **S\_OK**. Следующим вызовом является **MAPILogonEx**. Эта функция регистрации клиента для работы с **MAPI**.

HRESULT MAPILogonEx

(

ULONG ulUIParam,

LPTSTR lpszProfileName,

LPTSTR lpszPassword,

FLAGS flFlags,

LPMAPISESSION FAR \* lppSession

);

И последняя функция - это деинициализация клиента **MAPIUninitialize**, эта функция обратная **MAPIInitialize** и количество вызовов той и другой должно совпадать.

void MAPIUninitialize()

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?193) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?195) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc4.html)  
Автор **Каев Артем**.

## Шаг 195 - Создание своего вида

Как Вы уже знаете **MFC** уже имеет много классов видов. Естественно, что все Ваши потребности готовые классы удовлетворить не могут, поэтому мы с Вами попробуем изменить стандартный класс вида на свой манер. Давайте создадим приложение **MFC AppWizard** с именем **TestViews**. Пусть оно будет **SDI** и все дальше можете жать **Finish**.

Теперь нам надо создать класс порожденный от **CView**. Для этого выбираем **Insert -> New Class**, даем ему имя **MyView**, а базовым классом делаем **CView**. Нажимаем **OK** в результате у нас появляется новый класс. В проекте у нас уже есть класс вида **CTestViewsView**, который наследуется от **CView**, давайте унаследуем его от **MyView**, для начала в файле заголовка указываем на него ссылку.

.......

#endif // \_MSC\_VER > 1000

#include "MyView.h"

class CTestViewsView : public CView

.......

Теперь меняем папу.

......

class CTestViewsView : public MyView

{

......

Вот теперь в нашем классе **MyView** можно перегрузить виртуальную функцию **Create**.

BOOL MyView::Create(LPCTSTR lpszClassName, LPCTSTR lpszWindowName,

DWORD dwStyle, const RECT& rect, CWnd\* pParentWnd, UINT nID,

CCreateContext\* pContext)

{

// TODO: Add your specialized code here and/or call the base class

return CWnd::Create(lpszClassName, lpszWindowName, dwStyle,

rect, pParentWnd, nID, pContext);

}

А вот **CWnd::Create** можно заменить. Для этого посмотрим в исходных кодах **MFC** как это реализовано. Вот так.

BOOL CWnd::Create(LPCTSTR lpszClassName,

LPCTSTR lpszWindowName, DWORD dwStyle,

const RECT& rect,

CWnd\* pParentWnd, UINT nID,

CCreateContext\* pContext)

{

// can't use for desktop or pop-up windows (use CreateEx instead)

ASSERT(pParentWnd != NULL);

ASSERT((dwStyle & WS\_POPUP) == 0);

return CreateEx(0, lpszClassName, lpszWindowName,

dwStyle | WS\_CHILD,

rect.left, rect.top,

rect.right - rect.left, rect.bottom - rect.top,

pParentWnd->GetSafeHwnd(), (HMENU)nID, (LPVOID)pContext);

}

Но так можем и мы :-)

BOOL MyView::Create(LPCTSTR lpszClassName, LPCTSTR lpszWindowName,

DWORD dwStyle, const RECT& rect, CWnd\* pParentWnd,

UINT nID, CCreateContext\* pContext)

{

return CreateEx(0, lpszClassName, lpszWindowName,

dwStyle | WS\_CHILD | WS\_DLGFRAME ,

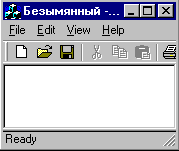
rect.left, rect.top,

rect.right - rect.left, rect.bottom - rect.top,

pParentWnd->GetSafeHwnd(), (HMENU)nID, (LPVOID)pContext);

}

Я специально добавил стиль **WS\_DLGFRAME**, чтобы показать возможность изменения. Посмотрите на результат этого изменения ниже и Вы увидите разницу в виде вида :-)



[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?194) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?196) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc4.html)  
Автор **Каев Артем**.

## Шаг 196 - Цвета элементов в диалоговом окне

Давайте разберемся как изменить цвета элементов в диалоговом окне. Вот такую цветовую раскраску я хочу получить. Этот шаг является некоторым развитием [шага 48](http://www.firststeps.ru/mfc/steps/r.php?48), в котором мы просто меняли стиль закраски окна. Теперь мы можем менять цвет отдельных элементов.

Создаем приложение **MFC AppWizard** на базе диалогового окна с именем **TestDlg**. Для управления цветами нам надо перехватить сообщение **WM\_CTRCOLOR**. Это можно сделать через **Add Windows Message Handler**. В таблице откликов должна появиться следующая запись.

BEGIN\_MESSAGE\_MAP(CTestDlgDlg, CDialog)

........

ON\_WM\_CTLCOLOR()

........

END\_MESSAGE\_MAP()

Общая мысль заключается в том, что есть возможность создать кисть для каждого отдельного типа элементов управления и сохранив ее использовать для закраски этих элементов вот и объявим ее в классе диалогового окна.

class CTestDlgDlg : public CDialog

{

......

//}}AFX\_DATA

CBrush\* m\_pEditBkBrush;

......

};

В конструкторе ее надо записать:

CTestDlgDlg::CTestDlgDlg(CWnd\* pParent /\*=NULL\*/)

: CDialog(CTestDlgDlg::IDD, pParent)

{

......

m\_pEditBkBrush = new CBrush(RGB(0, 0, 0));

}

И удалить при разрушении окна:

BOOL CTestDlgDlg::DestroyWindow()

{

// TODO: Add your specialized code here and/or call the base class

delete m\_pEditBkBrush;

return CDialog::DestroyWindow();

}

Ну, а теперь сама функция обработки цветов.

HBRUSH CTestDlgDlg::OnCtlColor(CDC\* pDC, CWnd\* pWnd, UINT nCtlColor)

{

switch (nCtlColor)

{

case CTLCOLOR\_EDIT:

{

pDC->SetTextColor(RGB(0, 255, 0));

pDC->SetBkColor(RGB(0, 0, 0));

return (HBRUSH)(m\_pEditBkBrush->GetSafeHandle());

}

default:

return (HBRUSH) CDialog::OnCtlColor(pDC, pWnd, nCtlColor);

}

}

В параметре **nCtlColor** передается тип элемента, к которому требуется тип закраски. И в зависимости от него мы возврашаем кисть.

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?195) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?197) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc4.html)  
Автор **Каев Артем**.

## Шаг 197 - Перемещение элемента в списке

Итак, задача будет связана с перемещением элемента в списке с помощью мышки. Можно будет схватить текст и мышкой его переместить в нужную позицию. Создаем приложение на базе диалогового окна. Пусть имя его будет **TestList**. Помещаем на него элемент управления **ListBox**. Вызываем **ClassWizard** и делаем его членом класса диалогового окна с типом **Control**.

class CTestListDlg : public CDialog

{

// Construction

public:

CTestListDlg(CWnd\* pParent = NULL); // standard constructor

// Dialog Data

//{{AFX\_DATA(CTestListDlg)

enum { IDD = IDD\_TESTLIST\_DIALOG };

CListBox m\_List;

//}}AFX\_DATA

......

Теперь в функции **OnInitDialog()** заполним список элементами.

BOOL CTestListDlg::OnInitDialog()

{

......

// TODO: Add extra initialization here

m\_List.AddString("1935");

m\_List.AddString("1936");

m\_List.AddString("1937");

m\_List.AddString("1938");

m\_List.AddString("1939");

return TRUE; // return TRUE unless you set the focus to a control

}

Ну и что дальше ??? А дальше ничего :-) Вы не сможете ничего сделать пока. Нам нужно перехватить нажатие и отпускание левой кнопки мыши на списке. Вы можете полазить по всем мастерам, но ничего похожего не найдете, а выход есть. Для того, чтобы подобные действия можно было производить нам нужно создать свой класс используя **new Class** как потомок от **CListBox** и жизнь точно покажется лучше. Добавляйте класс с именем **CMyList** и у Вас появится класс с таким описанием:

class CMyList : public CListBox

{

......

Давайте подменим его в классе диалога Вот так.

......

#endif // \_MSC\_VER > 1000

#include "MyList.h"

////////////////////////////////////////

// CTestListDlg dialog

class CTestListDlg : public CDialog

{

// Construction

public:

CTestListDlg(CWnd\* pParent = NULL); // standard constructor

// Dialog Data

//{{AFX\_DATA(CTestListDlg)

enum { IDD = IDD\_TESTLIST\_DIALOG };

CMyList m\_List;

......

Все это хорошо и на данную минуту работает. Теперь мы можем спокойно править класс **CMyList**. Первое, что мы сделаем это перегрузим события нажатия и отпускания кнопки мыши, с помощью **ClassWizard** конечно. Теперь он пригодится точно.

/////////////////////////////

// CMyList message handlers

void CMyList::OnLButtonDown(UINT nFlags, CPoint point)

{

// TODO: Add your message handler code here and/or call default

CListBox::OnLButtonDown(nFlags, point);

}

void CMyList::OnLButtonUp(UINT nFlags, CPoint point)

{

// TODO: Add your message handler code here and/or call default

CListBox::OnLButtonUp(nFlags, point);

}

Вот работать будет все так. Нажимается кнопка, запоминается позиция и текст. Отпускается кнопка, старая позиция удаляется, а в место новой вставляется старый текст и этот элемент выделяется. Значит нам нужно завести как минимум две переменные членов класса для того, чтобы хранить позицию и текст. Заведем.

class CMyList : public CListBox

{

......

// Generated message map functions

protected:

CString csOldText;

int intOldSel;

//{{AFX\_MSG(CMyList)

......

Ну что, запоминаем по нажатию текст и позицию ???

void CMyList::OnLButtonDown(UINT nFlags, CPoint point)

{

CListBox::OnLButtonDown(nFlags, point);

intOldSel=GetCurSel();

GetText(intOldSel,csOldText);

}

И по отпусканию мыши теперь это надо вставить и удалить.

void CMyList::OnLButtonUp(UINT nFlags, CPoint point)

{

// TODO: Add your message handler code here and/or call default

CListBox::OnLButtonUp(nFlags, point);

int intOnUpSel=GetCurSel();

DeleteString(intOldSel);

InsertString(intOnUpSel,csOldText);

SetCurSel(intOnUpSel);

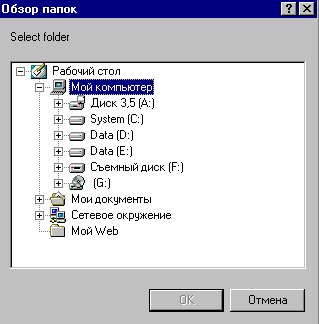
}

Вот и все :-)

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?196) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?198) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc4.html)  
Автор **Каев Артем**.

## Шаг 198 - Выбор только каталога и Shell API

Среди пользовательского интерфейса в **Windows** есть и набор функций под общим именем **Windows Shell API**. Вот одной из таких функций мы и воспользуемся для получения диалога выбора каталога. Вот как она будет выглядеть.



При создании проекта с данными возможностями нам требуется как минимум подключить **LIB** файл в **Project Setting Link** с именем **Shell32.lib** и заголовочный файл по месту назначения :-) с именем **Shlobj.h**. Итак, создавайте приложение на базе диалогового окна, ставьте в окно кнопку и привязывайте к ней событие нажатия, именно в нем мы и будем работать. Вот смотрите код.

BROWSEINFO bi;

TCHAR szDisplayName[MAX\_PATH];

LPITEMIDLIST pidl;

LPMALLOC pMalloc = NULL;

ZeroMemory(&bi, sizeof(bi));

bi.hwndOwner = NULL;

bi.pszDisplayName = szDisplayName;

bi.lpszTitle = TEXT("Select folder");

bi.ulFlags = BIF\_RETURNONLYFSDIRS;

pidl = SHBrowseForFolder(&bi);

if (pidl)

{

SHGetPathFromIDList(pidl, szDisplayName);

AfxMessageBox(szDisplayName);

}

Основу кода составляют две функции **SHBrowseForFolder** и **SHGetPathFromIDList**, одна из которых вызывает диалоговое окно, а вторая получает путь к каталогу.

WINSHELLAPI LPITEMIDLIST WINAPI SHBrowseForFolder

(

LPBROWSEINFO lpbi

);

WINSHELLAPI BOOL WINAPI SHGetPathFromIDList

(

LPCITEMIDLIST pidl,

LPSTR pszPath

);

Ну и естественно для ее использования нужно подготовить несколько структур. Основу составляет структура **BROWSEINFO**. Работает все так: после функции инициализации появляется указатель **LPITEMIDLIST pidl**, который используется для доступа ко всяким там функциям. А сам указатель появляется на основе заполнения структуры **BROWSEINFO**.

typedef struct \_browseinfo {

HWND hwndOwner;

LPCITEMIDLIST pidlRoot;

LPSTR pszDisplayName;

LPCSTR lpszTitle;

UINT ulFlags;

BFFCALLBACK lpfn;

LPARAM lParam;

int iImage;

} BROWSEINFO, \*PBROWSEINFO, \*LPBROWSEINFO;

Эта структура описывает кучу параметров, таких как родительское окно, заголовок диалогового окна, всякие там флаги и так далее.

К радости или сожалению **Shell API** достойна отдельной темы разговора или отдельного раздела.

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?197) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?199) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc4.html)  
Автор **Каев Артем**.

## Шаг 199 - Шаблоны, документы и работа с ними

Суть этого шага сводится к проблеме как нам получить все шаблоны существующие в документе и документы связанные с этими шаблонами. Общая структура в **MFC** представляет из себя дерево, которое из двух уровней. В **CWinApp** мы можем получать шаблоны, а из шаблонов документы.

CWinApp -> CDocTemplate

........

CDocTemplate -> CDocument

.........

Ну давайте попробуем ? Создайте приложение **MDI**, просто выберите тип и нажмите **Finish** и все. Потом создайте пункт в меню, к которому мы с Вами привяжем демонстрационный код. Реакция на событие пусть будет в классе документа.

/////////////////////

// CMy22Doc commands

void CMy22Doc::OnViewInfomdi()

{

}

Итак, нам нужно получить указатель на все шаблоны, которые есть у нас в документе. Такая возможность есть в классе **WinApp**, значит нам нужно сначала на него получить указатель.

void CMy22Doc::OnViewInfomdi()

{

CWinApp\* MyApp=AfxGetApp();

}

Теперь можно получить указатель и на шаблон документов. Это можно сделать функцией **GetFirstDocTemplatePosition**. Если шаблона нет, то вернется **NULL** - это есть начальная позиция для перебора.

void CMy22Doc::OnViewInfomdi()

{

// Step1

CWinApp\* MyApp=AfxGetApp();

// Step2

POSITION pos;

pos=MyApp->GetFirstDocTemplatePosition();

if (pos!=NULL)

{

AfxMessageBox("Yes Template");

}

}

Вот теперь мы спокойно можем идти по всем шаблонам используя функцию **GetNextDocTemplate**. Эта функция и обносит указатель, который при ситуации, что шаблона больше нет станет **NULL**.

void CMy22Doc::OnViewInfomdi()

{

// Step1

CWinApp\* MyApp=AfxGetApp();

// Step2

POSITION pos;

pos=MyApp->GetFirstDocTemplatePosition();

// Step 3

while (pos!=NULL)

{

CDocTemplate\* cd=MyApp->GetNextDocTemplate(pos);

}

}

Теперь мы можем получать документы из шаблона и из свойства. Общая логика опять повторяется.

void CMy22Doc::OnViewInfomdi()

{

// Step1

CWinApp\* MyApp=AfxGetApp();

// Step2

POSITION pos;

pos=MyApp->GetFirstDocTemplatePosition();

// Step 3

while (pos!=NULL)

{

CDocTemplate\* cd=MyApp->GetNextDocTemplate(pos);

// Step 4

if (cd!=NULL)

{

POSITION DocPos;

DocPos=cd->GetFirstDocPosition();

while (DocPos!=NULL)

{

CDocument\* cdoc=cd->GetNextDoc(DocPos);

if (cdoc!=NULL)

{

AfxMessageBox(cdoc->GetTitle());

}

}

}

}

}

Вот так можно получить список, например, всех возможно открытых документов из шаблонов. И вообще проводить групповые действия над всеми открытыми документами. Например, печатать все :-). Или закрыть или сохранить :-). Хорошо это.

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?198) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?200) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc4.html)  
Автор **Каев Артем**.

## Шаг 200 - О классе документа CDocument

Здесь на основе [прошлого шага](http://www.firststeps.ru/mfc/steps/r.php?199) я попробую ;-) рассказать о некорых функциях класса **CDocument**, которые могут Вам пригодиться совсем для разных целей. Вот в этот код из прошлого шага я буду добавлять использование разных функций.

void CMy22Doc::OnViewInfomdi()

{

.............

CDocument\* cdoc=cd->GetNextDoc(DocPos);

if (cdoc!=NULL)

{

// сюда !!!!!!!!!!!

AfxMessageBox(cdoc->GetTitle());

}

.............

}

Первая функция **GetPathName()** возврашает полный путь к документу на диске. Только если вы запустите приложение и посмотрите, то увидите пустую строку. Ее нет пока документ не сохранен на диске. Зачем использовать ??? Например, делать сложное хранение резервных копий при сохранении, что очень не плохо при наших электрических сетях, типа **\*.bk1**, **\*.bk2** и так далее.

// Описание

const CString& GetPathName( ) const

// вот можно посмотреть

if (cdoc!=NULL)

{

AfxMessageBox(cdoc->GetPathName());

}

Еще есть функция **GetTitle()**, которая возвращает имя документа. Это имя обычно совпадает с именем файла и находится в верхнем левом углу окна документа.

// Описание

const CString& GetTitle( ) const;

// вот можно посмотреть

if (cdoc!=NULL)

{

AfxMessageBox(cdoc->GetTitle());

}

Для определении были ли в докумен внесены изменения с последнего сохранения используется функция **IsModified**.

// Описание

BOOL IsModified( );

// вот можно посмотреть

if (cdoc!=NULL)

{

if (cdoc->IsModified()) AfxMessageBox("Modified");

}

Если вы произведете какие-то действия над приложением, например, удалите файл с диска, то класс документа не догадается о том, что документ не сохранен. Для этого можно воспользоваться функцией **SetModifiedFlag** и силовым методом сказать, что требуется напоминание о сохранении даже если ничего в классе документа не поменялось.

// Описание

void SetModifiedFlag( BOOL bModified = TRUE );

// вот можно посмотреть

if (cdoc!=NULL)

{

cdoc->SetModifiedFlag();

if (cdoc->IsModified()) AfxMessageBox("Modified");

}

Продолжая идею связанную со связями, то с каждым классом документа связаны виды один или несколько. Исходя из этого предыдущую схему можно расширить так

CWinApp -> CDocTemplate

........

CDocTemplate -> CDocument

.........

CDocument -> CView

.........

Опять тоже самое и иделогия такая же, смотрите, используются функции **GetFirstViewPosition** и **GetNextView**.

if (cdoc!=NULL)

{

POSITION ViewPos;

ViewPos=GetFirstViewPosition();

while (ViewPos!=NULL)

{

CView\* cv=GetNextView(ViewPos);

}

}

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?199) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?201) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc5.html)  
Автор **Каев Артем**.

## Шаг 201 - Функции прямого и обратного хода CWinApp - CView

Будем пробовать обобщать прошлые данные. Итак, мы можем от **CWinApp** дойти до любого **CView** и обратно от **CView** до **CWinApp**. Давайте разложим все по полочкам. Идем туда:

// CWinApp

AfxGetApp();

// CDocTemplate

GetFirstDocTemplatePosition();

GetNextDocTemplate(pos);

// CDocument

GetFirstDocPosition();

GetNextDoc(DocPos);

// CView

GetFirstViewPosition()

GetNextView(ViewPos);

Это мы с Вами уже делали в прошлых шагах. Теперь в обратную сторону от вида :-) Для точного понимания как и куда попадем :-). Для этого создадим новый пункт в меню из прошлого проекта и привяжем его к классу вида :-)

////////////////////////////////////////////////

// CMy22View message handlers

void CMy22View::OnViewInfomdiview()

{

}

Вот так и пойдем вверх. Первое, что мы должны получить - это к какому документу привязан вид. Это можно сделать используя функцию вида **GetDocument()**

//

CDocument\* GetDocument( ) const;

//

void CMy22View::OnViewInfomdiview()

{

CDocument\* cd=GetDocument();

if (cd!=NULL)

{

AfxMessageBox("Cool");

}

else AfxMessageBox("Error Document");

}

Теперь нам надо получить шаблон, в котором содержится документ. Это можно сделать воспользовавшись **GetDocTemplate()**

//

CDocTemplate\* GetDocTemplate( ) const;

//

void CMy22View::OnViewInfomdiview()

{

CDocument\* cd=GetDocument();

if (cd!=NULL)

{

CDocTemplate\* cDoc=cd->GetDocTemplate();

if (cDoc!=NULL)

{

AfxMessageBox("Cool");

}

else AfxMessageBox("Error Template");

}

else AfxMessageBox("Error Document");

}

Ну, а получение **CWinApp** знакомое через **AfxGetApp** делали уже. Теперь мы можем создать схему обратного хода.

// CView

GetDocument();

// CDocument

GetDocTemplate()

//CDocTempate

AfxGetApp(); // с любого места программы

С этого момента Вы свободны и можете двигаться в любом направлении из любого класса схемы Документ-Вид, вызывать любые функции, синхронизировать разные документы и вообще творить много чего.

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?200) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?202) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc5.html)  
Автор **Каев Артем**.

## Шаг 202 - Сохранение файлов и CDocument

То что в результате сохранения данных из класса документа вызывается функция **Serialize** мы с Вами уже знаем. Вот такой код гененирует **AppWizard**.

///////////////////////////////////////////

// CTestSaveDoc serialization

void CTestSaveDoc::Serialize(CArchive& ar)

{

if (ar.IsStoring())

{

// TODO: add storing code here

}

else

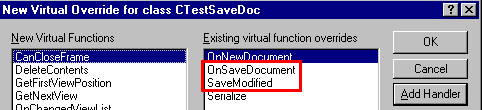
{

// TODO: add loading code here

}

}

Но мы то с Вами хотим больше и больше и еще больше контроля над процессом бритья ой :-) сохранения. Так вот в классе **CDocument** есть возможность улучшить контроль над сохранением файлов и добавить свои специфические действия. Это делается путем перегрузки виртуальных функций. Как их увидеть ??? Выделите класс потомок от **CDocument**, после этого нажмите правую кнопку мыши и выберите пункт меню **Add Virtual Funсtion**. Нас интересуют функции связанные с сохранением документа, а это **OnSaveDocument(LPCTSTR lpszPathName)** и **SaveModified()**.



Перегрузите их для исследования и добавьте следующий код с сообщениями

//////////////////////////////////

// CTestSaveDoc commands

BOOL CTestSaveDoc::SaveModified()

{

AfxMessageBox("SaveModified()");

return CDocument::SaveModified();

}

BOOL CTestSaveDoc::OnSaveDocument(LPCTSTR lpszPathName)

{

AfxMessageBox("OnSaveDocument(LPCTSTR lpszPathName)");

return CDocument::OnSaveDocument(lpszPathName);

}

Для полноты картины создадим пукт меню, который будет сообщать о том, что документ изменен. Мы это делали уже.

void CTestSaveDoc::OnEditModified()

{

SetModifiedFlag();

}

Вот. Давайте разбираться когда вызывается какая функция. **OnSaveDocument()** вызывается всегда когда сохраняется документ, в том числе и пунктами меню **Save** и **Save As...** из приложений созданных **App Wizard**. Так же Вы сами можете вызывать эту функцию во всяких специфических ситуациях, например, при получении сигнала с блока питания о перебоях с питанием :-). Эта функция вызовет **Serialize**.

virtual BOOL OnSaveDocument( LPCTSTR lpszPathName );

Функция **SaveModified()** вызывается перед закрытием измененного документа и именно она выводит диалоговое окно о том, что документ был изменен. Ее вам надо переопределять для того, чтобы вставить свои специфические действия. Какие ??? Ну, например, сообщить серверу кто правил документ, записать это в БД чтобы потом найти крайнего и виноватого :-). Общая схема такая.

Сохранение документа

OnSaveDocument

Serialize

Закрытие документа

SaveModified

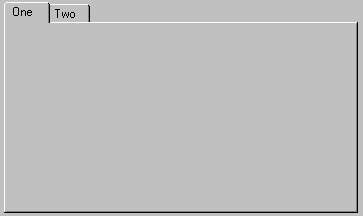
OnSaveDocument

Serialize

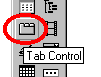
[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?201) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?203) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc5.html)  
Автор **Каев Артем**.

## Шаг 203 - Использование CTabCtrl

Настало время очередного элемента управления, теперь это **CTabCtrl**. Он довольно часто встречается в интерфейсе **Windows** и должен быть Вам знаком. Выглядит он вот так.



Размещать этот элемент можно и динамически и на панели диалога. На панели диалога это делается вот с помощью этой иконы.



Давайте создадим проект с именем **TestTab** на базе диалогового окна и нажмем **Finish**. Поместим на него элемент управления **Tab Control** и с помощью **Class Wizard** создадим для него переменную типа **Control**.

class CTestTabDlg : public CDialog

{

......

CTabCtrl m\_Tabs;

......

};

Если Вы запустите программу, то увидите, что на диалоге только одна вкладка. Для того, чтобы их было много их надо создать. Лучшее место на мой взгляд это в **OnInitDialog()**. Вот код создающий две закладки.

BOOL CTestTabDlg::OnInitDialog()

{

......

// TODO: Add extra initialization here

TC\_ITEM tci;

tci.mask = TCIF\_TEXT;

tci.iImage = -1;

tci.pszText = "One";

m\_Tabs.InsertItem(0, &tci);

tci.pszText = "Two";

m\_Tabs.InsertItem(1, &tci);

return TRUE; // return TRUE unless you set the focus to a control

}

Вы заполняете структуру **TC\_ITEM**, а потом используете функцию **InsertItem** для вставки закладки.

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?202) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?204) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc5.html)  
Автор **Каев Артем**.

## Шаг 204 - Экзотические форматы файлов - Access-VC

Интересно, а Вы можете работать со следующими форматами файлов совершенно одинаково ???

MDB

TXT

EXCEL

HTML

DBF

PARADOX

и Все что связано с ODBC ????

Я имею ввиду одинаково не меняя стиль программирования ??? Можете !!! Если будете использовать **Access** и **MDB**. Идея основана на том, что в **ACCESS** можно устанавливать связь с такими документами и они для Вас будут выглядеть как обычные таблицы **Access** вот так. Для убедительности я приведу пример. Мы с вами будем работать с таблицей **HTML** с помощью приложения на **VC** через **Access**. Для начала надо создать таблицу в **HTML**, я сделал такую.

<HTML>

<TABLE>

<TR>

<TD> Петя</TD>

<TD> Машинист</TD>

</TR>

<TR>

<TD>Вася

<TD>Водитель

</TR>

<TR>

<TD>Дима</TD>

<TD>Директор</TD>

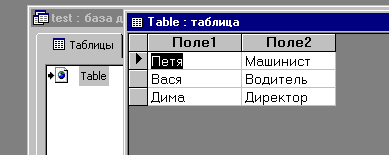
</TR>

</TABLE>

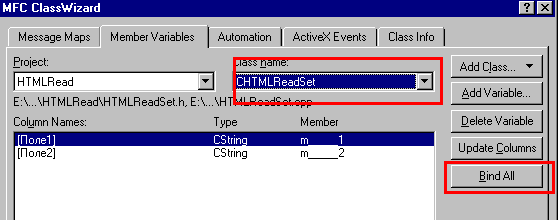
</HTML>

И назвал файл **test.html**.

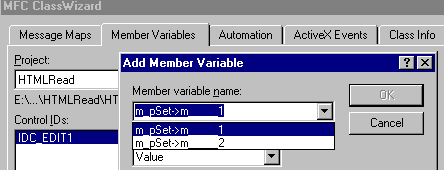
Теперь ссылку на нее надо сделать в базу данных **Access**. Я создал каталог **Test**, поместил туда **HTML** и создал там базу данных **test.mdb** и ссылку. Вот как это выглядит:



Итак, все настроено и можно создавать приложение на **VC**. Создаем приложение **MFC AppWizard** с именем **HTMLRead**. Пускай будет **Single**, а вот на **Step 2** надо установить опцию **Database View without file support**. В **DataSource** выбрать **DAO** и нашу базу с таблицей. После этого можно расслабиться и нажать **Finish**. Теперь нам надо создать переменные связанные с полями, это делается кнопкой **Build All** здесь.



После этого Вы можете связывать элементы скажем **CEdit** на форме с этими элементами, вот так:

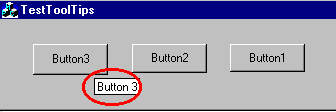


Ну вот свяжите и запустите приложение. Все будет работать и Вы сможете передвигаться по записям, просматривать их и так далее. Ну это же классно !!! Мне нравится !!!

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?203) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?205) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc5.html)  
Автор **Каев Артем**.

## Шаг 205 - Подсказка к элементам управления в диалоговых окнах

Вот чего я хочу, смотрите рисунок. Это посказка к кнопке в диалоговом окне. Называется это **ToolTips**:



Итак создаем приложение как **MFC AppWizard** на основе диалогового окна с именем **TestTip**. Помещаем на него три кнопки. Используя **ClassWizard** создаем соответствующий кнопкам класс типа **Control**. Вот так:

class CTestTipDlg : public CDialog

{

......

// Dialog Data

//{{AFX\_DATA(CTestTipDlg)

enum { IDD = IDD\_TESTTIP\_DIALOG };

CButton m\_B3;

CButton m\_B2;

CButton m\_B1;

//}}AFX\_DATA

......

};

Теперь нам надо создать указатель на класс подсказки как **CToolTipCtrl**:

class CTestTipDlg : public CDialog

{

// Construction

public:

CToolTipCtrl\* m\_pTips;

......

};

В конструкторе класса диалога обнулим указатель. Для надежности :-)

CTestTipDlg::CTestTipDlg(CWnd\* pParent /\*=NULL\*/)

: CDialog(CTestTipDlg::IDD, pParent)

{

.......

m\_pTips=NULL;

}

Создадим деструктор для того, чтобы при разрушении класса освободить память.

class CTestTipDlg : public CDialog

{

......

CTestTipDlg(CWnd\* pParent = NULL); // standard constructor

~CTestTipDlg();

......

};

CTestTipDlg::~CTestTipDlg()

{

if (m\_pTips!=NULL) delete m\_pTips;

}

}

Теперь нам надо создать сам класс подсказки и связать подсказку с элементами на диалоговом окне. Мы это сделаем в **OnInitDialog()**.

BOOL CTestTipDlg::OnInitDialog()

{

...........

if (!m\_pTips)

{

m\_pTips =new CToolTipCtrl;

m\_pTips->Create(this);

m\_pTips->AddTool(&m\_B1,"Button 1");

m\_pTips->AddTool(&m\_B2,"Button 2");

m\_pTips->AddTool(&m\_B3,"Button 3");

m\_pTips->Activate(TRUE);

}

return TRUE; // return TRUE unless you set the focus to a control

}

Следующий шаг заключается в передаче сообщения классу подсказки:

BOOL CTestTipDlg::PreTranslateMessage(MSG\* pMsg)

{

if (NULL != m\_pTips)

m\_pTips->RelayEvent(pMsg);

return CDialog::PreTranslateMessage(pMsg);

}

Ну вот и все. Запускайте и наслаждайтесь подсказками, что мол эта кнопка номер один, а эта номер два.

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?204) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?206) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc5.html)  
Автор **Каев Артем**.

## Шаг 207 - VB - VC перенос приложения

Спору нет, что в **VB** часто легче создать начальный интерфейс и показать, например, начальнику или заказчику и решить проблеммы с проектированием. Но вот если проект пишется все-таки на **VC**, то придется полностью переносить приложение. Но некоторые сервисные возможности по этому поводу заложены.

Итак, в **VB** окна приложения это формы (**Forms**). Формы - это обычные диалоги. Все приложение на **VB** когда оно сохраняется на диске выглядит примерно так.

test.vbp Файл проекта

form1.frm -

......... | Формы или диалоги

......... |

form\_n.frm -

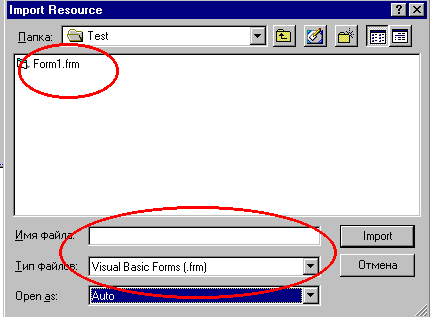
модуль1.bas -

.......... | Файлы модулей

.......... |

модуль\_n.bas -

Так вот формы **VB** так как они практически тоже самое, что и диалоги переносяться в **VC**. Это просто классно. Попробуйте создайте на **VB** простую форму, скажем с двумя кнопками. Сохраниите ее у нее будет расширение **FRM**. Теперь зайдите в **VC**, создайте любое приложение с ресурсами. Выберите кнопку **Import** находясь в окне с ресурсами. В списке вариантов найдите **Visual Basic forms**.



С более сложными вещами, например, с управляющими элементами появились проблемы. Я создал на форме источник записей и список связанный с данными. Вот тут то и началось, что этот элемент не установлен, а там ошибка. Вообщем этот шаг написан для демонстрации. Если кто серьезно будет заниматься этой проблемой пишите.

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?206) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?208) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc5.html)  
Автор **Каев Артем**.

## Шаг 208 - CStringArray и CListBox

По смыслу эти два класса похожи. И тот и другой умеет хранить строки. Только один это элемент графического интерфейса, а другой контейнер для строк. Наша задача научиться обмениваться данными между ними вручную. То есть не используя **DDX**, а путем создания нашей функции.

Итак, создаем приложение на основе диалогового окна с именем **TestString**. Помещаем на него элемент управления **CListBox** и связываем с ним класс (**Control**) вот так.

class CTestStringDlg : public CDialog

{

......

// Dialog Data

//{{AFX\_DATA(CTestStringDlg)

enum { IDD = IDD\_TESTSTRING\_DIALOG };

CListBox m\_ListBox;

......

Там же создаем массив на основе класса **CStringArray**.

class CTestStringDlg : public CDialog

{

// Construction

......

public:

CStringArray csArray;

......

Теперь создадим функцию передачи строк в элемент управления и из него.

class CTestStringDlg : public CDialog

{

// Construction

public:

void TransferData(CListBox\* cl, BOOL blGoy);

CStringArray csArray;

......

И ее реализация:

void CTestStringDlg::TransferData(CListBox\* cl, BOOL blGoy)

{

CString st;

if (blGoy)

{

for (int i=0; i < csArray.GetSize();i++)

{

st=csArray[i];

cl->AddString(st);

}

}

else

{

int iSize;

iSize=cl->GetCount();

csArray.RemoveAll();

for (int i=0;i < iSize;i++)

{

cl->GetText(i,st);

csArray.Add(st);

}

}

Как видите при установке флага **blGoy** данные либо пересылаются в массив, либо из него. Теперь нам надо дописать код в **OnInitDialog** для того, чтобы инициализировать набор строк при запуске диалога:

////////////////////////////////

// CTestStringDlg message handlers

BOOL CTestStringDlg::OnInitDialog()

{

CDialog::OnInitDialog();

...........

// TODO: Add extra initialization here

csArray.Add("Hello");

csArray.Add("Word");

TransferData(&m\_ListBox,TRUE);

return TRUE; // return TRUE unless you set the focus to a control

}

И перегрузка нажатия на кнопку для возврата назад.

void CTestStringDlg::OnOK()

{

TransferData(&m\_ListBox,FALSE);

CDialog::OnOK();

}

Вот имея такой код, например, можно создать свой класс на основе **CListBox** более интелектуальней чем базовый.

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?207) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?209) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc5.html)  
Автор **Каев Артем**.

## Шаг 209 - Заставка при запуске приложения

Привет Артем.

Решил принять участие в организации "Русского MSDN".

Посылаю пример создания заставки при загрузке приложения.

SLY - sly@vmz.nnov.ru

Спасибо большое. Вот мое описание а исходные код загрузите из проекта.

Там все что прислал SLY.

Я просто сделаю комментарии к коду написанного **SLY**, оставив так как есть. Давайте смотреть. Создано приложение на базе диалогового окна, для которого и должна появляться заставка. Сам класс заставки порожден от класса **CWnd**.

class CGRSplashWin : public CWnd

{

// Construction

public:

CGRSplashWin();

BOOL Create(CWnd\* pParent);

// Attributes

public:

// Operations

public:

// Overrides

// ClassWizard generated virtual function overrides

//{{AFX\_VIRTUAL(CGRSplashWin)

protected:

virtual BOOL PreCreateWindow(CREATESTRUCT& cs);

//}}AFX\_VIRTUAL

// Implementation

public:

virtual ~CGRSplashWin();

// Generated message map functions

protected:

//{{AFX\_MSG(CGRSplashWin)

afx\_msg BOOL OnEraseBkgnd(CDC\* pDC);

//}}AFX\_MSG

DECLARE\_MESSAGE\_MAP()

};

Сама реализация класса довольно проста. Интересным моментом является получение фона окна реализованного в функции **OnEraseBkgnd**.

BOOL CGRSplashWin::OnEraseBkgnd(CDC\* pDC)

{

//////////////////////////////////////////////////////////

//создадим фон представления

//////////////////////////////////////////////////////////

BITMAP m\_bkparam;

CBitmap m\_bkview;

CBitmap\* m\_oldbkview;

CDC m\_memDC;

RECT rect;

m\_bkview.LoadBitmap(IDB\_BKVIEW);

m\_bkview.GetObject(sizeof(m\_bkparam),&m\_bkparam);

m\_memDC.CreateCompatibleDC(pDC);

m\_oldbkview=m\_memDC.SelectObject(&m\_bkview);

GetClientRect(&rect);

//количество строк и столбцов для представления

int NumRows=(int)(rect.bottom/m\_bkparam.bmHeight);

int NumCols=(int)(rect.right/m\_bkparam.bmWidth);

//заполнить всю рабочую область представления

for (int row=0;rowBitBlt(col\*m\_bkparam.bmWidth,

row\*m\_bkparam.bmHeight,m\_bkparam.bmWidth,

m\_bkparam.bmHeight,&m\_memDC,0,0,SRCCOPY);

SelectObject(m\_memDC,m\_oldbkview);

return 1;

}

Вот главный вопрос - это когда заставку надо создать и когда удалить. Создаем мы ее в классе приложения используя функцию **InitInstance**.

BOOL CSplashWinApp::InitInstance()

{

......

#ifdef \_AFXDLL

Enable3dControls(); // Call this when using MFC in a shared DLL

#else

Enable3dControlsStatic(); // Call this when linking to MFC statically

#endif

//////////////////////////////////////////////

//показать заставку

//уничтожить ее при активизации диалога

//WM\_ACTIVATE

//////////////////////////////////////////////

m\_wndSplash = new CGRSplashWin();

m\_wndSplash->Create(m\_wndSplash->FromHandle(GetDesktopWindow()));

m\_wndSplash->ShowWindow(SW\_NORMAL);

CSplashWinDlg dlg;

m\_pMainWnd = &dlg;

......

Сам класс заставки объявлен в классе приложения:

class CSplashWinApp : public CWinApp

{

public:

CSplashWinApp();

CGRSplashWin \*m\_wndSplash;

......

А удаляется заставка в классе диалога при его активации:

void CSplashWinDlg::OnActivate(UINT nState, CWnd\* pWndOther, BOOL bMinimized)

{

CDialog::OnActivate(nState, pWndOther, bMinimized);

////////////////////////////////////////////

//сделаем задержку а то вдруг не увидим

//если прога быстро загружается

////////////////////////////////////////////

Sleep(1000);

if (((CSplashWinApp\*)AfxGetApp())->m\_wndSplash!=NULL)

{

delete ((CSplashWinApp\*)AfxGetApp())->m\_wndSplash;

((CSplashWinApp\*)AfxGetApp())->m\_wndSplash=NULL;

}

}

Вот у кого остались вопросы грузите проект и пишите **SLY**.

Материалы для шага прислал SLY ([sly@vmz.nnov.ru](mailto:sly@vmz.nnov.ru?SUBJECT=FromFirstStepSite)).

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/209.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?208) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?210) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc5.html)  
Автор **Каев Артем**.

## Шаг 210 - Градиентная заливка

----- Original Message -----

From: SLY

To: kaev@yandex.ru

Sent: Wednesday, July 26, 2000 7:15 PM

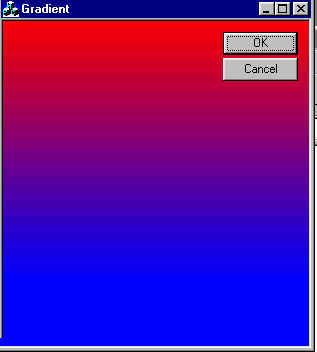
Subject: MFCByStep

Привет всем ! Посылаю пример организации градиентной заливки контекста

окна. Пример реализован для диалогового окна но может быть применен где угодно

так как сама заливка реализована в виде функции GradientFillRect

Вот как она выглядит:



Давайте посмотрим код. Реализуется эта заливка в функции **OnPaint**. То есть при перерисовке диалогового окна вызывается функция **GradientFillRect**.

void CGradientDlg::OnPaint()

{

if (IsIconic())

{

.........

}

else

{

CPaintDC dc(this); // получаем контекст устройтсва для рисования

CRect rect;

GetClientRect(rect); // получаем размер области рисования

// вызываем функцию рисования

GradientFillRect(dc.GetSafeHdc(),rect,RGB(255,0,0),RGB(0,0,255),128);

}

}

Теперь при изменении размера окна вызывется перерисовка, используя **Invalidate** в ответ на сообщение **WM\_SIZE**.

void CGradientDlg::OnSize(UINT nType, int cx, int cy)

{

CDialog::OnSize(nType, cx, cy);

//обновить при изменении размеров

Invalidate();

}

Ну вот и сама функция:

void CGradientDlg::GradientFillRect(HDC pDC, CRect ARect,

COLORREF StartColor,COLORREF EndColor, int Colors)

{

//если квадрат пустой то выйти

if (ARect.IsRectEmpty()) return;

HBRUSH Brush;

//если кол-во цветов меньше двух то закрасить первым цветом и выйти

if (Colors < 2)

{

Brush = CreateSolidBrush(StartColor);

FillRect(pDC, ARect, Brush);

DeleteObject(Brush);

return;

}

BYTE StartRGB[3];

int RGBDelta[3];

CRect ColorBand;

int I,Delta;

//начальные состовляющие цвета

StartRGB[0] = GetRValue(StartColor);

StartRGB[1] = GetGValue(StartColor);

StartRGB[2] = GetBValue(StartColor);

//шаг изменения цвета

RGBDelta[0] = GetRValue(EndColor) - StartRGB[0];

RGBDelta[1] = GetGValue(EndColor) - StartRGB[1];

RGBDelta[2] = GetBValue(EndColor) - StartRGB[2];

ColorBand = ARect;

Colors = max(2, min(Colors, ARect.Height()));

Delta = (int)(ARect.Height() / Colors);

//закрасить основную часть окна

if (Delta>0)

for (I=0;I<Colors;I++)

{

ColorBand.top = ARect.top + I \* Delta;

ColorBand.bottom = ColorBand.top + Delta;

Brush = CreateSolidBrush(RGB(

StartRGB[0] + MulDiv(I, RGBDelta[0], Colors - 1),

StartRGB[1] + MulDiv(I, RGBDelta[1], Colors - 1),

StartRGB[2] + MulDiv(I, RGBDelta[2], Colors - 1)));

FillRect(pDC, ColorBand, Brush);

DeleteObject(Brush);

}

//закрасить оставшуюся часть окна

div\_t Res;

Res=div(ARect.Height() , Colors);

Delta =Res.rem;//остаток от деления

if (Delta>0)

for (I=0;I<Colors;I++)

{

ColorBand.top = ARect.bottom - Delta;

ColorBand.bottom = ColorBand.top + Delta;

Brush = CreateSolidBrush(EndColor);

FillRect(pDC, ColorBand, Brush);

DeleteObject(Brush);

}

}

Вот так оно и красится. Если есть еще вопросы смотрите проект.

Материалы для шага прислал SLY ([sly@vmz.nnov.ru](mailto:sly@vmz.nnov.ru?SUBJECT=FromFirstStepSite)).

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/210.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?209) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?211) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc5.html)  
Автор **Каев Артем**.

## Шаг 211 - Таскаем форму за любое место

----- Original Message -----

From: SLY

To: kaev@yandex.ru

Sent: Friday, July 28, 2000 7:21 PM

Subject: MFCByStep

Привет!

Вот еще одна хитрость как обмануть Windows - перетаскивание окна

за форму при нажатии на левую кнопку mouse.

Вот история такая. Я тут два дня убил на тестирование :-) Пытался понять в чем прикол. Ведь она и левой кнопкой такскается сама по себе. Поэтому я таскал правой. Вся радость в том, что окно таскается только за заголовок, а в данном примере за любое место формы :-) Будьте внимательны. Создайте **AppWizard** приложение и посмотрите. Схватите мышкой в середине формы. Ни куда она не потянется :-). А вот, чтобы она передвигалась надо перегрузить сорбытие нажатия левой кнопки мыши через **Add Windows Message Handle** и написать такой код.

void CMoveWindowDlg::OnLButtonDown(UINT nFlags, CPoint point)

{

CDialog::OnLButtonDown(nFlags, point);

ReleaseCapture();

::SendMessage(AfxGetApp()->m\_pMainWnd->GetSafeHwnd(),

WM\_SYSCOMMAND, SC\_MOVE | HTCAPTION, 0);

}

Идея вообщем такая. Надо перенаправить сообщение передвижения мышью **SC\_MOVE** туда кто умеет его обрабатывать **HTCAPTION**.

Материалы для шага прислал SLY ([sly@vmz.nnov.ru](mailto:sly@vmz.nnov.ru?SUBJECT=FromFirstStepSite)).

[Загрузить проект](http://www.firststeps.ru/mfc/steps/prog/211.rar) | [Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?210) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?212) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc5.html)  
Автор **Каев Артем**.

## Шаг 212 - Объявление глобального объекта

Итак, задача объявить объект так, чтобы к нему был доступ из любого места в программе. Давайте сделаем это на основе объекта порожденного от класса **CWinApp**. Создавайте приложение на базе **MFC App Wizard**, как **SDI** например. Имя можно дать **TestGlobal**.

Для начала объявим ссылку на этот объект. Рядом с его определением. Вот так.

//////////////////////////////////////////////

// The one and only CTestGlobalApp object

CTestGlobalApp theApp;

**CTestGlobalApp\* NEAR m\_gpMyApp = &theApp;.**

Теперь нам надо перейти в **SDIAFX.H** и дописать сслыку на этот объект как внешнюю.

//{{AFX\_INSERT\_LOCATION}}

**#include "TestGlobal.h"**

**extern CTestGlobalApp\* NEAR gpMyApp;**

// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX\_STDAFX\_H\_\_E2879A47\_6C78\_11D4\_9411\_CEDF8A8E405C\_\_INCLUDED\_)

С этого момента в любом месте вы сможете иметь доступ через указатель к функциям класса. Давайте посмотрим. Сможем ли мы вызвать методы объекта из вида не вызывая функцию **AfxGetApp()**.

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?211) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?213) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc5.html)  
Автор **Каев Артем**.

## Шаг 213 - Работа с реестром CRegKey

Мы уже несколько раз касались работы с реестром, но ни разу не работали с классом **CRegKey**. А он на мой взгляд наиболее удачно сделан. Давайте создадим приложение на базе диалогового окна. Поместим туда элемент управления **Edit** и две кнопки для тестов. Для того, чтобы Вы могли работать с классом **CRegKey** Вам нужно подключить заголовочный файл.

#include "atlbase.h"

Для элемента редактирования я привязал переменную:

class CTestRegDlg : public CDialog

{

......

// Dialog Data

//{{AFX\_DATA(CTestRegDlg)

enum { IDD = IDD\_TESTREG\_DIALOG };

CString m\_Edit;

//}}AFX\_DATA

......

Вот код для первой кнопки. По нажатию на нее Вы получите информацию из реестра.

void CTestRegDlg::OnButton1()

{

CRegKey key;

LONG status = key.Open(HKEY\_LOCAL\_MACHINE, "Config\\0001\\Display\\Settings");

if (status == ERROR\_SUCCESS)

{

char setting[255];

DWORD settingSize=sizeof(setting);

status = key.QueryValue(setting, "Resolution", &settingSize);

if (status == ERROR\_SUCCESS)

{

CString s(setting);

m\_Edit=s;

UpdateData(FALSE);

}

else AfxMessageBox("Not Read Params");

}

else AfxMessageBox("Error Open Key");

key.Close();

}

Что здесь написано ??? Создаем объект класса **CRegKey key;**, открываем раздел реестра **key.Open**, если мы смогли это сделать **status == ERROR\_SUCCESS** тогда мы можем получить значение параметров **key.QueryValue**. Классно и удобно. В этом коде мы получим разрешение экрана в элементе редактирования. В конце объект реестра нужно закрыть **key.Close();**.

Давайте посмотрим как можно создавать разделы:

void CTestRegDlg::OnButton2()

{

CRegKey key;

LONG status = key.Create(HKEY\_CURRENT\_USER, "Software\\Step By Step");

if (status == ERROR\_SUCCESS)

{

status = key.Create(key.m\_hKey,"MFC Step by Step");

if (status == ERROR\_SUCCESS)

{

}

else AfxMessageBox("Not Create Params");

}

else AfxMessageBox("Error Create Key");

key.Close();

}

Опять создаем объект класса **CRegKey key** и создаем раздел реестра **key.Create()**, если создали **if (status == ERROR\_SUCCESS)**, тогда можем создать подраздел **key.Create()**. Вот так выглядит результат в реестре.



Если провести совершенно общий обзор этого класса так как это шаги, а не подробности :-) То вот главные функции

CRegKey::Open Открыть раздел

CRegKey::Create Создать раздел

CRegKey::QueryValue Получить значение

CRegKey::SetValue Установить значение

CRegKey::Close Закрыть раздел

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?212) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?214) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc5.html)  
Автор **Каев Артем**.

## Шаг 214 - Версии и типы DLL MFC

Этот раздел может быть и не Важен, если Вы разрабатываете приложения для себя, но вот если Вы собираетесь его распространять неплохо бы иметь некоторые знания по данному вопросу. В любом случае инсталляцию надо представлять как будто она производится на пустой компьютер с **Windows**. Так что давайте выясним некоторые подробности. Первое, написанный Вами код не будет включать все, что необходимо вашей программе, особенно если вы пользуетесь **Dinamic Link**. **MFC** появилась не вчера и у нее есть тоже версии. Весь код **MFC** находится в **DLL** и эти файлы имеют имена типа **MFCxx.DLL**. Ну, например:

......

MFC30.DLL

MFC40.DLL

......

Соотвественно Вам неплохо бы знать какую Версию Вы используете для компиляции, и при этом такая же **DLL** должна быть в системе. На данный момент **MFCxx.DLL** является частью системы **Windows**. Слишком много программного обеспечения используют их для работы. Например, в поставке **Windows 95** есть такие библиотеки:

Win95\_02.cab

mfc30.dll 322,832 07-11-1995 9:50:00a

mfcd30.dll 55,808 07-11-1995 9:50:00a

mfcn30.dll 15,872 07-11-1995 9:50:00a

mfco30.dll 133,392 07-11-1995 9:50:00a

Win95\_03.cab

mfcans32.dll 133,904 07-11-1995 9:50:00a

**Debug** версии отличаются от версий **Release** наличием буковки **D**. А вот это немного опаснее. Так как, если Вы скомпилировали отладочную версию, то надеяться на то, что в системе уже есть такие библиотеки точно не стоит. Вот пример соответствия.

Release Debug

MFCxx.DLL MFCxxD.DLL

MFCNxx.DLL MFCNxxD.DLL

Соотвественно **DLL** имеющие отношений к **UNICODE** тоже будут иметь свою буковку **U**. Все остальное по правилам. Вот смотрите.

Unicode Unicode Debug

MFCxx.DLL MFCxxU.DLL MFCxxUD.DLL

Естественно, что **MFC** по причине своего размера разбита на части в виде отдельных **DLL**:

MFCxx.ВLL Основная

MFCOxx.DLL OLE

MFCDxx.DLL Базы данных

MFCNxx.DLL Сеть

Основной модуль **MFC** может быть выполнен в виде **static** или **dinamic** компоновки. Они тоже отличаются по букве **S**.

Динамическая компоновка Статическая компоновка

MFCxx.LIB MFCSxx.LIB

Есть и отдельные заплатки, например, библиотека **MFCANS32.DLL** занимается конвертацией **UNICODE-MBCS**. Но это специфичная библиотека.

Все библиотеки **MFCxx.DLL** используют <>MSVCRTxx.DLL, то есть она тоже должна обязательно быть установлена. Эта библиотека используется для распределения памяти. Кстати она и не идет в поставке **Windows 95**, поэтому Вы должны позаботиться о ее наличии.

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?213) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?215) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc5.html)  
Автор **Каев Артем**.

## Шаг 215 - Познакомимся поближе с CObject

Посмотрите иерархию классов **MFC** и Вы увидите, что очень много объектов порождено от **CObject**. Так как он находится в самом верху иерархии классов, то естественно он обеспечивает только самые базовые возможности для объекта. К базовым возможностям относятся вот какие:

* Преобразование в последовательную форму
* Информация о классе
* Отладочная информация
* Совместимость с классами коллекций

Всего функций в классе немного вот они:

// --------------- Конструктора объекта

CObject( ); // Конструктор по умолчанию

CObject( constCObject& objectSrc ); // Конструктор копий

void\* operator new( size\_t nSize ); // Оператор создания объекта

throw( CMemoryException );

void\* operator new( size\_t nSize, LPCSTR lpszFileName, int nLine ); // Второй вариант создания объекта

throw( CMemoryException );

void operator delete( void\* p ); // Оператор удаления объекта

void operator =( const CObject& src );// Оператор получения копии

// -------------- Диагностика

virtual void AssertValid( ) const; // Проверка состояния объекта

virtual void Dump( CDumpContext& dc ) const; // Получения информации об объекте

// -------------- Сохранение обьекта

BOOL IsSerializable( ) const; // Проверка возможности сохранения объекта

virtual void Serialize( CArchive& ar );// Сохранение объекта

throw( CMemoryException );

throw( CArchiveException );

throw( CFileException );

//--------------- Дополнительные функции

virtual CRuntimeClass\* GetRuntimeClass( ) const; // Возвращает информацию о классе

BOOL IsKindOf( const CRuntimeClass\* pClass ) const; // Проверка получен ли объект из данного класса

Как видите в классе **CObject** собраны только функции, которые отвечают за существование объекта и больше ничего. Сами смотрите. Можно создать, сохранить, восстановить, сделать копию, проверить состоянии и все. Это каркас, который умеет существовать в компьютерной системе, и на который надо добавлять свои функции. При этом этот каркас приспособлен под жизнь вместе с **MFC**. Вы как программист обычно применяете наследования, чтобы добавить к этому классу функциональности. Давайте будем постепенно проводить наследование. Вот самый простой вариант.

#include "stdafx.h"

#include "afxwin.h"

#include "iostream.h"

class CMyObject : public CObject

{

};

void main()

{

CMyObject\* myObj = new CMyObject;

}

Мы класс **CMyObject** как потомок от **CObject**. Но наш класс пока ничего не умеет. Он просто появляется в программе. Дело в том несмотря ни на что он еще мертвый. Все функции, которые определены в нем виртуальные. У них нет реализации. Вот пример кода из **Afx.Inp**:

......

\_AFX\_INLINE void CObject::AssertValid() const

{ /\* no asserts in release builds \*/ }

\_AFX\_INLINE void CObject::Dump(CDumpContext&) const

{ /\* no dumping in release builds \*/ }

......

Нам надо оживить его. Функция за функцией для того, чтобы он смог жить в мире **MFC**. На вопрос с чем связано отсутствие реализаций функций прост. Дело в том, что класс абсолютно абстрактный, так как не имеет данных. Чего сохранять ??? Чего проверять ???? Нечего. Это просто шаблон, который говорит Вам, что надо делать так.

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?214) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?216) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc5.html)  
Автор **Каев Артем**.

## Шаг 216 - DECLARE\_DYNAMIC и IMPLEMENT\_DYNAMIC

Объявление данных макросов в классе позволяет обратиться к информации о классе во время выполнения. То есть эти макросы просто дают возможность во время работы программы иметь информацию о типе класса. Вот их описание:

DECLARE\_DYNAMIC( имя класса )

IMPLEMENT\_DYNAMIC(имя класса, имя класса родителя )

// Имя класса фактическое не заключенного в кавычки.

**DECLARE\_DYNAMIC** в описании класса **IMPLEMENT\_DYNAMIC** в реализации. **IMPLEMENT\_DYNAMIC** заменяется на код **C++** для получения информации времени выполнения. Если вы объявили эти макросы для класса, то можете обращаться к функции **IsKindOf** и макрокоманде **RUNTIME\_CLASS**. Вот выдержки из исходного кода по определению этих макросов.

#define IMPLEMENT\_DYNAMIC(class\_name, base\_class\_name) \

IMPLEMENT\_RUNTIMECLASS(class\_name, base\_class\_name, 0xFFFF, NULL)

#define \_IMPLEMENT\_RUNTIMECLASS(class\_name, base\_class\_name, wSchema, pfnNew) \

AFX\_DATADEF CRuntimeClass class\_name::class##class\_name = { \

#class\_name, sizeof(class class\_name), wSchema, pfnNew, \

RUNTIME\_CLASS(base\_class\_name), NULL }; \

CRuntimeClass\* class\_name::GetRuntimeClass() const \

{ return RUNTIME\_CLASS(class\_name); } \

Если не эти макросы, то Вам пришлось бы все это писать. Можете и сами писать. Но лучше уж макроопредления. Может с их реализацией мы подробнее когда-нибудь разберемся.

Давайте организуем поддержку информации о классе во время выполнения программы для нашего класса.

#include "stdafx.h"

#include "afxwin.h"

#include "iostream.h"

class CMyObject : public CObject

{

protected:

DECLARE\_DYNAMIC(CMyObject)

};

IMPLEMENT\_DYNAMIC(CMyObject,CObject);

void main()

{

CMyObject\* myObj = new CMyObject;

}

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?215) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?217) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc5.html)  
Автор **Каев Артем**.

## Шаг 217 - Проверка принадлежноcти классу IsKindOf

Эта функция может определить какому классу принадлежит объект в памяти компьютера. То есть уже сужествующий. Для пользования этой функцией в определении класса должны в объявлении и реализации класса должны быть макроопределения.

DECLARE\_DYNAMIC

IMPLEMENT\_DYNAMIC

В противном случае при компиляции будут выдаваться ошибки типа:

E:\PROJECT\TestObject\TestObject.cpp(29) : error C2039: 'classCMyObject' : is not a member of 'CMyObject'

E:\PROJECT\TestObject\TestObject.cpp(8) : see declaration of 'CMyObject'

Давайте посмотрим ее определение:

BOOL IsKindOf

(

const CRuntimeClass\* pClass // Указатель на струтуру CRuntimeClass

)

const;

Функция вернет ненулевое значение, если объект соотвествует классу или 0 при несоответствии. Давайте посмотрим пример.

// TestObject.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

#include "afxwin.h"

#include "iostream.h"

class CMyObject : public CObject

{

protected:

DECLARE\_DYNAMIC(CMyObject)

};

IMPLEMENT\_DYNAMIC(CMyObject,CObject);

class CAlternativeObject : public CObject

{

protected:

DECLARE\_DYNAMIC(CAlternativeObject)

};

IMPLEMENT\_DYNAMIC(CAlternativeObject,CObject);

void main()

{

CMyObject\* myObj = new CMyObject;

if (myObj->IsKindOf(RUNTIME\_CLASS(CMyObject)))

cout << "CMyObject Test 1" << endl;

else

cout << "Not CMyObject Test 1" << endl;

if (myObj->IsKindOf(RUNTIME\_CLASS(CAlternativeObject)))

cout << "CMyObject Test 2" << endl;

else

cout << "Not CMyObject Test2" << endl;

}

Мы имеем два класса и с помошью функции **IsKindOf** проверяем какому классу принадлежит объект. Зачем это нужно ??? Ну, например, чтобы узнать есть ли нужная функция в классе. Вы скажете, что знаете, так как сами пишите программу. Представьте, что у Вас есть набор классов в массиве. Как вы разберетесь какой указатель на какой показывает ??? Ну естественно создадите в классе поле **Identify**. Чтобы этого не делать и реализована эта функция.

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?216) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?218) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc5.html)  
Автор **Каев Артем**.

## Шаг 218 - Макрос RUNTIME\_CLASS

Этот класс используется для того, чтобы получить структуру **CRuntimeClass** с именем класса. Эту структуру возвратят только объекты объявленные как:

DECLARE\_DYNAMIC

DECLARE\_DYNCREATE

Описан этот макрос как:

RUNTIME\_CLASS( имя класса )

//Имя класса фактическое не заключенное в кавычки

На место этого макроса компилятор поставит код. Вот такой:

#define RUNTIME\_CLASS(class\_name) \

((CRuntimeClass\*)(&class\_name::class##class\_name))

Модернизация прошлого примера с учетом **RUNTIME\_CLASS** и тем, что он возвращает ссылку на **CRuntimeClass**.

void main()

{

CMyObject\* myObj = new CMyObject;

CRuntimeClass\* pClass = RUNTIME\_CLASS(CMyObject);

if (myObj->IsKindOf(pClass))

cout << "CMyObject Class" << endl;

else

cout << "Not CMyObject Class" << endl;

}

Используя этот макрос объект можно создавать динамически не используя оператора **new**, вот пример. Можно рассматривать это как альтернативный метод создания объекта. Но только в том случае, если в объвлении класса используется макрос **DECLARE\_DYNCREATE**.

#include "stdafx.h"

#include "afxwin.h"

#include "iostream.h"

class CMyObject : public CObject

{

protected:

DECLARE\_DYNCREATE(CMyObject)

};

IMPLEMENT\_DYNCREATE(CMyObject,CObject);

void main()

{

CRuntimeClass\* pClass = RUNTIME\_CLASS(CMyObject);

CObject\* myObj = pClass->CreateObject();

ASSERT(myObj);

if (myObj->IsKindOf(pClass))

cout << "CMyObject Class" << endl;

else

cout << "Not CMyObject Class" << endl;

}

Вот вывод такой. Мы можем создать функцию, которая будет создавать любой объект в зависимости от параметров. Давайте сделаем для понимания.

#include "stdafx.h"

#include "afxwin.h"

#include "iostream.h"

class CMyObject : public CObject

{

protected:

DECLARE\_DYNCREATE(CMyObject)

};

IMPLEMENT\_DYNCREATE(CMyObject,CObject);

class CAlternativeObject : public CObject

{

protected:

DECLARE\_DYNCREATE(CAlternativeObject)

};

IMPLEMENT\_DYNCREATE(CAlternativeObject,CObject);

void CreateAllObject(CRuntimeClass\* pClass);

void main()

{

CreateAllObject(RUNTIME\_CLASS(CMyObject));

CreateAllObject(RUNTIME\_CLASS(CAlternativeObject));

}

void CreateAllObject(CRuntimeClass\* pClass)

{

CObject\* myObj;

myObj= pClass->CreateObject();

ASSERT(myObj);

if (myObj->IsKindOf(RUNTIME\_CLASS(CMyObject)))

cout << "CMyObject Class" << endl;

else

{

if (myObj->IsKindOf(RUNTIME\_CLASS(CAlternativeObject)))

cout << "CAlternativeObject Class" << endl;

else

cout << "Not Info Class" << endl;

}

}

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?217) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?219) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc5.html)  
Автор **Каев Артем**.

## Шаг 219 - IMPLEMENT\_DYNCREATE и DECLARE\_DYNCREATE

Используется для динамического создания объекта. Например, когда Вы читаете объект с диска. Кроме того у Вас появляется возможность создания объекта без использования оператора **new**, а пользуясь функцией **CreateObject** класса **CRuntimeClass**.

DECLARE\_DYNCREATE( имя класса )

IMPLEMENT\_DYNCREATE( имя класса,имя класса родителя )

// Фактическое название без кавычек

В описании класса указываем макрокоманду **DECLARE\_DYNCREATE**, а в реализации **IMPLEMENT\_DYNCREATE**. Вот так:

class CMyObject : public CObject

{

protected:

DECLARE\_DYNCREATE(CMyObject)

};

IMPLEMENT\_DYNCREATE(CMyObject,CObject);

Так как для динамического создания необходима информация о типе класса, то макросы **DECLARE\_DYNAMIC** и **IMPLEMENT\_DYNAMIC** определяются внутри **DECLARE\_DYNCREATE** и реализации **IMPLEMENT\_DYNCREATE** - это абсолютно логическое утверждение подтверждено и в исходном коде.

#define IMPLEMENT\_DYNAMIC(class\_name, base\_class\_name) \

IMPLEMENT\_RUNTIMECLASS(class\_name, base\_class\_name, 0xFFFF, NULL)

#define IMPLEMENT\_DYNCREATE(class\_name, base\_class\_name) \

CObject\* PASCAL class\_name::CreateObject() \

{ return new class\_name; } \

IMPLEMENT\_RUNTIMECLASS(class\_name, base\_class\_name, 0xFFFF, \

class\_name::CreateObject)

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?218) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?220) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc5.html)  
Автор **Каев Артем**.

## Шаг 220 - Обобщение шагов 215-219

Посмотрев предыдущие шаги Вы могли подумать, что это полный ад. Куча макросов неизвестно зачем и почему. Для полного понимания всего сказанного в этих шагах мне кажется нужно думать так.

Любое приложение или структура данных состоит из многих объектов. Эти объекты нужно уметь объединять в массивы или коллекции. Ну, например, Окно диалоговой панели вмещает в себя много элементов разных типов: всяческие кнопки, элементы редактирования, статический текст и так далее. Вам нужно работать со всеми элементами сразу. Например, запустив механизм **DDX** или перерисовывая экран. То есть должен быть набор указателей на эти объекты. В **C++** есть проверка типов и поэтому все становится намного сложнее. Давайте рассмотрим теоритический массив. Вот такой

x\_type\* p1

x\_type\* p2

......

То есть нам нужен универсальный указатель на любой класс. Он есть и назван **CRuntimeClass**. Вот теперь какой наш массив

CRuntimeClass\* p1;

CRuntimeClass\* p2

......

Теперь в этот массив нужно помещать классы. Можно поступить просто указывать ссылку на созданные нами объекты и самим следить за расположением их в памяти и удалением. Это не удобно. Кроме того есть набор классов логически связанных, которые и появляются вместе и изчезают, ну например, шаблон приложения.

CSingleDocTemplate\* pDocTemplate;

pDocTemplate = new CSingleDocTemplate(

IDR\_MAINFRAME,

RUNTIME\_CLASS(CMy11Doc),

RUNTIME\_CLASS(CMainFrame),

RUNTIME\_CLASS(CMy11View));

AddDocTemplate(pDocTemplate);

Соответственно класс указателя **CRuntimeClass** должен уметь сам строить классы заменяя **new** и он это умеет, т.к. у него есть **CreateObject**. Только построение объекта делится на два шага: первое - это получение информации о нем и второе - непосредсвенное построение. В стандарте **C++** нет механизма для динамического определения типа класса. Значит этот механизм будет надстройкой. Значит классы, которые будут динамически создаваться надо специально определить. Так вот добавляя в класс **DECLARE\_DYNAMIC** мы создаем в классе информацию о его типе. Все. Теперь мы можем определить тип класса. Для этого есть функция **IsKindOf**. Имея информацию о типе можно говорить о создании. Но создание класса на основе информации о типе тоже не является стандартом. Для того, чтобы класс можно было создавать динамически надо ему дать эту возможность. Это мы делаем добавляя макрос **DECLARE\_DYNCREATE**. Применяя второй макрос мы автоматически применяем первый, так как без информации объект нельзя создать.

ИМЯ -> CRuntimeClass RUNTIME\_CLASS(имя)

CRuntimeClass -> ИМЯ IsKindOf(CRuntimeClass)

// Достаточно DECLARE\_DYNAMIC

Создать объект CRuntimeClass->CreateObject

// ТОлько при DECLARE\_DYNCREATE

Понятно я объяснил причины или нет ? Незнаю. Давайте попробую сказать коротко. Возможности описанные в шагах 215-219 созданы для того, чтобы в одном массиве или коллекции можно было хранить объекты разных типов.

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?219) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?221) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc5.html)  
Автор **Каев Артем**.

## Шаг 221 - IsSerializable

Если объект может быть преобразован в последовательную форму (сохранен на диск), тогда значение отличное от нуля, иначе ноль.

BOOL IsSerializable( ) const;

Для того, чтобы класс мог быть преобразован в последовательную форму в объявлении класса должно быть объявлено **DECLARE\_SERIAL**, а в реализации **IMPLEMENT\_SERIAL** смотрим пример:

#include "stdafx.h"

#include "afxwin.h"

#include "iostream.h"

class CMyObject : public CObject

{

protected:

DECLARE\_DYNCREATE(CMyObject)

};

IMPLEMENT\_DYNCREATE(CMyObject,CObject);

class CAlternativeObject : public CObject

{

protected:

DECLARE\_SERIAL(CAlternativeObject);

};

IMPLEMENT\_SERIAL(CAlternativeObject,CObject,1);

void CreateAllObject(CRuntimeClass\* pClass);

void main()

{

CMyObject cm;

CAlternativeObject ca;

if (cm.IsSerializable())

cout << "CMyObject serialize" << endl;

if (ca.IsSerializable())

cout << "CAlternativeObject serialize" << endl;

}

В классе **CAlternativeObject** объявлен механизм сохранения и восстановления объекта с помощью макросов, поэтому **IsSerializable** и даем возможность нам вывести надпись.

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?220) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?222) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc5.html)  
Автор **Каев Артем**.

## Шаг 222 - DECLARE\_SERIAL и IMPLEMENT\_SERIAL

IMPLEMENT\_SERIAL( имя класса, базовый класс, версия )

DECLARE\_SERIAL( имя класса )

// имя класса физическое не в кавычках

Эти макроопределения генерируют код необходимый для сохранения и восстановления класса на диск. Продолжение нашей цепочки:

IMPLEMENT\_DYNAMIC

IMPLEMENT\_DYCREATE

IMPLEMENT\_\_SERIAL

Все логично, чтобы восстанавливать и сохранять класс надо знать о нем информацию и уметь его динамически создавать. Это следующая ступень развития. А вот для убедительности код.

#define IMPLEMENT\_SERIAL(class\_name, base\_class\_name, wSchema) \

CObject\* PASCAL class\_name::CreateObject() \

{ return new class\_name; } \

**\_IMPLEMENT\_RUNTIMECLASS(class\_name, base\_class\_name, wSchema, \**

class\_name::CreateObject) \

AFX\_CLASSINIT \_init\_##class\_name(RUNTIME\_CLASS(class\_name)); \

CArchive& AFXAPI operator>>(CArchive& ar, class\_name\* &pOb) \

{ pOb = (class\_name\*) ar.ReadObject(RUNTIME\_CLASS(class\_name)); \

return ar; } \

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?221) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?223) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc5.html)  
Автор **Каев Артем**.

## Шаг 223 - AssertValid

virtual void AssertValid( ) const;

Отладочная функция. С помощью нее проверяют объект на допустимость значения. Если значения в объекте не допустимые, то вызывается ошибка с номером, в которой указываются номер строки и имя файла, т.е. место совершения ошибки. Для того, чтобы проверить наш объект нам нужно эту функцию перегрузить, написав свой проверочный код. И при этом обычно вызвают одноименный метод базового класса.

#include "afxwin.h"

#include "iostream.h"

class CMyObject : public CObject

{

public:

void AssertValid() const;

protected:

DECLARE\_DYNCREATE(CMyObject)

private:

int x;

int y;

};

IMPLEMENT\_DYNCREATE(CMyObject,CObject);

void CMyObject::AssertValid() const

{

CObject::AssertValid();

ASSERT(x==100);

ASSERT(y==100);

}

class CAlternativeObject : public CObject

{

protected:

DECLARE\_SERIAL(CAlternativeObject);

};

IMPLEMENT\_SERIAL(CAlternativeObject,CObject,1);

void main()

{

CMyObject cm;

}

В данном примере мы создали функцию, которая будет проверять, чтобы данные в классе **x** и **y** были равны 100. Так как функция **const**, то изменить значения внутри нее нельзя. Можно только проверить. При этом как вы заметили я не вызывал исключения, а вызывая макрос **ASSERT** для проверки значений и это правильно. Самому вызывать исключения внутри этой функции нельзя. Проверять можно не только значения, но и указатели, например, если в классе есть ссылка на другие объекты, что эти ссылки действительны. Давайте вызовем ее сразу после создания объекта:

void main()

{

CMyObject cm;

cm.AssertValid();

}

**AssertValid - это отладочная функция и применять вы ее можете только в версии DEBUG, во всех остальных она просто изчезнет. Поэтому грамотно проверять конфигурацию компиляции и использовать эту функцию в режиме отладки. Вот так.**

**......**

**class CMyObject : public CObject**

**{**

**public:**

**#ifdef \_DEBUG**

**void AssertValid() const;**

**#endif**

**protected:**

**DECLARE\_DYNCREATE(CMyObject)**

**private:**

**int x;**

**int y;**

**};**

**IMPLEMENT\_DYNCREATE(CMyObject,CObject);**

**#ifdef \_DEBUG**

**void CMyObject::AssertValid() const**

**{**

**CObject::AssertValid();**

**ASSERT(x==100);**

**ASSERT(y==100);**

**}**

**#endif**

**......**

**void main()**

**{**

**CMyObject cm;**

**#ifdef \_DEBUG**

**cm.AssertValid();**

**#endif**

**}**

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?222) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?224) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc5.html)  
Автор **Каев Артем**.

## Шаг 224 - Dump

Выводим пользовательские классы в отладочный контекст **CDumpContext**:

virtual void Dump( CDumpContext& dc ) const

Создавая свой класс-потомок от **CObject** Вы должны перегрузить функцию **Dump** для вывода Ваших данных описанных в классе. В ходе вывода в отладочный поток выводится и имя класса при условии, что к классу добавлена информация о нем с помощью

IMPLEMENT\_DYNAMIC

IMPLEMENT\_SERIAL

Это отладочная функция и использоваться она должна только в версии **Debug** следите за этим с помощью **#ifdef \_DEBUG #endif** директивами компиляции. Кроме того эта функция **const**, поэтому данные менять нельзя. Кроме того вы должны использовать **F5** для запуска в режиме отладки. Давайте посмотрим пример:

#include "stdafx.h"

#include "afxwin.h"

#include "iostream.h"

class CMyObject : public CObject

{

public:

#ifdef \_DEBUG

void AssertValid() const;

void Dump(CDumpContext& dc) const;

#endif

protected:

DECLARE\_DYNCREATE(CMyObject)

private:

int x;

int y;

};

IMPLEMENT\_DYNCREATE(CMyObject,CObject);

#ifdef \_DEBUG

void CMyObject::AssertValid() const

{

CObject::AssertValid();

ASSERT(y==100);

}

void CMyObject::Dump( CDumpContext& dc ) const

{

CObject::Dump(dc);

dc << "Hello Dump ";

}

#endif

class CAlternativeObject : public CObject

{

protected:

DECLARE\_SERIAL(CAlternativeObject);

};

IMPLEMENT\_SERIAL(CAlternativeObject,CObject,1);

void main()

{

CMyObject cm;

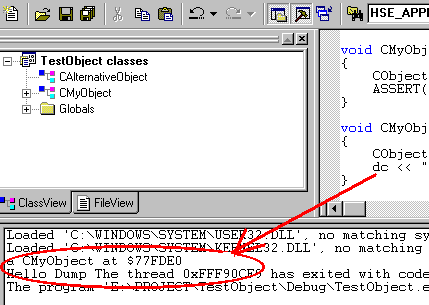
#ifdef \_DEBUG

cm.Dump(afxDump) ;

#endif

}

Я использовал стандартный контекст **afxDump**, поэтому весь вывод пойдет в окно отладки. Вот так:



[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?223) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?225) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc5.html)  
Автор **Каев Артем**.

## Шаг 225 - Serialize

Производит сохранение и восстановление данных:

virtual void Serialize( CArchive& ar );

throw( CMemoryException );

throw( CArchiveException );

throw( CFileException );

Объект **ar** типа **CArchive**, с которым производятся операции сохранения и восстановления.

При создании своего класса Вы перегружаете функцию **Seliarize** и в новой функции не забываете вызвать функцию предка.

class CMyObject : public CObject

{

public:

#ifdef \_DEBUG

void AssertValid() const;

void Dump(CDumpContext& dc) const;

#endif

void Serialize( CArchive& ar );

protected:

DECLARE\_SERIAL(CMyObject);

private:

int x;

int y;

};

void CMyObject::Serialize( CArchive& ar )

{

CObject::Serialize( ar );

if( ar.IsStoring() )

ar << x << y;

else

ar >> x >> y;

}

IMPLEMENT\_SERIAL(CMyObject,CObject,1);

Для использования возможностей сохранения Вы должны использовать следующие макроопределения

DECLARE\_SERIAL

IMPLEMENT\_SERIAL

Кроме того у класса должен быть конструктор без параметров. Для определения действия сохранения или удаления используется метод **IsStoring** класса **CArhive**.

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?224) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?226) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc5.html)  
Автор **Каев Артем**.

## Шаг 228 - DEBUG\_NEW

Макроопределение **DEBUG\_NEW** предназначенно для выявления утечки памяти. Этого вопроса мы уже не раз касались. Смотрите ["Шаг 65 - Ловим утечку памяти"](http://www.firststeps.ru/mfc/steps/r.php?65) и ["Подробности - Утечка памяти"](http://www.firststeps.ru/mfc/detail/r.php?21). Здесь Вы можете узнать, что это за понятие "утечка памяти". Мы же с Вами займемся стандартным решением на основе **DEBUG\_NEW**.

#define new DEBUG\_NEW

В результате переопределения **new** мы добавляем к обычному оператору **new** информацию о файле и строке. Вот как это выглядит в исходном коде.

#define DEBUG\_NEW new(THIS\_FILE, \_\_LINE\_\_)

Естественно, что пользоваться этим нужно в отладочной версии и поэтому разумно делать так четко подчеркивая, что это отладочный Вариант. Это делать не обязательно, так как в версии **RELEASE** все равно заменится на обычный **new**, но зато понятно. Это только для отладки !!!

#ifdef \_DEBUG

#define new DEBUG\_NEW

#endif

Сформировав так информацию ее можно получить вызвав функцию **DumpAllObjectsSince**, которая вернет информацию о каждом файле и строке где распределена память с использованием **DEBUG\_NEW**. Вы конечно и без **DEBUG\_NEW** можете получить информацию о утечке, но только более сжатую о чем было ранее написано в шагах. Давайте посмотрим.

#include "stdafx.h"

#include "afxwin.h"

#include "iostream.h"

#ifdef \_DEBUG

#define new DEBUG\_NEW

#endif

class CMyObject : public CObject

{

public:

#ifdef \_DEBUG

void AssertValid() const;

void Dump(CDumpContext& dc) const;

#endif

void Serialize( CArchive& ar );

protected:

DECLARE\_SERIAL(CMyObject);

private:

int x;

int y;

};

void CMyObject::Serialize( CArchive& ar )

{

CObject::Serialize( ar );

if( ar.IsStoring() )

ar << x;

else

ar >> x;

}

IMPLEMENT\_SERIAL(CMyObject,CObject,1);

#ifdef \_DEBUG

void CMyObject::AssertValid() const

{

CObject::AssertValid();

ASSERT(y==100);

}

void CMyObject::Dump( CDumpContext& dc ) const

{

CObject::Dump(dc);

dc << "Hello Dump ";

}

#endif

void main()

{

CMyObject \*cm;

cm=new CMyObject;

}

В результате запуска в отладочном окне мы видим вот что.

......

Detected memory leaks! // Утечка памяти !!!

Dumping objects -> // в файле TestObject.cpp на строке 67 выделена память которая не удалена

E:\PROJECT\TestObject\TestObject.cpp(67) : {19} client block at 0x008A0E90, subtype 0, 12 bytes long.

a CMyObject object at $008A0E90, 12 bytes long //

Object dump complete.

......

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?227) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?229) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc5.html)  
Автор **Каев Артем**.

## Шаг 268 - Поддержка WinSock в MFC

**MFC** поддерживает **WinSock** через два класса:

CAsyncSocket

CSocket

**CAsyncSocket** - это класс, который инкапсулирует возможности работы с **WinSock API**.

**CSocket** - это более высокоуровневый класс, который происходит от **CAsyncSocket**. И естественно обеспечивает работу более высокого уровня. Он обеспечивает ряд функций, например, функцию блокирования, которая необходима для работы с классом **CArhive**. Описание классов находится в файле:

#include "afxsock.h"

Для обеспечения поддержки работы **WinSock** и классов **MFC** необходимо произвести инициализацию в **InitInstance(CWinApp::InitInstance)** класса приложения вызвав функцию **AfxSocketInit**:

BOOL AfxSocketInit( WSADATA\* lpwsaData = NULL );

При инициализации можно передать указатель на структуру **WSADATA**, которая будет заполнена в ходе инициализации. При удачной инициализации возврат этой функции отличен от нуля, иначе ноль.

struct WSAData

{

WORD wVersion; // Версия Windows Sockets DLL

WORD wHighVersion; // Версия Windows Sockets DLL

char szDescription[WSADESCRIPTION\_LEN+1]; // строка описания

char szSystemStatus[WSASYSSTATUS\_LEN+1]; // строка описания

unsigned short iMaxSockets; // максимальное число доступных сокетов

unsigned short iMaxUdpDg; // размер самой большой датаграммы (UDP)

char FAR \* lpVendorInfo; // указатель на специальную структуру

// которая не является частью стандарта Windows Sockets

};

Внимание **ClassWizard** не умеет порождать классы от данных. Это придется делать руками.

[Предыдущий Шаг](http://www.firststeps.ru/mfc/steps/r.php?267) | [Следующий Шаг](http://www.firststeps.ru/mfc/steps/r.php?269) | [Оглавление](http://www.firststeps.ru/mfc/steps/mfc6.html)  
Автор **Каев Артем**.

1. шаблонов может быть несколько [↑](#endnote-ref-1)
2. ClassWizard [↑](#endnote-ref-2)
3. Events [↑](#endnote-ref-3)
4. инициализацию нужно сделать в методе OnNewDocument класса документа, не в конструкторе [↑](#endnote-ref-4)