Министерство образования Республики Беларусь
Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра экономической информатики

ЛАБОРАТОРНЫЙ ПРАКТИКУМ

по курсу **«Визуальные средства разработки приложений»**

для студентов специальности 40 01 02-02

“Информационные системы и технологии в экономике”

Минск 2002

УДК 002.5 +681.306(075.8)
ББК 65.39 я 73
Л 12

Авторы:
Комличенко Виталий Николаевич,
Живицкая Елена Николаевна,
Соколов Сергей Александрович,
Онищук Вениамин Викеньтьевич,
Унучек Евгений Николаевич

Лабораторный практикум по курсу «Визуальные средства разработки прило-
Л12 жений» для студентов специальности 40 01 02-02 “Информационные системы и технологии в экономике” / В.Н. Комличенко, Е.Н. Живицкая, С.А. Соколов и др. -Мн.: БГУИР, 2002.-89с.: ил.

ISBN 985-444-362-0

В работе представлены: основные темы лекционного курса «Основы объектно - ориентированного программирования», методические рекомендации, примеры программной реализации типового задания; список используемой литературы и задания для лабораторных работ.

Коллектив авторов выражает благодарность студентке Синявской О.А. за помощь при составлении лабораторного практикума

УДК 002.5+681.306 (075.8)
ББК 65.39 я 73

ISBN 985-444-362-0

ЛАБОРАТОРНАЯ РАБОТА №1
«СОЗДАНИЕ SDI ПРИЛОЖЕНИЯ»

Цели работы:

1) ознакомиться с процессом создания Win32-приложения с помощью Microsoft Visual C++;

2) узнать возможности классов, используемых при создании Win32-приложения.

Задачи:

1) создание простейшего Win32-приложения;

2) разработка программы MiniDraw рисования прямых линий с возможностями удаления последней из нарисованных и всех линий рисунка.

Методические указания

При разработке программ в VC++ и MFC обычно предполагается использование архитектуры «документ/представление». Названная архитектура позволяет связать данные с их представлением пользователю на экране. Логическое разделение данных программы и методов их визуального представления позволяет отображать документы разными способами, связав документ с несколькими представлениями (например, в Microsoft Word доступны три вида одного и того же документа: обычный, разметка страниц и структура документа). Кроме того, в этом случае изменения, вносимые в документ в одном представлении, отображаются во всех других. В разработке приложений можно использовать как готовые представления (основанные на элементах управления, таких, как, деревья просмотра, списки, предоставляемых MFC), так и создавать собственные, перегружая функцию отображения, обработчики сообщений от клавиатуры, мыши и пунктов меню.

Архитектура «документ/представление» предоставляет множество возможностей для работы с документом. Так, AppWizard способен генерировать каркас приложения, реализующий документы и представления средствами классов, производных от классов CDocument и CView (классы документа и представления).

Класс документа в MFC отвечает за хранение данных, а также за их загрузку из файлов на диске; содержит функции, позволяющие другим классам (в частности, классу представления) получать или изменять данные таким образом, чтобы они были доступны для просмотра и редактирования. Этот класс должен обрабатывать команды меню, непосредственно воздействующие на данные документа.

Представление – это часть программы, использующая библиотеку MFC для управления окном просмотра, обработки информации, вводимой пользователем, и отображения документа в окне.

После того как AppWizard создаст основной шаблон программы (приложение 1), в класс представления необходимо добавить код для отслеживания действий мыши и рисования прямых линий в окне представления. Для создания

обработчиков сообщений мыши и настройки окна представления используется мастер ClassWizard, для изменения меню программы – редактор ресурсов.

В первую очередь необходимо определить класс для сохранения данных о каждой созданной линии. В начало файла заголовка MiniDrawDoc.h перед определением класса CMiniDrawDoc добавьте следующее определение класса CLine:

```
class CLine : public CObject
{
protected :
   int m_Xl, m_Yl, m_X2 , m_Y2;
public :
   CLine (int XI, int Yl, int X2, int Y2)
   {
      m_Xl = XI; m_Yl = Yl; m_X2 = X2; m_Y2 = Y2;
   }
   void Draw (CDC *PDC) ;
};
```

Переменные m_X1 и m_Yl класса CLine сохраняют координаты одного конца прямой линии, m_X2 и m_Y2 - другого. Класс CLine содержит также функцию Draw для рисования линии.

В эту функцию передается указатель на объект класса CDC. Он необходим для вызова рисующих функций контекста устройства. Класс CDC инкапсулирует функции инициализации контекста устройства, а также функции рисования.

В дальнейшем мы узнаем, почему класс CLine порождается от CObject.

Теперь добавим в класс CMiniDrawDoc требуемые члены, вводя в начало определения класса следующие операторы:

```
class CMiniDrawDoc : public CDocument
{
   protected :
      CTypedPtrArray<CObArray , CLine*> m_LineArray;
   public :
      void AddLine (int XI, int Yl, int X2, int Y2) ;
      CLine *GetLine (int Index) ;
         int GetNumLines () ;
      // остальные определения класса CminiDrawDoc…
}
```

Примечание. В файле MiniDrawDoc.h надо подключить файл #include "Afxtempl.h"

Новая переменная m_LineArray – это экземпляр шаблона CTypedPtrArray. Класс CTypedPtrArray генерирует семейство классов, каждый из которых является производным от класса, заданного в первом параметре шаблона (им может быть CObArray или CPtrArray). Каждый из этих классов предназначен для хранения переменных класса, описанных вторым параметром шаблона.

ром шаблона. Таким образом, переменная m_LineArray является объектом класса, порожденного от класса CObArray и сохраняющего указатели на объекты класса CLine.

Класс CObAггay – это один из классов коллекций общего назначения в библиотеке MFC, используемый для сохранения групп переменных или объектов. Экземпляр класса CObAггay хранит множество указателей на объекты класса CObject (или любого класса, порожденного от CObject) в структуре данных, подобной массиву. CObject – это MFC-класс, от которого прямо или косвенно порождаются практически все остальные классы. Однако вместо использования экземпляра класса общего назначения CObAггay программа MiniDraw использует шаблон CTypedPtrArray, спроектированный специально для хранения объектов класса CLine. Это позволяет компилятору выполнять более интенсивный контроль соответствия типов данных, уменьшать число ошибок и сокращать число операций приведения типов при использовании объектов класса.

Для использования шаблона класса CTypedPtrArray (или любого из шаблонов MFC-классов) в программу нужно включить файл заголовков MFC Afxtempl.h. Чтобы сделать этот файл доступным для любого заголовочного или исходного файла в проекте MiniDraw, его можно включить в стандартный файл заголовков StdAfx.h следующим образом:

```
#include<afxwin.h>   // стандартные компоненты MFC
#include<afxext.h>   // расширения библиотеки MFC
#include<afxdtctl.h> // поддержка общих элементов
                     //управления для Internet Explorer 4
#include <afxtempl.h>  // шаблоны библиотеки MFC
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h> //поддержка общих элементов
                    //управления для Windows 95
#endif // _AFX_NO_AFXCMN_SUPPORT
```

В MiniDrawDoc переменная m_LineArray используется для хранения указателя на каждый объект класса CLine, сохраняющий информацию о линии. Функции класса AddLine, GetLine и GetNumLines предоставляют доступ к информации о линии, хранящейся в массиве m_LineArray (другие классы не имеют к нему прямого доступа, так как переменная m_LineArray является защищенной).

Теперь в конце файла реализации документа MiniDrawDoc.cpp введите определение (код) функции CLine::Draw:

```
void CLine::Draw(CDC *PDC)
{
    PDC->MoveTo(m_Xl, m_Yl);
    PDC->LineTo(m_X2, m_Y2);
}
```

Чтобы создать линию по координатам, сохраненным в текущем объекте, функция Draw вызывает две функции класса CDC – MoveTo и LineTo. Далее в

конце файла MiniDrawDoc.cpp добавьте определения функций AddLine, GetLine и GetNumLines класса CminiDrawDoc:

```
void CMiniDrawDoc::AddLine(int X1, int Y1, int X2, int Y2)
{
    CLine *pLine=new CLine(X1, Y1, X2, Y2);
    m_LineArray.Add(pLine);
}
CLine* CMiniDrawDoc::GetLine(int Index)
{
    if(Index<0||Index>m_LineArray.GetUpperBound ())
        return 0;
    return m_LineArray.GetAt(Index);
}
int CMiniDrawDoc::GetNumLines()
{
    return m_LineArray.GetSize();
}
```

Чтобы добавить указатель объекта в коллекцию указателей на класс CLine, сохраненных в массиве m_LineArray, функция AddLine создает новый объект класса CLine и вызывает функцию Add класса CObArray.

Обратите внимание: указатели, сохраненные в массиве m_LineArray, индексируются. Первый добавленный указатель имеет индекс 0, второй 1 и т.д. Функция GetLine возвращает указатель с индексом, содержащимся в переданном параметре. Вначале она контролирует попадание индекса в допустимый интервал значений. Функция GetUpperBound класса CObArray возвращает наибольший допустимый индекс, т.е. индекс последнего добавленного указателя. Далее функция GetLine возвращает соответствующий указатель класса CLine, получаемый в результате вызова функции GetAt класса CTypedPtrArray.

Наконец, функция GetNumLines возвращает количество объектов класса CLine, сохраненных в переменной m_LineArray. Для этого вызывается функция GetSize класса CObArray. Как вы увидите вскоре, функции AddLine, GetLine и GetNumLines вызываются функциями-членами класса представления.

Добавим в класс представления несколько переменных: m_className, m_Dragging, m_HCross, m_PointOld и m_PointOrigin. Для этого откройте файл MiniDraw.h и добавьте выражения, выделенные полужирным шрифтом, в начало определения класса CminiDrawView:

```
class CMiniDrawView : public View
{
 protected:
        CString m_ClassName;
        int m_Dragging;
        HCURSOR m_HCross;
```

```
        CPoint m_PointOld;
        CPoint m_PointOrigin;
}
```

Назначение этих элементов описано ниже.

Добавьте в конструктор класса CMiniDrawView в файле код инициализации переменных m_Dragging и m_Hcross:

```
        // Конструктор класса CminiDrawView
        CMiniDrawView::CMiniDrawView()
{
        // TODO: Здесь добавьте код конструктора
        m_Dragging = 0;
        m_HCross=AfxGetApp()->LoadStandardCursor(IDC_CROSS);
}
```

Переменная m_HCross хранит дескриптор указателя мыши, отображаемого программой, когда тот находится внутри окна представления. Функция AfxGetApp возвращает указатель на объект класса приложения (класс CMiniDrawApp, порожденный от класса CWinApp), где он используется для вызова функции LoadStandardCursor класса CWinApp. Эта функция при получении идентификатора IDC_CROSS возвращает дескриптор стандартного крестообразного указателя. Ниже приведены значения, которые можно передать в функцию LoadStandardCursor для получения дескрипторов других стандартных указателей:

IDC_ARROW……..…….... стандартный указатель-стрелка
IDC_CROSS……...……..…….перекрестье, используемое для выбора
IDC_IBEAM……..……...указатель для редактирования текста
IDC_SIZEALL…….……указатель из четырех стрелок для изменения размеров окна
IDC_SIZENESW……..……двунаправленная стрелка, указывающая на северо-восток и юго-запад
IDC_SIZENS ……...……двунаправленная стрелка, указывающая на север и юг
IDC_SIZENWSE…….двунаправленная стрелка, указывающая на северо-запад и юго-восток
IDC_SIZEWE……..……..двунаправленная стрелка, указывающая на запад и восток
IDC_UPARROW..……вертикальная стрелка
IDC_WAIT……«песочные часы», используемые при длительном выполнении задачи

AfxGetApp – это глобальная функция библиотеки MFC, которая не является членом класса и содержит глобальные функции, начинающиеся с префикса Afx.

Чтобы пользователь мог рисовать линии внутри окна представления с помощью мыши, программа должна реагировать на события, происходящие внутри этого окна. Для обработки сообщения, передаваемого мышью, в класс представления необходимо добавить функцию обработки сообщений.

Обработка сообщений. С каждым окном в графической программе связана функция, называемая процедурой окна. Когда происходит некоторое событие, операционная система вызывает эту функцию, передавая ей идентификатор происшедшего события и данные для его обработки. Подобный процесс называется передачей сообщения окну.

Процедура окна с помощью библиотеки MFC создается автоматически. Если необходима собственная обработка сообщения, то создается функция обработки сообщения, являющаяся членом класса управления окном. Для определения обработчика сообщения можно воспользоваться ClassWizard, как описано ниже.

Например, если указатель находится внутри окна представления, то при нажатии левой кнопки мыши передается идентификатор WM_LBUTTONDOWN. Чтобы предусмотреть собственную обработку этого сообщения, используйте мастер ClassWizard для создания функции класса представления, обрабатывающей данное сообщение.

Командные сообщения. MFC обеспечивает специальную обработку сообщений, генерируемых объектами пользовательского интерфейса, стандартными элементами, поддерживаемыми библиотекой MFC: меню, комбинации клавиш, кнопки панелей инструментов, строки состояния, элементы управления диалоговых окон (термин “объект” в данном случае относится не к объектам языка C++). Сообщения, генерируемые объектами пользовательского интерфейса, называют командными сообщениями. Каждый раз, когда пользователь выбирает объект интерфейса или когда один из этих объектов необходимо обновить, объект передает командное сообщение главному окну. Однако библиотека MFC сразу направляет сообщение объекту окна представления. Если он не имеет нужного обработчика, библиотека MFC направляет сообщение объекту документа. Если же объект документа не содержит обработчик, библиотека MFC направляет сообщение объекту главного окна программы. Если главное окно также не располагает обработчиком, сообщение направляется объекту приложения. Наконец, если объект приложения не обеспечивает обработку, то сообщение обрабатывается стандартным образом.

Таким образом, библиотека MFC расширяет основной механизм сообщений, чтобы командные сообщения обрабатывались не только объектами, управляющими ими, но и любыми другими объектами приложения. Каждый из них принадлежит классу, прямо или косвенно порожденному от класса CCmdTarget, реализующего механизм передачи сообщений.

Важной особенностью такого механизма является то, что программа может обрабатывать нужное сообщение внутри наиболее подходящего для этого класса. Например, в программе, созданной мастером AppWizard, команда Exit в меню File

обрабатывается классом приложения, так как эта команда воздействует на приложение в целом. С другой стороны, команда Save в меню File обрабатывается классом Документа, так как этот класс отвечает за хранение и запись данных документа.

Далее вы узнаете, как добавлять команды меню и другие объекты интерфейса, определять, какому именно классу следует обрабатывать конкретное сообщение и как создавать обработчики сообщений.

Функция OnLButtonDown. Следующая задача состоит в определении обработчика сообщения WM_LBUTTONDOWN, которое передается при каждом нажатии левой кнопки мыши, если указатель находится внутри окна представления**.** Для определения функции выполните следующие действия:

1. Выберите в меню View команду ClassWizard... или нажмите Ctrl+W. Откроется диалоговое окно мастера ClassWizard.

2. В диалоговом окне мастера ClassWizard откройте вкладку Message Maps, позволяющую определить обработчик сообщений.

3. В списке Class name выберите класс CMiniDrawView, чтобы добавить функцию обработки сообщений в класс представления.

4. В списке Object IDs выберите пункт CMiniDrawView. Выбор класса CMiniDrawView задает функцию-член для обработки любого уведомляющего сообщения, переданного окну представления, что позволяет переопределить одну из виртуальных функций, которую класс CMiniDrawView наследует от CView и других базовых классов библиотеки MFC. Другие пункты в списке Object IDs определяют сообщения, поступающие от объектов интерфейса (команды меню). Выбор одного из этих пунктов задает обработчик сообщений, поступающих от выбранного объекта.

5. В списке сообщений Messages выберите идентификатор WM_LBUTTONDOWN, обрабатываемый определяемой функцией. Messages содержит идентификаторы всех уведомляющих сообщений, которые передаются окну представления (идентификаторы сообщений - это константы, записанные заглавными буквами и начинающиеся с префикса WM_). Список Messages также содержит имена всех виртуальных функций, принадлежащих классу CView. Их можно выбирать, чтобы переопределять стандартные функции. Обратите внимание: при выборе конкретного идентификатора сообщения или виртуальной функции внизу диалогового окна ClassWizard появляется соответствующее краткое описание.

6. Щелкните на кнопке AddFunction. Теперь ClassWizard создает шаблон обработчика сообщения с именем OnLButtonDown. В частности, ClassWizard объявляет функцию в определении класса CMiniDrawView в файле MiniDrawView.h, вносит ее определение в файл MiniDrawView.cpp и добавляет функцию в схему сообщений класса. Теперь имя сообщения в списке Messages отображается полужирным шрифтом - для него задан обработчик. Заметьте: имя функции и сообщения добавлено в список функций Member functions. Пункт, отмеченный буквой "W", означает обработчик сообщения Windows; "V" - виртуальную функцию (рис. 1.1).

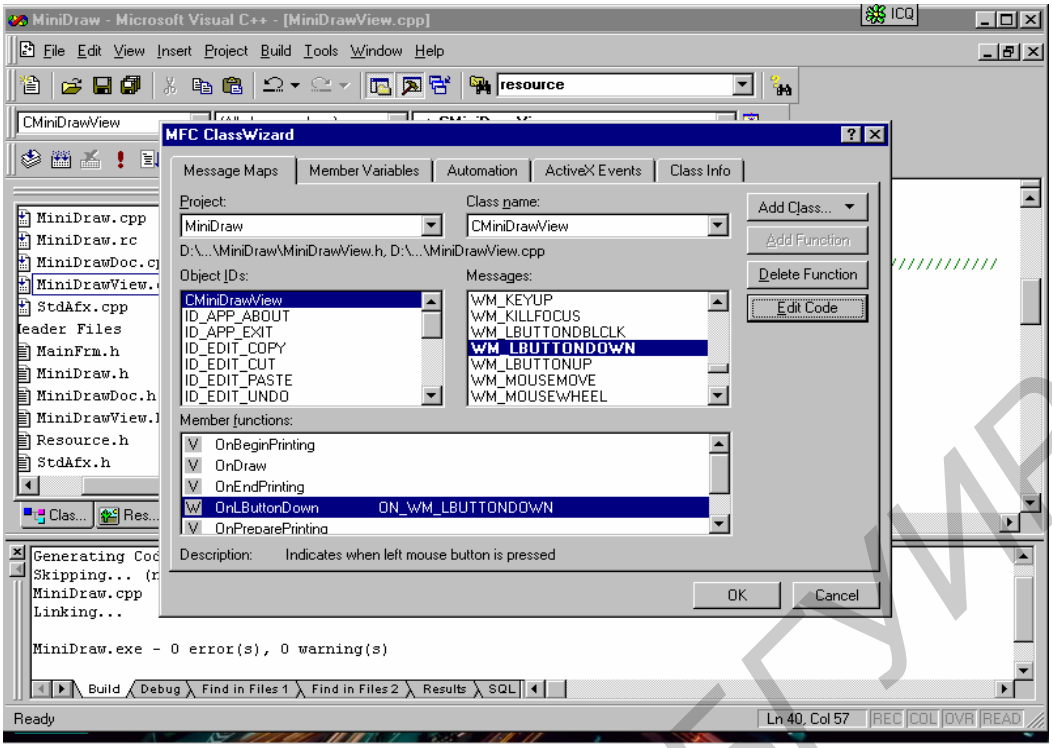

Рис.1.1 *Диалоговое окно для создания обработчиков событий и переопределения виртуальных функций*

7. Щелкните на Edit Code – диалоговое окно ClassWizard закроется. Мастер ClassWizard откроет файл MiniDrawView.cpp (если он еще не открыт) и отобразит только что созданную функцию OnLButtonDown, чтобы можно было добавить ее код.

8. Добавьте в функцию OnLButtonDown операторы, выделенные полужирным шрифтом:

```
void CMiniDrawView::OnLButtonDown(UINT nFlags, CPoint point)
{
        // TODO: Здесь добавьте собственный код обработчика
        // и/или вызов стандартного обработчика
    m_PointOld=point;
    SetCapture();
    m_Dragging=1;
    RECT Rect;
        //CONST RECT *Rect1;
    GetClientRect(&Rect);
    ClientToScreen(&Rect);
    ::ClipCursor(&Rect);

    CView::OnLButtonDown(nFlags, point)
}
```

Если указатель находится в окне представления, то после нажатия левой кнопки мыши управление будет передано функции OnLButtonDown, а параметр *point* будет содержать текущую позицию указателя. Эта позиция сохраняется в переменных m_PointOrigin и m_PointOld. Переменная m_PointOrigin хранит координаты начальной точки линии. Переменная m_PointOld, как вы скоро увидите, используется другими обработчиками сообщений для получения информации о положении указателя мыши в момент предыдущего сообщения.

Мастер ClassWizard добавляет строку в функцию OnLButtonDown, вызывающую функцию OnLButtonDown, определенную в базовом классе. Это делается для того, чтобы базовый класс мог выполнять стандартную обработку сообщений.

Вызов функции SetCapture класса CWnd приводит к захвату мыши, и все последующие ее сообщения передаются в окно представления, пока захват не будет отменен. Таким образом, окно представления полностью контролирует мышь в процессе рисования линии. Значение переменной m_Dragging устанавливается равным 1, что информирует других обработчиков сообщений о выполнении операции рисования.

Оставшийся фрагмент программы предназначен для ограничения перемещения указателя мыши границами окна представления. Функция CWnd::GetClientRect возвращает текущие координаты окна представления, а CWnd::ClientToScreen преобразовывает их в экранные (т.е. координаты, заданные по отношению к верхнему левому углу экрана). Наконец, функция ::ClipCursor ограничивает перемещения указателя в пределах заданных координат, удерживая его в окне представления.

Функция ::ClipCursor содержится в Win32 API, а не в MFC. Поскольку она описана как глобальная, ее имени предшествует операция расширения области видимости (::). Использование данной операции не является необходимым, если глобальная функция не скрыта функцией-членом с таким же именем.

Схема сообщений. Когда мастер ClassWizard создает обработчик сообщения, то помимо объявления и определения функции-члена он также добавляет ее в специальную структуру MFC, называемую схемой (картой) сообщений (message map) и связывающую функции с обрабатываемыми сообщениями. Схема сообщений позволяет библиотеке MFC вызывать для каждого типа сообщения соответствующий обработчик.

Мастера AppWizard и ClassWizard создают необходимый код для реализации схемы сообщений, основанной на наборе MFC-макросов. После применения мастера ClassWizard для определения обработчиков трех видов сообщений мыши (создание которых рассматривается ниже) в файл CMiniDrawView.h будут добавлены следующие макросы и объявления функций:

// Сгенерированные функции схемы сообщений

protected:

//{{AFX_MSG(CMiniDrawView)

afx_msg void OnLButtonDown(UINT nFlags, CPoint point);

afx_msg void OnMouseMove(UINT nFlags, CPoint point);

afx_msg void OnLButtonOp(UINT nFlags, CPoint point);

//})AFX_MSG DECLARE MESSAGE MAP()

В файл реализации представления MiniDrawView.cpp будут добавлены соответственно следующие макросы:

BEGIN_MESSAGE_MAP (CMiniDrawView, CView)

//{(AFX_MSG_MAP(CMiniDrawView)

ON_WM_LBUTTONDOWN ()

ON_WM_MOUSEMOVE ()

ON_WM_LBUTTONUP ()

//}}AFX_MSG_MAP END_MESSAGE_MAP()

Когда сообщение передается объекту класса, MFC обращается к схеме сообщений, чтобы определить, есть ли в классе обработчик такого сообщения. Если обработчик найден, ему передается управление. При отсутствии обработчика MFC ищет его в базовом классе. Если это не дает результата, то поиск будет продолжен по иерархии классов до первого встретившегося обработчика. Если в иерархии обработчик отсутствует, то будет выполнена стандартная обработка сообщения. Если же это командное сообщение, то оно перенаправляется следующему объекту в описанной ранее последовательности.

Обратите внимание: некоторые из классов библиотеки MFC, от которых порождены классы программы, содержат обработчики сообщений. Следовательно, даже если производный класс не определяет обработчик сообщения, обработчик базового класса может обеспечить соответствующую обработку. Например, базовый класс Документа CDocument содержит обработчики сообщений, поступающих при выборе команд Save, Save As... и Close в меню File (соответственно OnFileSave, OnFileSaveAs И OnFileClose).

Функция OnMouseMove. Далее следует определить функцию для обработки сообщения WM_MOUSEMOVE. Так как пользователь перемещает указатель мыши внутри окна представления, это окно получает последовательность сообщений WM_MOUSEMOVE, каждое из которых содержит информацию о текущей позиции указателя. Для определения обработчика таких сообщений используйте мастер ClassWizard. Выполните последовательность действий, приведенную в предыдущем параграфе. Однако на шаге 5 в списке Messages выберите сообщение WM_MOUSEMOVE вместо сообщения WM_LBUTTONDOWN. Мастер ClassWizard создаст функцию OnMouseMove.

После щелчка на Edit Code введите в функцию OnMouseMove строки:

```
void CMiniDrawView::OnMouseMove(UINT nFlags, CPoint point)
{
   // TODO:  Здесь  добавьте  собственный код обработчика
   // и/или вызов  стандартного обработчика
   ::SetCursor (m_HCross);
   if (m_Dragging)
   {
      CClientDC ClientDC (this);
      ClientDC.SetROP2(R2_NOT);
      ClientDC.MoveTo(m_PointOrigin);
      ClientDC.LineTo(m_PointOld);
      ClientDC.MoveTo (m_PointOrigin);
      ClientDC.LineTo(point);
      m_PointOld=point;
   }
   CView::OnMouseMove(nFlags, point);
}
```

При перемещении указателя мыши внутри окна представления функция OnMouseMove вызывается через определенные промежутки времени. Добавленный в нее код обеспечивает решение двух основных задач: первой – вызов API-функции ::SetCursor для отображения крестообразного курсора вместо стандартного курсора-стрелки. Вспомните: дескриптор крестообразного курсора получен в конструкторе класса; второй – выполнения операции рисования (значение переменной m_Dragging отлично от нуля). При этом выполняются следующие действия:

1) стирание линии, нарисованной при получении предыдущего сообщения WM_MOUSEMOVE(если она имеется);
2) рисование новой линии от начальной точки, в которой была нажата левая кнопка мыши с координатами, сохраняемыми в переменной m_PointOrigin, до текущей позиции указателя, заданной параметром point;
3) сохранение текущей позиции указателя в переменной m_PointOld.

Для рисования внутри окна функция OnMouseMove создает объект контекста устройства, связанный с окном представления. Затем OnMouseMove вызывает функцию CDC::SetROP2, задающую режим рисования, в котором линии строятся методом инвертирования (обращения) текущего цвета экрана. В этом режиме линия, нарисованная в определенной позиции в первый раз, будет видима, а при повторном выводе в той же самой позиции - невидима. Таким образом, обработчики сообщения легко отображают и удаляют группы временных линий. Линии выводятся с помощью CDC::MoveTo, указывающей положение одного конца линии, и CDC::LineTo, задающей положение другого конца.

Окончательный результат использования OnMouseMove состоит в том, что при перемещении указателя с нажатой кнопкой мыши внутри окна представления временная линия всегда соединяет свое начало с текущей позицией

указателя. Показывает, какой будет постоянная линия, если пользователь отпустит кнопку мыши сейчас.

Функция OnLButtonUp. Наконец, необходимо определить функцию для обработки сообщения WM_LBUTTONUP, передаваемого при отпускании левой кнопки мыши. Чтобы создать функцию, используйте мастер ClassWizard, как и для двух предыдущих сообщений. Однако в списке Messages диалогового окна мастера выберите идентификатор WM_LBUTTONUP. Когда функция будет сгенерирована – добавьте в ее определение в файле MiniDraw.cpp следующий текст:

```
void CMiniDrawView::OnLButtonUp(UINT nFlags, CPoint point)
{
   // TODO: Здесь добавьте код собственного обработчика
   // и/или вызов стандартного обработчика
   if (m_Dragging)
   {
      m_Dragging = 0;
      ::ReleaseCapture();
      ::ClipCursor (NULL);
      CClientDC ClientDC (this);
      ClientDC.SetROP2 (R2_NOT);
      ClientDC.MoveTo (m_PointOrigin);
      ClientDC.LineTo (m_PointOld);
      ClientDC.SetROP2 (R2_COPYPEN);
      ClientDC.MoveTo (m_PointOrigin);
      ClientDC.LineTo (point);
      CMiniDrawDoc* pDoc=GetDocument();
      pDoc->AddLine(m_PointOrigin.x, m_PointOrigin.y, point.x, point.y);
   }
   CView::OnLButtonUp(nFlags, point);
}
```

Если пользователь перемещает указатель мыши с нажатой кнопкой (значение переменной m_Dragging отлично от нуля), добавленный код завершает операцию рисования и строит постоянную линию. В частности, выполняются следующие действия:

1) присваивание значения 0 переменной m_Dragging, что информирует других обработчиков сообщений о завершении операция рисования;

2) вызов API-функции ::ReleaseCapture для завершения захвата мыши. После этого сообщения мыши будут снова передаваться любому окну, в котором находится указатель;

3) передача указателя NULL в API-функцию ::ClipCursor, что позволяет пользователю снова перемещать указатель мыши по всему экрану;

4) стирание временной линии, выведенной предыдущим обработчиком сообщения WM_MOUSEMOVE;

5) вывод постоянной линии от начальной точки до текущей позиции указателя;

6) добавление новой линии в список существующих линий.

WM_LBUTTONUP – это последнее сообщение мыши, которое нужно обработать в программе MiniDraw. Ниже приведен полный список уведомляющих сообщений. Возможно, вы захотите обработать некоторые из них в других программах Windows:

WM_MOUSEMOVE…….……..	перемещен указатель мыши на новое место внутри рабочей области
WM_LBUTTONDOWN……..	нажата левая кнопка
WM_MBUTTONDOWN…….	нажата средняя кнопка
WM_RBUTTONDOWN……..	нажата правая кнопка
WM_LBUTTONUP…….	отпущена левая кнопка
WM_MBUTTONUP…….	отпущена средняя кнопка
WM_RBUTTONUP …….	отпущена правая кнопка
WM_LBUTTONDBCLICK…….……	выполнено двойное нажатие левой кнопки
WM_MBUTTONDBCLICK……	выполнено двойное нажатие средней кнопки
WM_RBUTTONDBCLICK……	выполнено двойное нажатие правой кнопки

Параметры сообщений мыши. Всем обработчикам сообщений передаются два параметра: nFlags и point.

Параметр nFlags показывает состояние кнопок мыши и некоторых клавиш в момент наступления события. Состояние каждой кнопки или клавиши представляется специальным битом. Для обращения к отдельным битам можно использовать битовые маски (табл. 1.3). Например, в следующем фрагменте проверяется, была ли нажата клавиша Shift при перемещении мыши:

```
void CminiDrawView::OnMouseMove(UINT nFlags, CPoint point)
{
    if(nFlags&MK_SHIFT)
    // клавиша Shift была нажата при перемещении мыши
}
```

Параметр point – это структура CPoint, задающая координаты курсора мыши в тот момент, когда произошло событие мыши. Поле x (point.x) содержит горизонтальную координату указателя, поле y (point.y) – вертикальную. Координаты определяют местоположение указателя относительно верхнего левого угла окна. Точнее говоря, параметр point задает координаты острия указателя мыши. Острие – это отдельный пиксель внутри указателя, выделяемый при его проектировании. Острием стандартного курсора-стрелки является ее конец, а острием стандартного крестообразного курсора – точка пересече-

ния его линий. Ниже приведены битовые маски для доступа к битам параметра nFlags, передаваемого обработчику сообщения мыши:

MK_CONTROL……Ctrl
MK_LBUTTON……левая кнопка мыши
MK_MBUTTON……средняя кнопка мыши
MK_RBUTTON……правая кнопка мыши
MK_SHIFT……Shift

Перерисовка окна. Теперь программа постоянно хранит данные, позволяющие восстановить линию, а класс представления может использовать их при перерисовке окна. Вспомните: для перерисовки окна система удаляет его содержимое, а затем вызывает функцию OnDraw класса представления. В минимальную версию функции OnDraw, генерируемую мастером AppWizard, необходимо добавить собственный код для перерисовки окна. Для этого в функцию CMiniDrawView: :OnDraw в файле MiniDrawView.cpp необходимо добавить строки:

```
// Отображение данных класса CMiniDrawView
void CMiniDrawView::OnDraw(CDC* pDC)
{
    CMiniDrawDoc* pDoc=GetDocument();
    ASSERT_VALID(pDoc);
     int Index=pDoc->GetNumLines();
     while(Index--)
       pDoc->GetLine (Index)->Draw (pDC);
}
```

В этом коде вызывается функция CMiniDrawDoc::GetNumLines, позволяющая определить количество линий, сохраненных объектом документа. В этом фрагменте программы для каждой линии сначала вызывается функция CMiniDrawDoc::GetLine, которая получает указатель на соответствующий объект класса CLine, а затем этот указатель используется для рисования линии с помощью CLine::Draw

Так как графические данные хранятся внутри объекта документа, целесообразно добавить в программу некоторые команды меню Edit, чтобы можно было эти данные изменять. Далее будут добавлены команды Undo для удаления последней нарисованной линии и Delete All для удаления всех линий.

Рассмотрим, как можно использовать редактор ресурсов для настройки программы MiniDraw (меню и значка). В окне Project Workspace откройте вкладку ResourceView и разверните граф ресурсов программы всех категорий: Accelerator, Dialog, Icon, Menu, String Table и Version.

Чтобы настроить меню программы, сделайте следующее:

1. Выполните двойной щелчок на идентификаторе IDR_MAIN_FRAME в разделе Menu. Обратите внимание: идентификатор IDR_MAIN_FRAME используется также для таблицы горячих клавиш и главного значка программы.

Developer Studio открывает окно редактора меню, отображающее меню программы MiniDraw, созданное мастером AppWizard.

2. В окне редактора щелкните на меню File, удалите все пункты, кроме команд New, Exit и разделителя между ними. Чтобы удалить пункт меню (команду или разделитель), щелкните на нем и нажмите клавишу Del. (Нельзя удалять пустое окно в нижней части меню. Эта область предназначена для добавления новых пунктов и в завершенном меню программы отсутствует).

3. Щелкните кнопкой мыши на меню Edit и нажмите клавишу Del, чтобы удалить его. После запроса об удалении меню щелкните на ОК.

4. Выполните двойной щелчок в пустой заготовке справа от последнего пункта меню. Редактор меню откроет диалоговое окно Menu Item Properties.

5. В поле Caption введите &Edit. После этого в строке меню появится меню Edit. Обратите внимание: при создании ниспадающего меню идентификатор не вводится. Идентификаторы присваиваются только командам меню.

6. С помощью мыши перетащите меню Edit влево таким образом, чтобы оно разместилось между меню File и Help.

7. Выполните двойной щелчок на пустом прямоугольнике внутри меню Edit (под заголовком), чтобы снова открыть диалоговое окно Menu Item Properties для определения новых команд меню.

8. В поле ID введите ID_EDIT_UNDO, а в Caption - &Undo. Теперь в меню Edit появится команда Undo.

9. Выполните двойной щелчок на пустом поле внизу меню Edit (под командой Undo) и в диалоговом окне Menu Item Properties отметьте опцию Separator. Под командой Undo будет вставлен разделитель.

10. Выполните двойной щелчок на пустом поле внизу меню Edit, затем введите в поле ID значение ID_EDIT_CLEAR_ALL, а в Caption - Delete ALL. В меню добавится команда Delete All. Теперь меню Edit завершено.

11. Закройте окно редактора меню и сохраните результаты, выбрав команду Save All в меню File или щелкнув на кнопке Save All панели инструментов Standard.

Если хотите сконструировать свой значок программы MiniDraw (для замены стандартного значка, предоставляемого библиотекой MFC), выполните следующие действия:

1. Сделайте двойной щелчок на идентификаторе IDR_MAINFRAME в разделе Icon в окне Project Workspace. Developer Studio откроет окно графического редактора, отображающее текущий значок программы. Обратите внимание: файл изображения содержит две версии рисунка: крупный (32 x 32 пикселя) и мелкий (16 x 16 пикселей). Мелкий рисунок используется там, где отображаются «мелкие значки» (например, в области заголовка окна программы или в окне Explorer, если в меню View выбрать опцию Small Icons). Крупные рисунки – там, где «крупные значки» (например, в диалоговом окне About или Explorer, если выбрана опция Large Icon в меню View).

2. Чтобы отредактировать крупный рисунок, выберите пункт «Standard (32x32)» в списке Device в верхней части окна. Для удаления текущего значка (при разработке нового) нажмите клавишу Del. После чего вы можете создать абсолютно новый рисунок.

3. Для отображения мелкого значка выберите в списке пункт «Small (16x16)». При желании рисунок можно отредактировать или удалить, выбрав команду Delete Device Image в меню Image. Если мелкий значок отсутствует, Windows сожмет крупный рисунок (с некоторой потерей качества изображения) и заменит им мелкие.

4. После редактирования значка удалите окно графического редактора, выполнив двойной щелчок на системном меню или щелкнув на Close.

Сохраните внесенные изменения, выбрав в меню File команду Save All. Основная информация о ресурсах хранится в файле определения ресурсов MiniDraw.rc, а информация о значке – в MiniDraw.ico подкаталога \res каталога проекта. Файл определения ресурсов содержит оператор ICON, идентифицирующий файл значка. Когда программа будет сгенерирована, программа Rc.exe (Resource Compiler – компилятор ресурсов) обработает информацию о ресурсах, содержащуюся в этих файлах, и внесет данные о них в исполняемый файл.

Настройка окна MiniDraw. При использовании программы MiniDraw в настоящем виде возникают две проблемы.

Первая: хотя обработчик сообщения WM_MOUSEMOVE отображает требуемый указатель крестообразной формы, Windows также пытается отобразить стандартный курсор-стрелку, назначенный окну представления библиотекой MFC. В результате из-за переходов между этими двумя формами при перемещении указателя возникает неприятное мерцание.

Вторая проблема: если пользователь выбирает на панели управления темный цвет «Window», линии, нарисованные в окне представления, становятся невидимыми или едва заметными. При создании окна MFC присваивает ему установки, задающие цвет фона с использованием текущего цвета «Window». Однако программа всегда выводит черные линии.

Обе проблемы можно решить, добавив необходимые строки в функцию PreCreateWindow класса CMiniDrawView. При генерации программы мастер AppWizard определяет шаблон функции CMiniDrawView::PreCreateWindow, переопределяющей виртуальную функцию PreCreateWindow класса CView, которую MFC вызывает непосредственно перед созданием окна представления.

Чтобы настроить окно представления MiniDraw, добавьте операторы в PreCreateWindow в файл MiniDrawView.cpp:

```
BOOL CMiniDrawView::PreCreateWindow(CREATESTRUCT &cs)
{
  // TODO: Здесь измените класс или стили окна
```

```
    // модифицируя поля структуры cs
    m_ClassName=AfxRegisterWndClass(
CS_HREDRAW|CS_VREDRAW, // стили окна
                    0,                          //без указателя;
                    (HBRUSH)::GetStockObject (WHITE_BRUSH),
                                // задать чисто белый фон;
                    0);             //без значка
    cs.lpszClass = m_ClassName;
    return CView::PreCreateWindow(cs);
}
```

В функцию PreCreateWindow передается ссылка на структуру CREATESTRUCT, поля которой хранят свойства окна, задаваемые библиотекой MFC при его создании (координаты окна, его стили и т.д.). При присвоении значений одному или нескольким полям структуры MFC использует заданные значения вместо стандартных.

Одно поле структуры CREATESTRUCT (lpszClass) хранит имя класса окна windows, но это не класс языка C++, а структура данных, сохраняющая набор общих свойств окна. В добавленном фрагменте вызывается функция AfxRegisterWndClass, создающая новый класс окна, а затем присваивается имя класса полю lpszClass структуры CREATESTRUCT. Таким образом, окно представления создается с настраиваемыми свойствами, сохраняемыми внутри данного класса. Обратите внимание: AfxRegisterWndClass – глобальная функция, предоставляемая библиотекой MFC.

При вызове функции AfxRegisterWndClass устанавливаются следующие параметры:

- первый параметр – задает стили CS_HREDRAW и CS_VREDRAW, позволяющие перерисовать окно при изменении его размеров (обычно окна представления создаются с использованием этих двух стилей);
- второй – задает форму указателя, автоматически отображаемого в окне Windows. Этому параметру присваивается значение 0, поэтому Windows не пытается его отобразить с помощью функции OnMouseMove. Таким образом нежелательное мерцание устраняется;
- третий параметр – определяет стандартную белую кисть, используемую для заливки фона окна представления. В результате цвет фона окна всегда будет белым, а черные линии – видимыми, независимо от цвета “Window”, выбранного в панели управления;
- последний параметр определяет значок окна. Так как окно представления его не отображает, параметру присваивается значение 0 (значок программы задается для главного окна).

Удаление данных документа. Каждый раз, когда пользователь выбирает в меню File команду New, MFC (а именно, функция OnFileNew класса CWinApp) вызывает виртуальную функцию CDocument:: DeleteContents для удаления содержимого текущего документа перед инициализацией нового.

В последующих версиях программы MiniDraw эта функция будет также вызываться перед открытием существующего документа.

Чтобы удалить данные, сохраняемые этим классом, необходимо написать новую версию этой функции в виде члена класса документа.

Переопределение виртуальной функции является общераспространенным и эффективным способом настройки MFC. Чтобы сгенерировать объявление и оболочку функции DeleteContents, воспользуйтесь мастером ClassWizard:

1. Выберите в меню View команду ClassWizard или нажмите Ctrl+W. Появится диалоговое окно мастера ClassWizard.

2. Откройте вкладку Message Maps, позволяющую определить функции-члены.

3. В списке Class Name выберите CMiniDrawDoc. Это имя класса выбирается потому, что необходимо определить виртуальную функцию, принадлежащую классу документа.

4. В списке Object IDs выберите пункт CMiniDrawDoc, что приведет к отображению в списке Messages имен виртуальных функций, определенных в родительских классах. Каждую из этих функций можно переопределить. Все пункты списка Messages (кроме идентификаторов сообщений, начинающихся с WM_) – это виртуальные функции (пусть имя списка Messages (сообщения) не вводит вас в заблуждение). Обратите внимание: при выборе имени функции в нижней части диалогового окна мастера появляется ее описание.

5. В списке Messages выберите пункт DeleteContents и щелкните на Add Function.

6. Щелкните на кнопке Edit Code. Мастер AppWizard вызовет перемещение курсора в тело функции DeleteContents, сгенерированной внутри файла MiniDrawDoc.cpp

Теперь добавьте следующие строки в функцию DeleteContents, сгенерированную мастером ClassWizard:

```
void CMiniDrawDoc::DeleteContents()
{
    // TODO: Здесь добавьте собственный код обработчика
    // и/или вызов базового класса
    int Index=m_LineArray.GetSize();
    while(Index--)
    delete m_LineArray.GetAt (Index);
    m_LineArray.RemoveAll ();
    CDocument::DeleteContents();
}
```

В коде, добавленном в определение функции DeleteContents, сначала происходит обращение к функции CObArray класса GetSize, чтобы получить количество указателей класса CLine, сохраненных в данный момент объектом m_LineArray. Затем при вызове функции CTypedPtrArray::GetAt выбирается

каждый указатель, а оператор delete используется для удаления каждого соответствующего объекта класса CLine (вспомните: объекты класса CLine создавались с использованием оператора new). Наконец, для удаления всех указателей, сохраненных в данное время в массиве m_LineArray, вызывается функция RemoveAll класса CObArray.

После вызова функции DeleteContents библиотека MFC (косвенным образом) удаляет окно представления и вызывает функцию OnDraw. Однако функция OnDraw не отображает линии, потому что они были удалены из класса документа. Окончательный результат – удаление данных документа командой New (из меню File) и очистка окна представления перед созданием нового рисунка.

Реализация команд меню. Теперь воспользуемся мастером ClassWizard для реализации двух команд, добавленных в меню Edit: Delete All и Undo.

Выполните следующие действия для определения обработчика сообщения, получающего управление при выборе команды Delete All:

1. Откройте диалоговое окно ClassWizard, а затем - вкладку Message Maps.

2. В списке Class name выберите пункт CMiniDrawDoc, чтобы функция обработки сообщения, генерируемого при выборе команды Delete All, стала членом класса документа. Вспомните, что командное сообщение может обрабатываться любым из четырех основных классов программы (классом представления, документа, главного окна или приложения). Для обработки команды Delete All выбирается класс документа, потому что эта команда непосредственно воздействует на данные документа (удаляет их) и, следовательно, попадает в "компетенцию" класса документа.

3. В списке Object IDs выберите идентификатор ID_EDIT_CLEAR_ALL. Вспомните, что именно он при создании меню был присвоен команде Delete All. Сразу после выбора идентификатора в списке Messages отобразятся идентификаторы двух типов сообщении, которые эта команда меню может передавать объекту класса документа: COMMAND и UPDATE_COMMAND_UI. Идентификатор COMMAND указывает на сообщение, передаваемое при выборе пользователем пункта меню. Идентификатор UPDATE_COMMAND_UI указывает на сообщение, передаваемое при первом открытии меню, содержащего команду. Обратите внимание: идентификаторы COMMAND и UPDATE_COMMAND_UI указывают на два типа командных сообщений, которые могут генерироваться объектом пользовательского интерфейса.

4. В списке Messages выберите сообщение COMMAND.

5. Щелкните на кнопке Add Function для генерации функции обработки сообщения. Когда мастер ClassWizard отобразит диалоговое окно Add Member Function, щелкните на кнопке OK, чтобы принять стандартное имя функции OnEditClearAll, и продолжите генерацию функции. Теперь мастер ClassWizard объявляет функцию внутри определения класса CMiniDrawDoc в файле MiniDrawDoc.h, добавляет минимальное определение функции в файл

MiniDrawDoc.cpp и генерирует код для добавления функции в схему обработки сообщений класса документа.

6. При выбранной функции OnEditClearAll в списке Member Functions щелкните на кнопке Edit Code. Мастер ClassWizard откроет файл MiniDrawDoc.cpp (если он еще не открыт) и перейдет на определение функции OnEditClearAll.

7. Добавьте в функцию OnEditClearAll следующий код:

```
void CMiniDrawDoc::OnEditClearAll()
{
    // TODO: Здесь добавьте собственный код обработчика
    DeleteContents();
    UpdateAllViews(0);
}
```

Вызов ранее определенной функции DeleteContents приводит к удалению содержимого документа. Далее функция UpdateAllViews класса CDocument удаляет текущее содержимое окна представления.

Следующим этапом будет определение обработчика сообщения UPDATE_COMMAND_UI, посылаемого при первом открытии пункта меню, содержащего команду Delete All (это меню Edit). Так как это сообщение посылается до того, как меню станет видимым, обработчик может использоваться для инициализации команды в соответствии с текущим состоянием программы. Обработчик делает команду Delete All доступной, если документ содержит одну или более линий, и недоступной, если документ их не содержит. Чтобы сделать это, следуйте описанным действиям, однако в списке Messages выберите идентификатор UPDATE_COMMAND_UI. Мастер ClassWizard сгенерирует функцию с именем OnUpdateEditClearAll. После щелчка на кнопке Edit Code добавьте в эту функцию ниже приведенный код:

```
void CMiniDrawDoc::OnUpdateEditClearAll(CcmdUI *pCmdUI)
{
  // TODO: Здесь добавьте собственный код обработчика
  pCmdUI->Enable(m_LineArray.GetSize());
}
```

Функции OnUpdateEditClearAll передается указатель на объект класса CcmdUI – это MFC-класс, предоставляющий функции для инициализации команд меню и других объектов пользовательского интерфейса. Добавленный код вызывает функцию Enable класса CCmdUI. Она делает доступной команду меню Delete All, если документ содержит хотя бы одну линию. В противном случае блокирует команду, которая отображается затененной серым цветом, и пользователь не может ее выбрать. Таким образом, функцию OnEditClearAll нельзя вызвать, если документ пуст.

Инициализация команд меню. В функцию, которая обрабатывает сообщение UPDATE_COMMAND_UI команды меню, передается указатель на объект класса CCmdUI, связанный с выбранной командой. Класс CCmdUI

предоставляет четыре функции, которые можно использовать для инициализации команд: Enable, SetCheck, SetRadio и SetText.

Чтобы сделать команду доступной, в функцию Enable передается значение TRUE, а для блокирования команды – FALSE. Например:

virtual void Enable(BOOL bOn=TRUE);

В функцию SetCheck можно передать значение 1, чтобы сделать пункт меню выбранным, или значение 0, чтобы отменить выбор. Например:

virtual void SetCheck (int nCheck=1);

Обычно команда меню, представляющая какую-либо функциональную возможность программы, выделяется, если данная возможность активизирована.

Чтобы отметить пункт меню с помощью специального маркера (кружка), можно передать значение TRUE функции SetRadio, а чтобы удалить маркер – значение FALSE. Например:

virtual void SetRadio(BOOL bOn=TRUE);

Наконец, чтобы изменить надпись команды меню, можно вызвать функцию SetText:

virtual void SetText(LPCTSTR lpszText);

где lpszText – указатель на новую строку текста. Например, если предыдущее действие состояло в удалении текста, то для замены команды Undo на Undo Delete можно вызвать функцию SetText.

Обработка команды Undo. Последняя задача состоит в реализации функции обработки команды Undo меню Edit.

Сначала определим функцию, получающую управление, когда пользователь выбирает команду Undo. Чтобы сделать это, запустите мастер ClassWizard и выполните действия, описанные в предыдущем параграфе. Однако при выполнении пункта 3 в списке Object IDs выберите идентификатор ID_EDIT_UNDO. Мастер ClassWizard создаст функцию с именем OnEditundo. Добавьте в эту функцию следующий код:

```
void CMiniDrawDoc: : OnEditundo ()
{
    // TODO: Здесь добавьте собственный код обработчика
    int Index=m_LineArray.GetUpperBound();
    if (Index>-1)
    {
        delete m_LineArray.GetAt(Index);
        m_LineArray.RemoveAt(Index);
    }
    UpdateAllViews(0);
}
```

Для получения индекса последней линии в добавленном коде сначала вызывается функция GetUpperBound класса CObArray. Затем с целью получения указателя на объект класса CLine для последней линии вызывается функция CTypedPtrArray::GetAt, а для удаления этого объекта используется опера-

тор delete. И, наконец, вызывается функция UpdateAllViews, которая удаляет окно представления и вызывает функцию CMiniDrawView::OnDraw. После этого обработчик OnDraw перерисовывает линии, оставшиеся в документе. Обратите внимание: при многократном выборе команды Undo обработчик OnEditundo продолжает удалять линии до тех пор, пока не останется ни одной.

Теперь определите функцию инициализации команды меню Undo. Чтобы сделать это, воспользуйтесь алгоритмом, приведенным в предыдущем параграфе. Однако в пункте 3 из списка Object IDs выберите идентификатор ID_EDIT_UNDO, а в пункте 4 из списка Message - идентификатор сообщения UPDATE_COMMAND_UI. Мастер ClassWizard создаст функцию с именем OnUpdateEditUndo. Добавьте в эту функцию следующий код:

```
void CMiniDrawDoc::OnUpdateEditUndo(CCmdUI* pCmdUI)
{
    // TODO: Здесь добавьте собственный код обработчика
        pCmdUI->Enable(m_LineArray.GetSize());
}
```

Работа этой функции аналогична работе функции OnUpdateEditClearAll, описанной ранее. Она делает команду Undo доступной при наличии хотя бы одной линии для удаления.

Выше были приведены все необходимые изменения. Теперь можно построить и запустить программу.

Задания:

1. Разработать программу, позволяющую рисовать окружности.
2. Разработать программу, позволяющую рисовать прямоугольники.
3. Разработать программу, позволяющую рисовать линии, содержащие стрелку на конце.
4. Разработать программу, позволяющую рисовать линии, содержащие стрелки на 2 концах.
5. Разработать программу, позволяющую выводить набранный текст.
6. Разработать программу, позволяющую рисовать окружности несколькими цветами.
7. Разработать программу, позволяющую рисовать прямоугольники несколькими цветами.
8. Разработать программу, позволяющую рисовать линии, содержащие стрелку на конце, несколькими цветами.
9. Разработать программу, позволяющую рисовать линии, содержащие стрелки на 2 концах, несколькими цветами.
10. Разработать программу, позволяющую выводить набранный текст несколькими цветами.

ЛАБОРАТОРНАЯ РАБОТА №2
«ХРАНЕНИЕ ДАННЫХ»

Цель работы – ознакомиться с процессом сохранения и считывания данных Win32-приложения с помощью Microsoft Visual C++.

Методические указания

В лабораторной работе изучаются принципы сохранения и загрузки данных документа с файлов на диске. Для демонстрации базовых методов ввода-вывода рассмотрена методика добавления кода, реализующего стандартные команды меню File (New, Open..., Save и Save As...), в программу, написанную в прошлой лабораторной работе. Вы узнаете, как реализовать технологию drag-and-drop, позволяющую открывать файл, перетаскивая объект файла из папки Windows или окна Windows Explorer и отпуская его в окне программы.

Добавление средств ввода-вывода в программу MiniDraw. Для добавления команд в меню File после открытия проекта в Developer Studio откройте вкладку Resource View в окне Workspace. Чтобы изменить меню программы, откройте редактор меню, выполнив двойной щелчок на идентификаторе IDR_MAINFRAME.

В редакторе меню откройте меню File. Под командой New в меню File необходимо добавить команды Open…, Save, Save As…, разграничитель и команду Recent File. Для этого используйте методику, описанную ранее. Ниже для каждой новой команды в таблице приведены идентификатор, надпись и другие свойства, задаваемые в диалоговом окне Menu Item Properties.

Свойства пунктов, добавляемых в меню File программы

Идентификатор пункта	Надпись	Другие параметры
ID_FILE_OPEN	&Open...\tCtrl-O	Отсутствуют
ID_FILE_SAVE	&Save\tCtri+S	Отсутствуют
ID_FILE_SAVE_AS	Save &As...	Отсутствуют
Отсутствует	Отсутствует	Разделитель
ID_FILE_MRU_FILE1	Recent File	Недоступны

Если в программе открыт хотя бы один файл, то MFC заменяет надпись Recent File именем последнего открытого файла. MFC будет добавлять в меню File имена последних использованных файлов. При создании программы мастер AppWizard устанавливает максимальное количество последних использованных файлов, равное 4. MFC хранит их имена в файле инициализации программы (MiniDraw.ini) каталога Windows, поэтому список команд сохраняется, когда пользователь выходит из программы и перезапускает ее. Закройте окно редактора меню. Для этих команд задавать комбинации клавиш не нуж-

но, так как мастер AppWizard уже определил их при первичном создании приложения. Вспомните: сгенерированное меню содержало все команды, перечисленные в вышеуказанной таблице. (В прошлой лабораторной работе они были удалены, так как не использовались.)

Следующий этап – изменение строкового ресурса программы для определения стандартного расширения файлов, отображаемых в диалоговых окнах Open и Save As. Для этого откройте редактор строк, выполнив двойной щелчок на элементе String Table графа ResourceView.

Первая строка в окне редактора имеет идентификатор IDR_MAINFRAME. Она создана мастером AppWizard и содержит информацию, относящуюся к программе. Ее текущее значение такое:

MiniDraw\n\nMiniDr\n\n\nMiniDraw.Document\nMiniDr Document

Чтобы модифицировать строку, откройте диалоговое окно String Properties. Смените содержимое поля Caption следующим образом:

MiniDraw\n\nMiniDr\nMiniDraw
Files(*.drw)\n.drw\nMiniDraw.Document\nMiniDr Document

При редактировании строки не нажимайте клавишу Enter. Когда текст достигнет край текстового поля, тогда он автоматически будет перенесен на следующую строку.

Вставьте строку "MiniDraw Files(*.drw)", отображаемую в списке Files of type (или Save as type) диалогового окна Open (или Save As) и определяющую стандартное расширение файлов программы. Затем вставьте "(.drw)" – стандартное расширение файлов. Если в процессе выполнения программы MiniDraw расширение файла при открытии или сохранении не указано, то в диалоговых окнах Open и Save As отобразится список всех файлов со стандартными расширениями, а в диалоговом окне Save As стандартное расширение файла будет добавлено к его имени.

Задание расширения файлов в новой программе. Приведенные инструкции относятся только к заданию стандартного расширения файлов в существующей программе. Это можно также сделать при создании приложения мастером AppWizard:

1. В диалоговом окне мастера AppWizard (Step 4) щелкните на кнопке Advanced..., чтобы открыть диалоговое окно Advanced Options, а затем откройте в этом окне вкладку Document Template Strings.

2. Введите в поле File extension стандартное расширение файла (без точки), например drw.

3. После ввода стандартного расширения мастер AppWizard автоматически введет описание расширения в поле Filter name (например, MiniDraw Files (*.drw)). Эта строка отображается в диалоговом окне Open (или Save As) в списке Files of type (или Save as type). При желании эту строку можно отредактировать.

4. Щелкните на кнопке Close и введите оставшуюся информацию в диалоговые окна мастера AppWizard.

Поддержка команд меню File. В предыдущей лабораторной работе при добавлении команд Undo и Delete All в меню Edit для определения их обработчиков использовался мастер ClassWizard. Для команд New, Open..., Save и Save As... определять обработчики не требуется, так как они предоставляются MFC. В этом случае необходимо написать код для их поддержки. Библиотека MFC также предоставляет обработчики команд для работы с последними использованными файлами в меню File. Функция OnFileNew класса CWinApp (от которого порождался класс приложения MiniDraw) обрабатывает команду New. Эта функция вызывает виртуальную функцию DeleteContents для удаления текущего содержимого документа, а затем инициализирует новый документ.

Команда Open... обрабатывается функцией OnFileOpen класса CWinApp, которая отображает стандартное диалоговое окно Open. Если выбрать файл и щелкнуть на кнопке Open, то OnFileOpen откроет файл для чтения, а затем вызовет функцию Serialize класса документа (CMiniDrawDoc:: Serialize). Функция Serialize предназначена для фактического выполнения операции чтения. Функция OnFileOpen сохраняет полный путь к загруженному файлу и отображает имя файла в строке заголовка главного окна. (Обработчик команды загрузки последнего использованного файла открывает файл для чтения и вызывает функцию Serialize, не отображая диалоговое окно Open).

Если файл еще не был сохранен, то в поле File name диалогового окна Save As отобразится имя файла по умолчанию, которое создается добавлением стандартного расширения файла (.drw для программы MiniDraw) к имени "Untitled".

Функция OnFileSave класса CDocument (от которого порожден класс документа программы MiniDraw) обрабатывает команду Save, а функция OnFileSaveAs класса CDocument – команду Save As... Если документ сохраняется впервые, то функции OnFileSaveAs и OnFileSave начинают работу с отображения стандартного диалогового окна Save As, позволяющего задать имя файла. Эти функции открывают файл для записи, а затем вызывают функцию CMiniDrawDoc::Serialize для выполнения собственно операции записи. Они также сохраняют полный путь к файлу и отображают его имя в строке заголовка. Длина имени и расширения файла в Windows 95/98 и Windows NT не ограничена восемью и тремя символами соответственно. В Windows 95, например, имя файла может содержать до 255 символов. Если хотите быть уверенным в достаточном размере буфера, то для хранения имени файла и полного или частичного пути к файлу используйте одну из констант, определенных в файле Stdio.h: _MAX_PATH, _MAX_DRIVE, _MAX_DIR, _MAX_FNAME, и _MAX_EXT.

Сериализация данных документа. При создании программы MiniDraw мастер AppWizard определяет в файле MiniDrawDoc.cpp следующую минимальную реализацию функции Serialize:

```
void CMiniDrawDoc::Serialize(CArchive& ar)
{
   if(ar.IsStoring())
   {
      // TODO: здесь добавьте код сохранения
   } else
   {
      // TODO: здесь добавьте код загрузки
}
```

В эту функцию необходимо добавить собственный код для чтения или записи данных. MFC передает функции Serialize ссылку на экземпляр класса CArchive. Для открытого файла задается объект класса CArchive, предоставляющий набор функций для чтения и/или записи данных этого файла.

Если файл открыт для записи (т.е. выбрана команда Save или Save As...), то функция IsStoring класса CArchive возвращает значение TRUE, а если файл открыт для чтения (т.е. выбрана команда Open... или команда вызова последнего использованного файла из меню File), то функция IsStoring возвращает значение FALSE. Следовательно, код операции вывода помещается внутри блока if, а код операции ввода - внутри блока else.

В программе MiniDraw класс документа хранит единственную переменную m_LineArray, управляющую множеством объектов класса CLine. Переменная m_LineArray имеет собственную функцию-член Serialize (наследуемую от класса CObArray), которая вызывается для чтения или записи всех объектов класса CLine, хранимых в данной переменной. В результате функцию CMiniDrawDoc::Serialize можно дописать, просто добавив два обращения к функции CObArray::Serialize:

```
void CMiniDrawDoc::Serialize(CArchive& ar)
{
   if(ar.IsStoring())
   {
      //TODO: здесь добавьте код сохранения
      m_LineArray.Serialize(ar);
   }else
   {
      // TODO: здесь добавьте код загрузки
      m_LineArray.Serialize (ar);
}
```

При записи данных в файл функция CObArray:Serialize выполняет два основных действия для каждого объекта класса Cline:

1) записывает в файл информацию о классе объекта;
2) вызывает функцию Serialize объекта, записывающую данные объекта в файл.

При чтении данных из файла функция CObArray::Serialize выполняет два действия для каждого объекта класса Cline:

1) читает информацию класса из файла, динамически создает объект соответствующего класса (CLine) и сохраняет указатель на объект;

2) вызывает функцию Serialize объекта для чтения данных из файла во вновь созданный объект.

Необходимо обеспечить поддержку сериализации класса CLine. Для этого включите в его определение два макроса (DECLARE_SERIAL, а также IMPLEMENT_SERIAL) и определите конструктор класса по умолчанию. Эти макрокоманды и конструктор позволяют функции CObArray::Serialize сохранить информацию класса в файле, а затем использовать ее для динамического создания объекта класса. Макрокоманды и конструктор обеспечивают выполнение первого пункта двух приведенных выше алгоритмов с помощью функции CObArray::Serialize.

Добавьте макрокоманду DECLARE_SERIAL и конструктор по умолчанию в определение класса CLine в файле MiniDrawDoc.h как показано ниже:

```
    class CLine : public CObject
{
protected:
      int m_Xl, m_Yl, m_X2, m_Y2; CLine(){}
      DECLARE_ SERIAL (CLine)
public:
   оставшаяся часть определения класса CLine
}
```

Имя класса передается макросу DECLARE_SERIAL как параметр. В файле MiniDrawDoc.cpp макрокоманда IMPLEMENT_SERIAL добавляется сразу перед определением функции CLine::Draw:

```
    CMiniDrawDoc IMPLEMENT_SERIAL (CLine, CObject, 1)
void CLine::Draw(CDC *pDC)
{
   pDC->MoveTo (m_XI, m_Yl) ;
    pDC->LineTo (m_X2, m_Y2);
}
```

Первый параметр, переданный макросу **IMPLEMENT_SERIAL**, - это имя самого класса, а второй – имя базового класса. Третий параметр – это номер версии, идентифицирующий отдельную версию программы. Он сохраняется внутри записанного файла, прочитать который может только программа, указавшая такой же номер. Номер версии предотвращает чтение данных программой другой версии. Для текущей версии MiniDraw задан номер 1. В более поздних версиях он увеличивается при каждом изменении формата данных. Нельзя задавать номер версии -1.

Второе действие, необходимое для сериализации класса CLine, - это добавление в класс CLine функции Serialize, вызываемой в методе CObArray::Serialize для чтения или записи данных каждой строки. Добавьте объявление функции Serialize в раздел public определения класса CLine в файле MiniDrawDoc.h:

```
    public:
  CLine (int X1, int Y1, int X2, int Y2) {
    m_X1=X1; m_Y1=Y1; m_X2=X2; m_Y2=Y2;
  }
  void Draw (CDC *pDC);
  virtual void Serialize(CArchivet ar);
```

Добавьте определение функции Serialize в файл реализации с именем MiniDrawDoc.cpp после определения CLine::Draw:

```
    void CLine::Serialize(CArchive& ar)
{
  if(ar.IsStoring()
        ar«m_X1«m_Y1«m_X2«m_T2;
  else
        ar»m_X1»m_Y1»m_X2»m_Y2;
}
```

Класс, объекты которого могут быть сериализованы, должен прямо или косвенно порождаться от MFC-класса CObject.

Фактически операции чтения и записи выполняет функция CLine::Serialize, а не одноименные функции других классов. Функция Serialize использует перегруженные операторы “«” и “»” для записи переменных класса CLine в файл и для чтения их из файла соответственно. Эти операторы определяются классом CArchive и используются для чтения и записи данных различных типов.

Как видно из кода ввода-вывода, добавленного в программе MiniDraw, объект класса, сохраняющий данные, обычно отвечает за их запись на диск и чтение с диска. Этот класс должен предоставлять функцию Serialize, обеспечивающую чтение и запись. Общий принцип объектно-ориентированного программирования состоит в том, что каждый объект работает с собственными данными. Например, объект может нарисовать, прочитать, сохранить себя или выполнить другие характерные операции с собственными данными. Для переменных, являющихся объектами класса, вызывается функция-член Serialize их собственного класса. Для переменных, не являющихся объектами, функция Serialize использует перегруженные операторы “«” и ”»”, предоставляемые классом CArchive.

Установка флага изменений. Класс CDocument поддерживает флаг изменений, показывающий, содержит ли документ несохраненные данные. MFC проверяет этот флаг перед вызовом функции DeleteContents класса документа для удаления данных. MFC вызывает функцию DeleteContents перед созданием нового документа, открытием уже существующего или выходом из программы. Если флаг содержит значение TRUE (в документе имеются несохраненные данные), то выводится соответствующее сообщение.

Класс CDocument устанавливает значение флага в FALSE, когда документ открыт и сохранен. Для установки флага в TRUE (при каждом изменении

документа) вызывается функция CDocument::SetModifiedFlag. Добавьте функции SetModifiedFlag в функцию AddLine в файле MiniDrawDoc.cpp:

```
void CMiniDrawDoc::AddLine(int X1, int Y1, int X2, int Y2)
{
    CLine *pLine=new CLine(X1, Y1, X2, Y2);
    m_LineArray.Add(pLine);
    SetModifiedFlag();
}
```

Теперь проделайте то же самое для функции OnEditClearAll в том же файле:

```
void CMiniDrawDoc::OnEditClearAll()
{
    //TODO: Здесь добавьте собственный код обработчика
    DeleteContents();
    UpdateAllViews(0);
    SetModifiedFlag();
}
```

И, наконец, добавьте вызов функции SetModifiedFlag в обработчик OnEditUndo в файле MiniDrawDoc.cpp:

```
void CMiniDrawDoc::OnEditUndo()
{
    // TODO: Здесь добавьте собственный код обработчика
    int Index=m_LineArray.GetUpperBound();
    if(Index>-1){
       delete m_LineArray.GetAt (Index);
       m_LineArray.RemoveAt (Index);
    }
     UpdateAllViews(0);
     SetModifiedFlag();
}
```

Так как параметр функции SetModifiedFlag имеет стандартное значение TRUE, флаг изменений можно установить в TRUE, вызывая эту функцию без аргументов. Чтобы установить флаг изменений в FALSE, необходимо явно передать это значение (обычно эту задачу выполняет класс CDocument).

Поддержка технологии "drag-and-drop". Если программа поддерживает традиционную технологию "drag-and-drop", можно открывать файл, перемещая объект файла из папки Windows (а также из окна программы Explorer или любого другого окна, поддерживающего эту операцию) и отпуская его в окне программы.

Для поддержки операции перетаскивания в программе MiniDraw вызовите функцию CWnd::DragAcceptFiles для объекта главного окна. Поместите это обращение внутри функции Initlnstance класса приложения в файле MiniDraw.cpp после вызова UpdateWindow:

```
BOOL CMiniDrawApp::InitInstance()
{
    // другие операторы
     if(!ProcessShellCommand(cmdInfo))
        return FALSE;
     m_pMainWnd->ShowWindow(SW_SHOW);
     m_pMainWnd->UpdateWindow();
     m_pMainWnd->DragAcceptFiles();
     return TRUE;
}
```

Объект приложения содержит переменную m_pMainWnd (определенную в классе CWinThread, базовом для CWinApp), являющуюся указателем на объект главного окна. Функция InitInstance использует этот указатель для вызова функции DragAcceptFiles. Обращение к ней помещается после вызова функции ProcessShellCommand, так как внутри последней создает главное окно и присваивает значение переменной m_pMainWnd.

После вызова функции DragAcceptFiles (когда пользователь отпускает перетаскиваемый значок файла) MFC автоматически открывает файл, создает объект класса CArchive и вызывает функцию Serialize. Следовательно, для поддержки операции перетаскивания писать дополнительный код не нужно.

Регистрация типа файла. В системный реестр Windows следует добавить информацию, позволяющую открыть файл программы MiniDraw (т.е. файл с расширением .drw), выполняя двойной щелчок на файле в папке Windows или в окне программы Explorer (или любом другом окне, поддерживающем указанную операцию). Для этого вызовите функции EnableShellOpen и RegisterShellFileTypes класса CWinApp из определения функции InitInstance в файле MiniDraw.cpp:

```
BOOL CMiniDrawApp::InitInstance()
{
    // другие операторы ...
     AddDocTemplate (pDocTemplate);
     EnableShellOpen();
     RegisterShellFileTypes ();
     // Анализ командной строки с целью поиска команд
     // оболочки, DDE, открытия файлов
     CCommandLineInfo cmdInfo; // другие
     //операторы ...
}
```

Эти функции создают в реестре Windows связь между стандартным расширением файла программы MiniDraw (.drw) и самой программой. Объект, представляющий любой файл с этим расширением, отображает значок программы MiniDraw, а двойной щелчок на объекте запускает программу MiniDraw, если она еще не запущена, и открывает файл в этой программе.

Такая связь остается в реестре до тех пор, пока она не будет изменена явным образом.

Заметьте! Обращения к функциям EnableShellOpen и RegisterShellFileTypes помещаются после вызова функции AddDocTemplate, добавляющей шаблон документа в приложение, чтобы информация о стандартном расширении файла и типе документа была доступна объекту приложения (стандартное расширение вводится в строковый ресурс с идентификатором IDR_MAINFRAME, который передается в шаблон.)

Задание: дополнить программу, разработанную в предыдущей лабораторной работе, возможностями сохранения и восстановления данных.

ЛАБОРАТОРНАЯ РАБОТА №3
«СТАНДАРТНЫЕ ЭЛЕМЕНТЫ УПРАВЛЕНИЯ»

Цели работы:

1) создать диалог и его класс. При помощи редактора ресурсов установить на диалог надпись (static text), текстовое поле (edit box), кнопку (button). В заготовке каждого диалогового окна уже присутствуют кнопки “OK” и “Cancel”, но можно добавить еще и свои – столько, сколько посчитаете нужным, флажок (check box), переключатель-радиокнопку (radio button), список (list box). Создать переменные для элементов управления;

2) создать обработчики событий для элементов интерфейса при помощи ClassWizard.

В работе необходимо реализовать:

1) вызов диалога при старте приложения;

2) определение статуса закрытия диалога (нажата кнопка “OK” или “Cancel”);

3) вывод сообщения (в MessageBox) в зависимости от состояния флажка и переключателя-радиокнопки;

4) изменение выводимого сообщения (в MessageBox) в зависимости от состояния флажка и переключателя-радиокнопки.

Методические указания

На первом этапе создадим SDI-приложение, а на втором этапе – диалоговое окно. Процесс создания ресурсов для диалоговых окон подробно рассмотрен в приложении 2.

Установите элементы управления на поверхности диалога. Для этого щелкните мышью на пиктограмме соответствующего элемента управления, а затем щелкните на форме в том месте, где будет расположен этот элемент. Если посчитаете нужным, "ухватитесь" указателем мыши за один из размерных маркеров вокруг изображения элемента. С помощью его передвижения можно изменять границы размеров элемента.

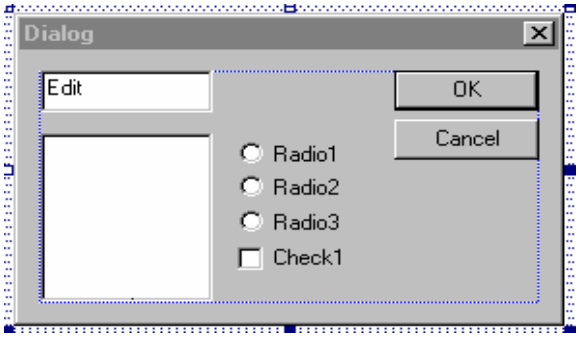

Рис. 3.1. Диалоговое окно с элементами управления

Добавьте элемент типа флажок и три переключателя в диалоговое окно, чтобы оно приняло вид, изображенный на рис. 3.1. В поле Caption измените надписи для переключателей. Пусть это будут One (один), Two (два) и Three (три). Чтобы выровнять включенные в окно элементы (выстроить их в колонку), выберите один из них, а затем, нажав и удерживая клавишу <Ctrl>, выберите по очереди остальные. После этого выберите в меню Lay-Out\Align Control Left, затем выберите команду Lay-Out\Space Evenly\Down (Размещение\Подравнять интервал\Вниз). Эта команда позволяет установить одинаковый интервал между элементами по вертикали.

Выберите переключатель One, вызовите окно Dialog Properties и установите в нем флажок Group (Группа). Такая установка означает, что переключатель One является первым элементом группы переключателей.

Теперь добавьте в формируемое окно еще и элемент типа список. Установите его справа от переключателей и измените размеры в соответствии с заданием. В то время, когда этот элемент еще остается выбранным, с помощью команды View=> Properties вызовите на экран окно Dialog Properties. Выберите вкладку Styles (Стили) и проверьте, не установлен ли флажок Sort (Сортировка). Если этот флажок установлен, то при выполнении программы элементы в списке будут отсортированы по алфавиту.

Краткая характеристика основных Windows-элементов управления, используемых для построения диалога:

- **надпись** (static text) – по существу, это "неполноценный" элемент управления, поскольку он используется только как поле для вывода надписи, относящейся к "настоящему" элементу управления, расположенному рядом;
- **текстовое поле** (edit box) – может быть однострочным или многострочным; сюда пользователь может ввести текст;
- **кнопка** (button) – данный элемент предназначен для начала каких-либо действий после нажатия на неё;
- **флажок** (check box) – используется для установки опций, каждая из которых может быть выбрана независимо от других;
- **переключатель-радиокнопка** (radio button) – используется для выбора одной из групп связанных опций; если выбрана одна из них, то другие полагаются невыбранными;
- **список** (list box) – используется для выбора одного элемента из заранее подготовленного набора; набор может быть как жестко установленным на этапе разработки программы, так и меняться программно в процессе выполнения приложения; главное – пользователь по своей воле не может непосредственно менять элементы в наборе, он может только их выбирать;
- **поле со списком (Combo box)** – это комбинация текстового поля и списка; такой элемент управления позволяет пользователю не только выбирать элементы из ранее подготовленного набора, но и самостоятельно пополнять его, непосредственно внося необходимый текст в текстовое поле.

Задание идентификаторов диалогового окна и элементов управления. Поскольку каждое диалоговое окно в приложении является уникальным объектом (исключение составляют только стандартные окна, о которых речь будет

идти в последующих главах), разработчику практически всегда нужно присваивать окнам и элементам управления, входящим в их состав, идентификаторы по собственному выбору. Конечно, можно согласиться и с теми идентификаторами, которые предлагает редактор диалоговых окон по умолчанию. Они не несут смысловой нагрузки (как правило, нечто вроде IDD_DIALOG1, IDC_EDIT1, IDC_RADI01) и их можно заменить другими, связанными с назначением и функциями окна или элемента. Но в любом случае рекомендуется соблюдать соглашение о префиксах: идентификаторы диалоговых окон имеют префикс IDD_, а идентификаторы элементов управления – **IDC_**. Заменить идентификатор можно с помощью диалогового окна Dialog Properties. Для этого выберите элемент управления или диалог и Edit Properties, если ранее окно Dialog Properties не было выведено и закреплено на экране. Затем измените идентификатор ресурса в поле ID, но при этом не забывайте о префиксах: IDD_ – для диалоговых окон и IDC_ – для элементов управления.

Создание ассоциированных переменных. Ассоциированная переменная класса диалогового окна задается в ClassWizard на закладке Member Variables (рис.3.2) и соответствует либо значению - содержимому элемента управления, либо объекту класса соответствующего данному типу элемента управления. Пример, представленный на рис. 3.3, демонстрирует один из вариантов ассоциативной связи. Элементу IDC_CHECK1 следует присвоить идентификатор переменной m_check. Нужно проверить, чтобы в раскрывающемся списке Category (Категория) было выбрано Value (Значение). Если вы раскроете список Variable type (Тип переменной), то увидите, что вам предоставлен единственный "свободный" выбор – BOOL. Флажок может быть либо установлен, либо сброшен, а значит, ассоциирован только с переменной типа BOOL, которая принимает только два значения – TRUE и FALSE. Нажмите на OK для завершения процедуры.

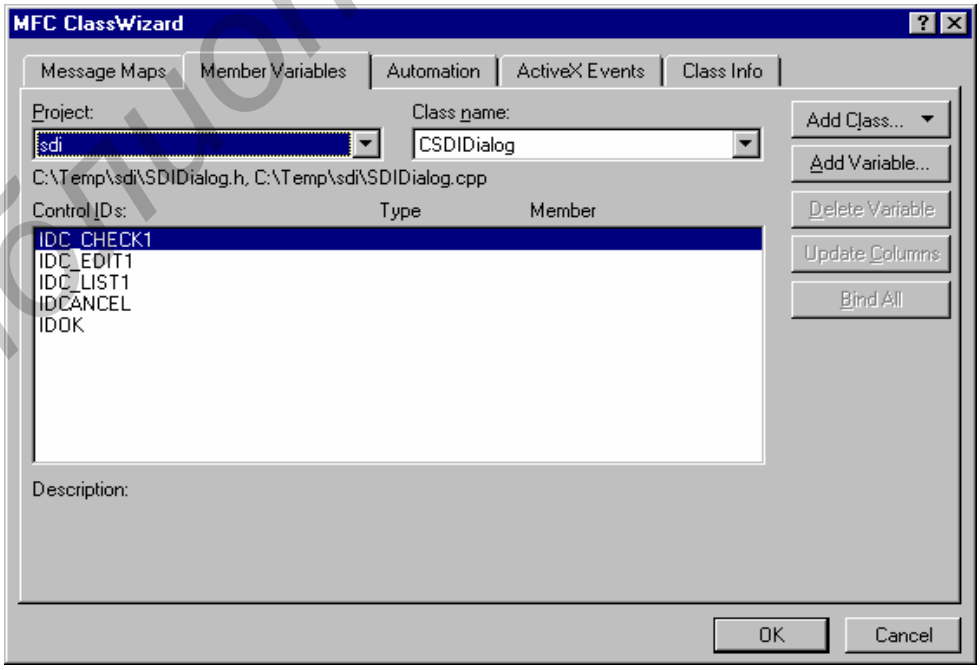

Рис.3.2. Закладка Member Variables и элементы управления

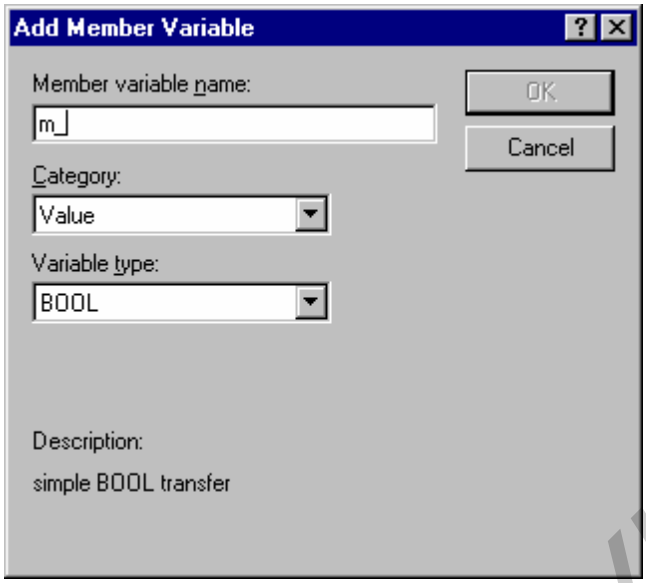

Рис. 3.3. Диалог установки идентификатора члена-переменной класса, ассоциированного с некоторым элементом управления

Ниже перечислены типы переменных, которые могут быть ассоциированы с тем или иным типом элемента управления:

- текстовые поля – как правило, строковый тип(CString), но иногда и другие – int, fIoat и Iong;
- кнопки – int;
- флажки – BOOL;
- переключатели – int;
- список – строковый тип;
- поле со списком – строковый тип;
- полоса прокрутки – int .

Свяжите таким же образом значение, которое содержится в элементе IDC_EDIT1, с членом-переменной m_edit типа CString. Элемент IDC_LIST1 должен быть связан с членом-переменной m_list, который должен быть объектом класса CListBox (в списке Category должно быть избрано Control). Первый переключатель в группе IDC_RADI01 должен быть связан с членом-переменной m_radio типа int, причем связь должна быть установлена по значению.

После добавления переменных ClassWizard предложит установить параметры, которые могут быть использованы для проверки достоверности ввода данных. Это делается не для всех видов переменных. Но, например, если речь идет о переменной, связанной с текстовым полем, ClassWizard предлагает в поле Maximum Characters (Максимум символов) установить максимальную длину вводимой строки (рис.3.4.). Для члена-переменной m_edit установите значение этого параметра равным 10. Если текстовое поле ассоциировано с переменной типа int или float, ClassWIzard использует эту же часть окна для установки верхнего и нижнего пределов вводимого пользователем значения. В дальнейшем всю работу по проверке соответствия введенного значения установленным ограни-

чениям и выдачу в случае их нарушения сообщения с просьбой повторить ввод берут на себя функции из библиотеки MFC.

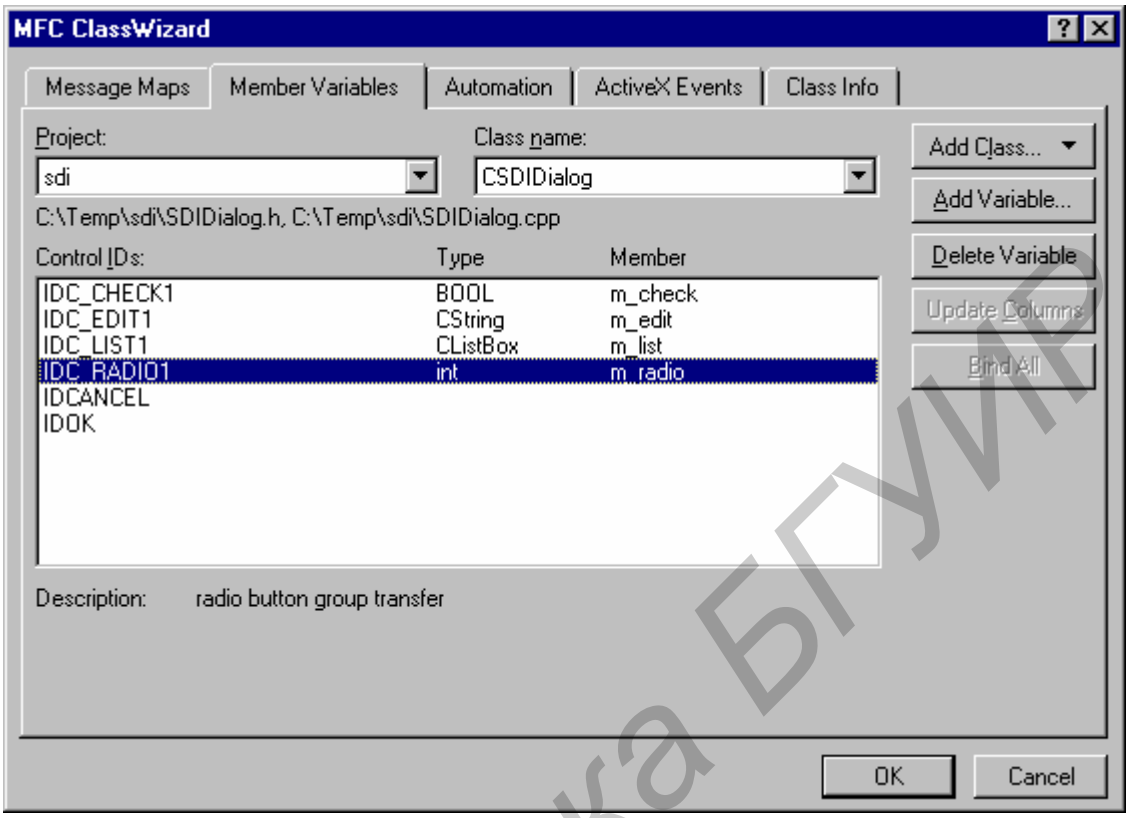

Рис.3.4.Выбор элементов управления для установки параметров проверки достоверности ввода данных

В классе, связанном с диалогом, присутствует функция OnOK(), которая унаследована от базового класса CDialog, классом-наследником которого является наш диалог. Помимо прочего в нем находится функция DoDataExchange(), подготовленная средствами ClassWizard. Вот как она выглядит в настоящий момент:

```
void  CSDIDialog::DoOataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CSdiDiaIog)
    DDX_Control(pDX, IDC_LIST1, m_list);
    DDX_Check(pDX, IDC_CHECK1, m_check);
    DDX_Text(pDX, IDC_EDIT1, m_edit);
    DDV_MaxChars(pDX, m_edit, 10);
    DDX_Radio(pDX, IDC_RADI01, m_radio);
    //}}AFX_DATA_MAP
}
```

Все функции, имена которых начинаются с DDX, выполняют обмен данными. Вторым аргументом каждой функции является идентификатор элемента

управления, а третьим – переменная класса. Именно таким образом CIassWizard устанавливает соответствие между элементами управления и членами класса диалогового окна – это CIassWizard подготовил такой текст программы вместо вас.

Имеются 34 функции, их имена начинаются с DDX – одна на каждый тип данных, которыми могут обмениваться диалоговое окно и соответствующий класс. Каждая функция включает в свое имя также имя элемента управления. Например, функция DDX_Check() используется для связи между элементом типа флажок (check box) и членом-переменной типа BOOL. Аналогично DDX_Text() используется для связи члена-переменной типа CString с текстовым полем. CIassWizard выбирает соответствующую функцию в процессе выполнения описанной выше операции связывания членов класса с элементами управления.

Существует несколько DDX-функций, за которые ClassWizard не несет ответственности. Например, если вы связываете список по значению с переменной, то единственным выбором для вас является тип CString. В этом случае ClassWizard формирует функцию DDX_LBString(), которая связывает выбранный в списке элемент с членом-переменной типа CString. Однако иногда эффективнее использовать индекс выбранного элемента, а не сам элемент. Для этого случая имеется функция DDX_LBlndex(), которая выполняет соответствующий обмен. Вызов этой функции можно добавить в текст функции-члена DoDataExchange. Соответствующая строка может быть вставлена в том месте программы, где имеется специальный комментарий, созданный ClassWizard. При этом не забудьте добавить соответствующую переменную-член в объявление класса в файле заголовка. Полный список DDX-функций можно найти в электронной документации.

Функции, имена которых начинаются с DDV, ответственны за проверку соблюдения заданных ограничений на вводимые данные. ClassWizard вставляет вызов DDV_MaxChars() сразу за вызовом DDX_Text, которая передает содержимое текстового поля IDC_EDIT1 в переменную m_edit. Второй аргумент вызова функции – идентификатор переменной-члена, а третий – значение параметра, ограничивающего длину вводимой строки. Если пользователь когда-нибудь при работе с программой попытается ввести символов больше, чем дозволено, то DDV_MaxChars() организует вывод на экран предупреждающего сообщения. Так что вы можете только указать величину ограничения и рассчитывать, что все дальнейшее будет организовано CIassWizard и MFC.

Организация вывода диалогового окна на экран. Выберите вкладку ClassView в рабочей зоне проекта, раскройте пункт SDI Classes, а в нем – CSDIView. Дважды щелкните на функции-члене InitInstance(). Эта функция вызывается при любом запуске приложения. Перейдите в самое начало файла и после уже имеющихся директив # include вставьте еще одну:

#include "sdidialog.h"

Теперь при трансляции компилятор будет знать, где взять информацию о классе CSDIDialog. Перейдите в конец текста функции CSDIView::Initlnstance() в файле SDIDialog.CPP и добавьте перед окончанием текста функции следующие строки:

```
CSDIDialog dlg;
dlg.m_check = TRUE;
 dlg.m_edit = "hi there";
 CString msg;
 if (dIg.DoModal) == IDOK) {
        msg = "You clicked OK. ";}
        else
{
        msg = "You can clicked Cancel. ";
}
msg += "Edit Box is: ";
msg += dig.m_edit;
AfxMessageBox(msg);
```

Приведенный выше фрагмент программы – экземпляр класса диалогового окна. Он устанавливает параметры по умолчанию для двух элементов управления – флажка и текстового поля. Сам по себе вывод диалогового окна производится функцией DoModaI(), которая возвращает числовое значение – IDOK, если пользователь вышел из окна, нажав на OK, и IDCANCEL, если выход произошел после нажатия на Cancel. Затем в приведенном фрагменте формируется сообщение, которое выводится на экран функцией AfxMessageBox().

Создание обработчиков событий нажатия на кнопки OK и CANCEL. Простейшим событием является нажатие на кнопку в диалоге. Для его создания можно просто дважды нажать левой клавишей мыши в окне ресурса по соответствующей кнопке. После этого на экране появится диалог (рис.3.5).

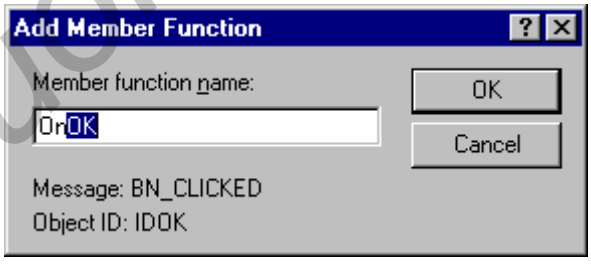

Рис.3.5. Диалог определения имени функции

По умолчанию ClassWizard предлагает очень удачное имя для этой функции обработки события.

Листинг: SDIDIALOG.CPP , функция SDIDiaIog::OnOK():

```
void CSDIDialog::OnOK()
{
 int index = m_list.GetCurSel();
```

```
    if (index != LB_ERR)
    {
        m_list.GetText(index, m_selected);}
    else {
        m_selected = "";
    }
    CDialog::OnOK();
}
```

Работа этого фрагмента программы начинается с вызова функции-члена класса объектов список GetCurSel(), которая возвращает константу LB_ERR в случае, если не выбран ни один элемент списка или если выбрано более одного элемента. Иначе возвращается индекс выбранного элемента. Функция GetText() (член того же класса) переписывает строку выбранного элемента в переменную m_selected класса CSDIDialog. Первым аргументом функции является индекс выбранного элемента. После этого вызывается функция OnOK() базового класса CDialog, которая выполняет все стандартные действия по закрытию окна.

Сейчас можно скорректировать текст функции CSDIApp::Initlnstance() для того, чтобы в выведенном после закрытия окна сообщении было упомянуто, какой выбор сделал пользователь в списке. Эти строки программы будут выполняться независимо от того, каким образом было закрыто окно – нажал пользователь на ОК или на Cancel. Сначала нужно создать дополнительно функцию, которая обрабатывала бы щелчок на Cancel. Такая функция – OnCanceI() – создается точно таким же образом, как и OnOK(), но в правом списке Class or object to handle выделите IDCANCEL и согласитесь с именем функции OnCancel() Как видно из листинга сформированной функции, переменная m_selected очищается, поскольку пользователь отказался от диалога с программой.

Листинг: SDIDIALOG.CPP, функция CSDIDialog: :OnCancel():

```
void  CSDIDialog::OnCanceI()
{
    m_selected = "";
        CSDIDdialog::OnCancel();
}
```

В функцию CSDIView::InitInstance() добавьте следующие строки перед обращением к функции AfxMessageBox():

```
msg += ". List Selection: ";
msg += dig. Deselected;
```

Для выбора одного из переключателей в группе диалогового окна необходимо добавить следующий текст в функцию CSDIDialog::0nlnitDialog():

```
m_radio = 1;
UpdateData(FALSE);
```

Членом-переменной является m_radio, с которым связана группа переключателей. Она (переменная) представляет собой индекс выбранного переключателя в этой группе элементов управления (как всегда, индекс начинается с 0).

Значение индекса 1 соответствует второму переключателю в группе. Вызов функции UpdateData() в этом фрагменте обновляет содержимое элементов управления диалогового окна соответственно состоянию связанных с ними переменных-членов. Аргумент функции UpdateData() указывает направление передачи данных: UpdateData(TRUE) обновило бы содержимое переменных соответственно элементам управления, т.е. переписало бы в m_radio индекс выбранного в группе переключателя.

В отличие от списка группа переключателей доступна и после того, как диалоговое окно убрано с экрана. Так что вам не придется добавлять что-либо в функции OnOK() и OnCanceI(). Вместо этого у вас будет другая проблема – как преобразовать целое значение индекса в строковое выражение, которое нужно будет добавить в "хвост" текста сообщения в переменную msg. Существует множество ее решений, включая функцию-член Format() класса CString, но в данном случае можно поступить гораздо проще – использовать оператор switch, поскольку индекс может принимать лишь ограниченное множество значений. В конец текста функции CSDIApp::Initlnstance(), перед вызовом AfxMessageBox(), добавьте несколько строк, представленных в листинге.

Листинг функции CSDIApp::Initlnstance():

```
        void CSDIApp::Initlnstance()
{
        msg += "\r\n";
        msg += "Radio Selection: ";
        switch (dig.m_radio)
        {
        case 0:
        msg += "O";
        break;
        case 1:
        msg += "1";
        break;
        case 2:
        msg += "2";
        break;
        default:
        msg += "none";
        break;
        }
}
```

Первая из новых строк добавляет в сообщение два специальных символа – перевод каретки \r и перевод строки \n, которые в совокупности представляют маркер конца строки Windows. В результате дальнейшая часть сообщения msg начнется с новой строки.

Запуск и компиляция приложения. Запустите процесс компиляции и компоновки проекта, выбрав команду Build=>Build или щелкнув на пиктограмме Build (Построить) панели инструментов Build. Запустите выполнение приложения, воспользовавшись командой Build\Execute (Построить\Выполнить) или щелкнув на пиктограмме Execute (Выполнить) панели инструментов Build. Вы увидите, что на экране появилось диалоговое окно с параметрами элементов управления, установленными по умолчанию в программе, которую вы только что отредактировали.

После загрузки созданного вами приложения поменяйте что-либо в текстовом поле (например, введите текст hello), а затем щелкните на OK. После этого на экране должно появиться окно сообщения.

Снова запустите приложение, отредактируйте текст в поле и выйдите из окна, щелкнув на Cancel. Обратите внимание на сообщение, гласящее, что в поле остался исходный текст hello. Это получилось потому, что MFC не дублирует содержимое текстового поля (как элемента управления) в переменную-член в случае, если пользователь щелкает на Cancel для выхода из окна. И снова, как и в предыдущем эксперименте, закройте приложение.

После того, как вы щелкнете на OK, приложение выведет в окне сообщения копию текста, введенного в текстовом поле. Надпись в окне гласит: *B"You clicked OK. Edit Box is: hello"*.

Если вы щелкнете на Cancel, приложение проигнорирует любые изменения элементов управления. Надпись в окне гласит: "*You can clicked Cancel. Edit Box is: hello"*.

Варианты задания:

1.Разработать приложение управления двумя списками, расположенными на диалоге горизонтально. Приложение должно обеспечивать перемещение любого выбранного элемента или содержимого всего списка в другой список, как из левого – в правый, так и из правого – в левый. Элемент при перемещении должен исчезать из одного списка и появляться в другом. Помимо того приложение должно обеспечивать управление левым списком – добавление нового элемента, редактирование, удаление.

2.Разработать приложение, обеспечивающее возможность множественного выбора элементов из списка. Выбранные элементы должны образовывать строку текста и помещаться в строку редактирования. Предусмотреть возможность вывода сообщения в случае, если суммарное количество символов будет превышать 100.

3.Разработать приложение, реализующее калькулятор. Приложение должно иметь строку редактирования, набор кнопок 0…9, кнопки арифметических действий – суммирование, вычитание, деление, умножение.

4.Разработать приложение, реализующее калькулятор. Приложение должно иметь две строки редактирования. Набор переключателей-радио кнопок определяет, какое арифметическое действие необходимо выполнить: суммирование, вычитание, деление, умножение.

5.Разработать приложение, обеспечивающее поиск в списке (list box) фрагмента текста. Строки, в которых будет найден искомый фрагмент, должны

быть выделены (предполагается, что несколько строк может иметь искомый фрагмент). Помимо этого приложение должно обеспечивать управление содержимым списка – добавление нового элемента, редактирование, удаление.

6.Разработать приложение генерации счетчика 100 случайных чисел и вывод их в отсортированном по убыванию порядке. Обеспечить возможность расчета суммы трех наибольших чисел и трех наименьших. Суммируемые числа выделить.

7.Разработать приложение управления двумя списками, расположенными на диалоге горизонтально. Приложение должно обеспечивать перемещение любого количества выбранных элементов. Элемент при перемещении не исчезает, а выделяется. Помимо этого приложение должно обеспечивать заполнение левого списка 10 строками при запуске приложения.

8. Разработать приложение управления списком. На диалоге установлено два флажка (check box). При первом включенном флажке осуществляется выбор всех нечетных строк, при втором включенном флажке осуществляется выбор всех четных строк.

ЛАБОРАТОРНАЯ РАБОТА №4
«РАЗРАБОТКА ПРИЛОЖЕНИЯ УПРАВЛЕНИЯ БАЗОЙ ДАННЫХ»

Цель работы – ознакомиться с процессом создания Win32-приложений, поддерживающих работу с базами данных.

Задачи:

1) создание приложения, отображающего данные из БД;
2) обеспечение удаления, редактирования, добавления записей в БД;
3) обеспечение сортировки и фильтрации записей из БД.

Методические указания

Создание экранной формы для отображения содержимого базы данных. Пусть наша программа называется DB. Как указано в приложении 5 создайте заготовку программы. Затем вам необходимо модифицировать диалог, связанный с производным классом от CRecordView, предназначенным для отображения данных в окне приложения. Поскольку этот диалог является просто специализированным типом диалогового окна, связанного с базой данных, модификацию можно осуществить с помощью редактора ресурсов Visual Studio:

1. Для отображения ресурсов приложения щелкните на корешке вкладки ResourceView. Разверните дерево ресурсов, щелкнув на знаке "+'" перед папкой DB Resources. Далее, откройте папку ресурсов Dialog и сделайте двойной щелчок на идентификаторе диалогового окна IDD_DB_FORM (AppWizard генерирует диалог IDD_DB_FORM) и тем самым откройте диалог в редакторе ресурсов.

2. Выделите строку в центре диалогового окна, а затем удалите ее, нажав клавишу <Del>.

3. Пользуясь инструментами редактора диалогового окна, добавьте в него текстовые поля редактирования и статические надписи по образцу, показанному на рис. 4.1. Присвойте полям редактирования идентификаторы в соответствии с шаблоном: IDC_названиетаблицы_названиеполя (например для поля ID таблицы User IDC_USERID, а для поля FirstName – идентификатор IDC_USERFIRSTNAME). Для текстового поля, содержащего идентификатор IDC_USERID, установите стиль Read-OnIy (определяется одноименным флажком на вкладке Styles в окне свойств Edit Properties). Часто перед названием поля на этапе проектирования ставится название таблицы. Особенно это удобно, если поле с таким названием существует в нескольких таблицах.

4. Каждое из этих текстовых полей будет представлять собой поле записи базы данных. Атрибут Read-Only (только для чтения) установлен для первого (текстового) поля по той причине, что оно будет содержать первичный ключ базы данных, который не подлежит изменению.

5. Для вызова мастера ClassWizard выберите команду View\ClassWizard и в раскрывшемся окне щелкните на корешке вкладки Member Variables.

6. Выбрав ресурс IDC_USERFIRSTNAME, щелкните на кнопке Add Variable. Раскроется диалоговое окно Add Member Variable.

7. Щелкните на стрелке рядом с раскрывающимся списком Member Variable Name и выберите в нем значение m_pSet->m_UserFirstName (рис. 4.2).

8. Аналогично свяжите с элементами редактирования остальные переменные-члены (m_pSet->m_UserMiddleName, m_pSet->m_UserLastName и m_pSet->m_UserEMail). Когда это будет сделано, вкладка Member Variables окна MFC ClassWizard должна выглядеть так, как показано на рис. 4.3.

Выбрав переменные-члены класса приложения CDBSet (производного от класса MFC CRecordset) в качестве переменных для элементов управления в классе представления базы данных (в форме), вы установили связь, посредством которой может происходить обмен данными между элементами редактирования и источником данных.

9. После щелчка на кнопке OK в окне MFC ClassWizard внесенные изменения будут зафиксированы в тексте программы.

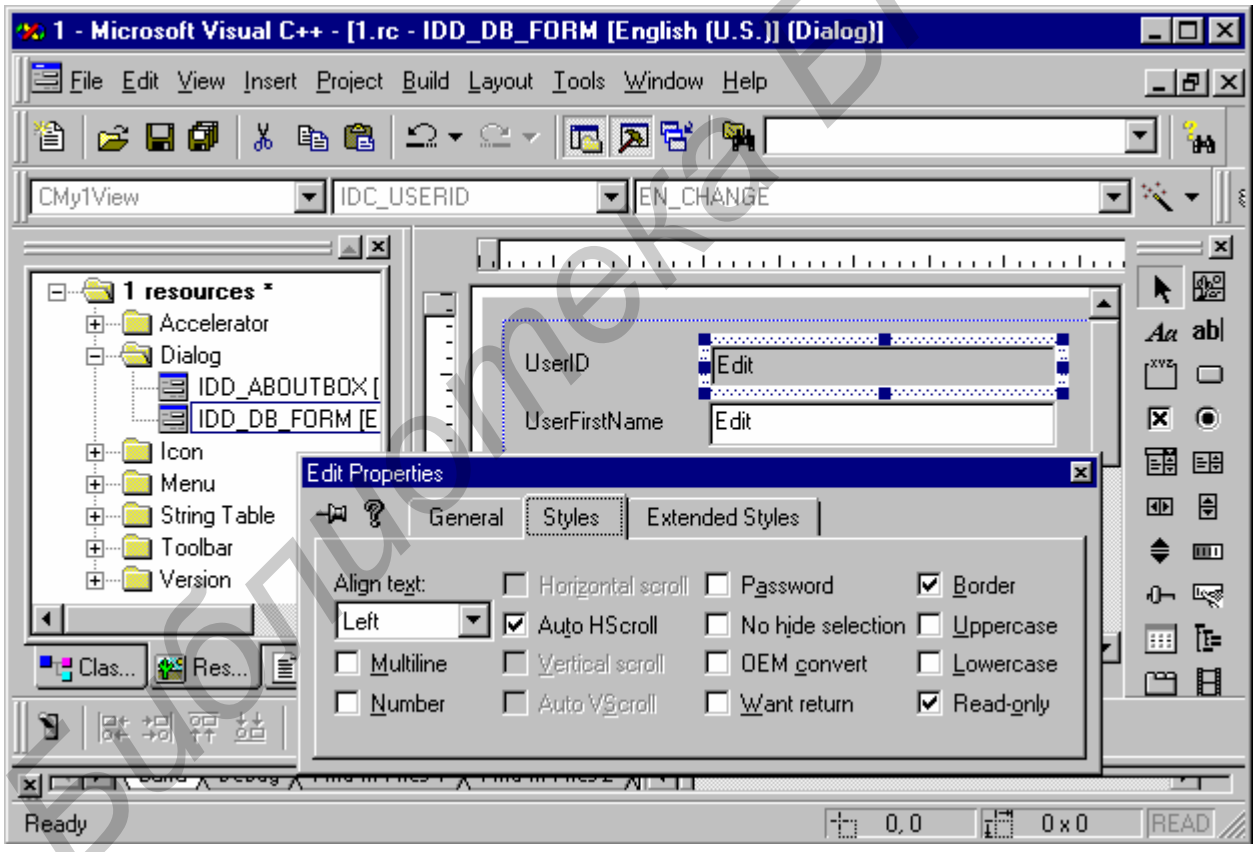

Рис.4.1. Создание диалогового окна, которое будет использоваться в качестве формы для базы данных

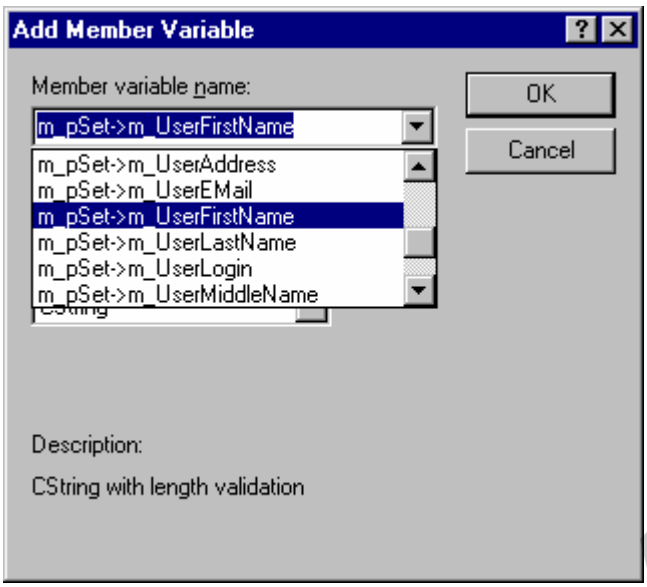

Рис. 4.2. Связывание поля IDC_USER _USERFIRSTNAME с переменной-членом m_UserFirstName класса выборки данных

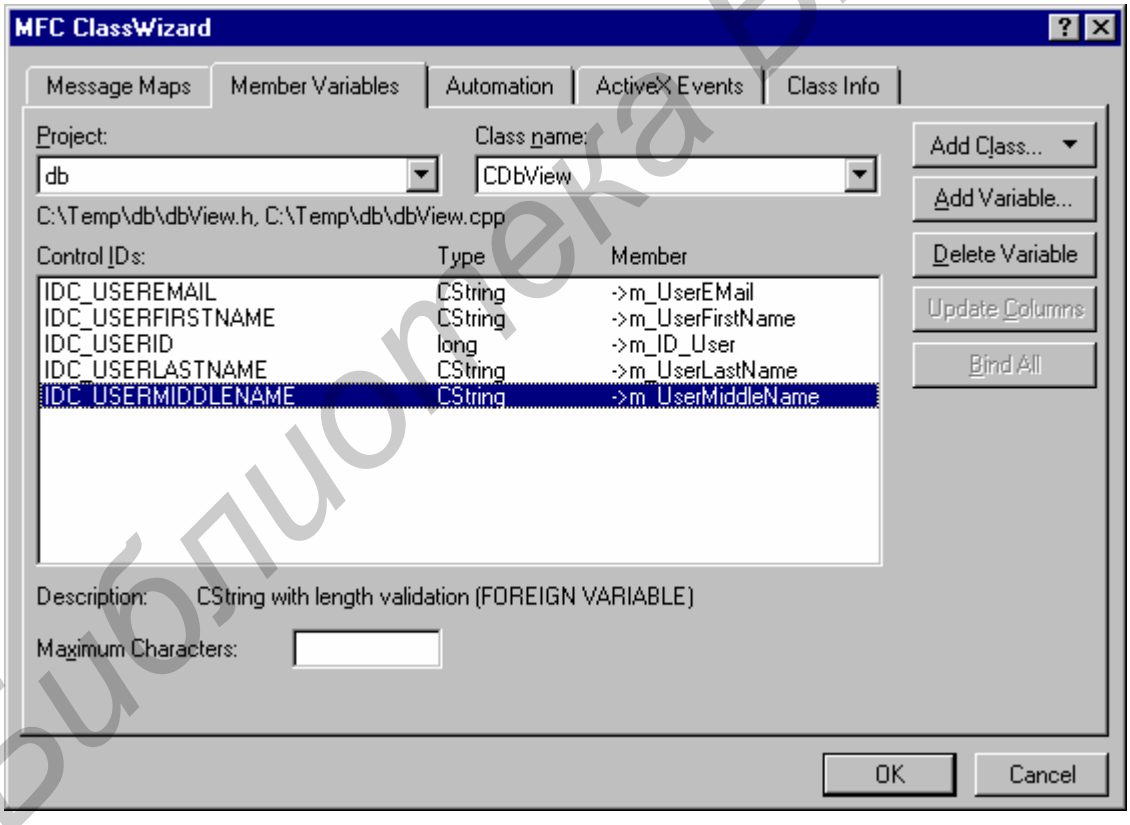

Рис. 4.3.Связь элементов управления с переменными-членами класса CDBSet

Мы завершили создание экранной формы для отображения данных в приложении Employee. Откомпилируйте и запустите программу и вы увидите окно, показанное на рис. 4.4. Приложение отображает содержимое записей таблицы

User. Используя элементы навигации, расположенные на панели инструментов, можно перемешаться от одной записи таблицы User к другой.

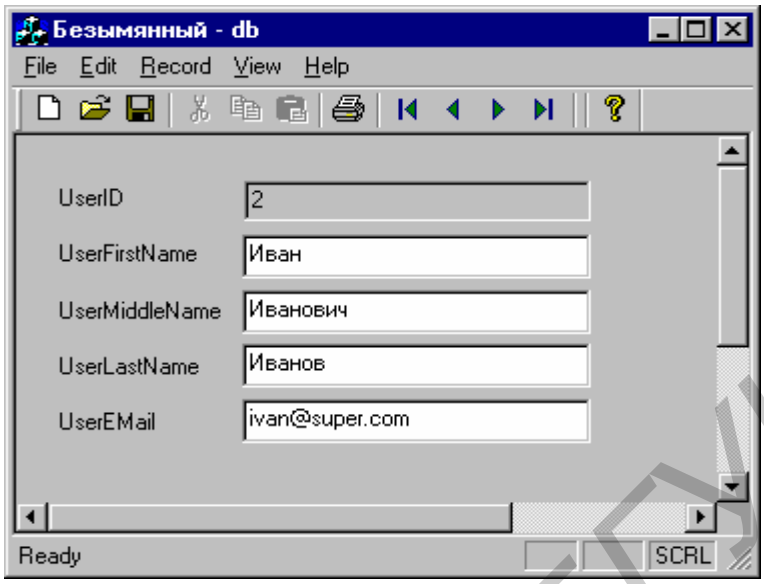

Рис. 4.4. Отображение в приложении данных из таблицы User

Проверив возможность перемещения в базе данных, попробуйте обновить любую из записей. Для этого достаточно просто изменить содержимое любого из полей записи (за исключением поля UserID, которое является первичным ключом и не может быть изменено). При переходе к другой записи приложение автоматически перенесет отредактированные данные в таблицу. Команды меню Record (Запись) приложения позволяют перемещаться по записям в базе данных точно так, как пиктограммы панели инструментов.

Добавление и удаление записей. После включения в создаваемое приложение возможности добавлять и удалять записи в таблице базы данных оно превратится в полнофункциональную программу обработки однофайловой (но не реляционной) базы данных. В нашем случае в роли однофайловой базы данных выступает таблица User реляционной базы данных виртуального магазина. Добавление и удаление записей в таблице базы данных реализуются достаточно просто благодаря существованию в Visual C++ классов CRecordView и CRecordset, предоставляющих все необходимые методы для выполнения этих стандартных операций. Необходимо будет добавить в приложение несколько команд меню (приложение 3). Для добавления к приложению команд Add (Добавить) и Delete (Удалить) выполните следующие действия:

1. Щелкните на корешке вкладки ResourceView, откройте папку Menu и сделайте двойной Щелчок на меню IDR_MAINFRAME. На экране раскроется окно редактора меню.

2. Щелкните в меню Record и тем самым откройте его, а затем щелкните на пустой области в нижней части этого меню. Выберите команду View\Properties и переместите раскрывшееся диалоговое окно Properties на подходящее для него место.

3. В поле ID введите значение ID_RECORD_ADD, а в поле Caption введите значение &Add Record. В результате в меню Record будет добавлена новая команда.

4. В следующий пустой элемент меню внесите команду удаления, имеющую идентификатор ID_RECORD_DELETE (поле ID) и заголовок Delete Record (поле Caption), как показано на рис.4.5.

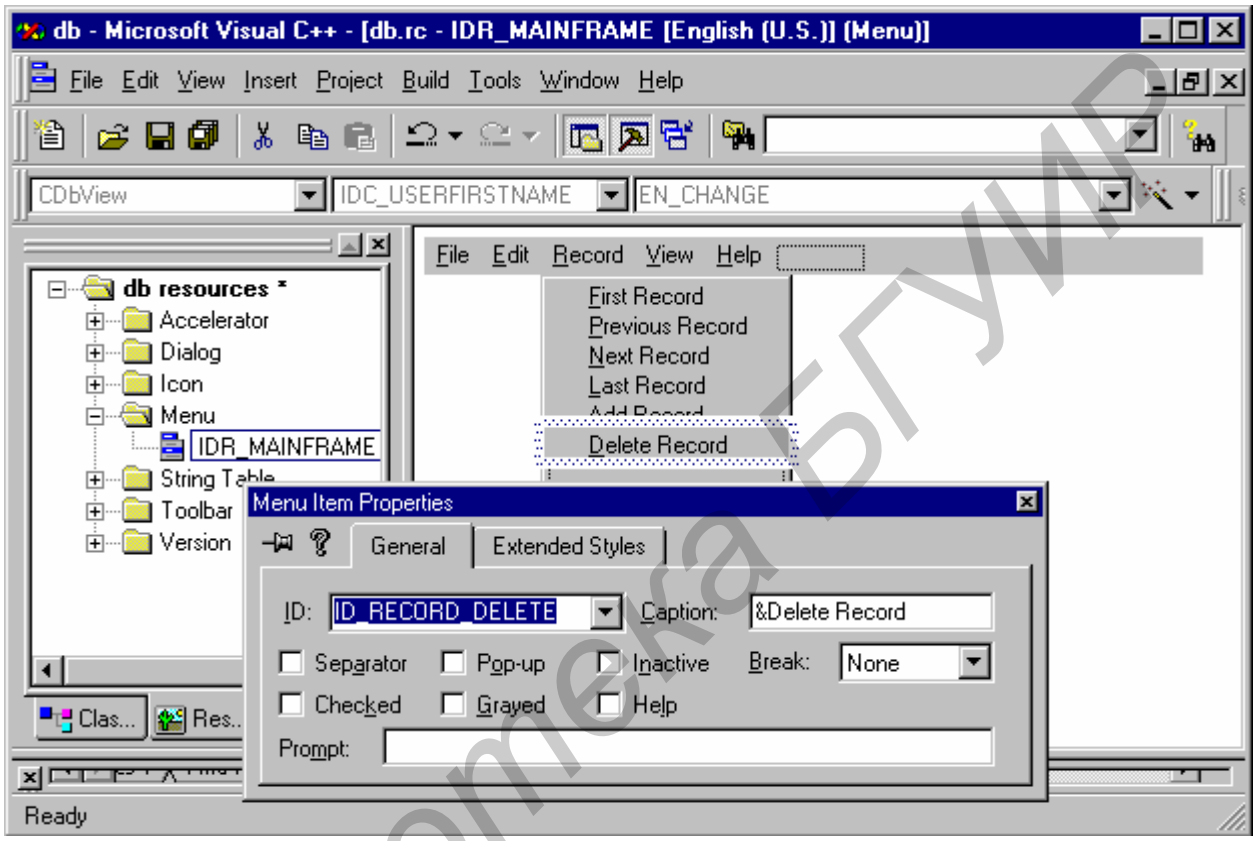

Рис. 4.5. Добавление в меню команд добавления и удаления записей

Далее необходимо добавить на панель инструментов пару новых пиктограмм (приложение 4) и связать с командами меню. Выполните следующие действия:

1. В дереве ресурсов в окне ResourceView откройте папку Toolbar и сделайте двойной щелчок на идентификаторе IDR_MAINFRAME. Панель инструментов приложения будет отображена в окне редактора ресурсов.

2. Щелкнув на пустой пиктограмме панели инструментов, выберите ее, а затем с помощью инструментов графического редактора нарисуйте на ней голубой знак "плюс", как показано на рис. 4.6.

3. Сделайте двойной щелчок на новой пиктограмме панели инструментов. Раскроется окно свойств Toolbar Button Properties. В списке ID выберите значение ID_RECORD_ADD.

4. Снова выделите пустую пиктограмму панели инструментов и нарисуйте на ней красный знак "минус", присвойте пиктограмме идентификатор ID_RECORD_DELETE аналогично “плюсу”. Перетащите пиктограммы добавления и удаления левее пиктограммы справки, помеченной знаком вопроса.

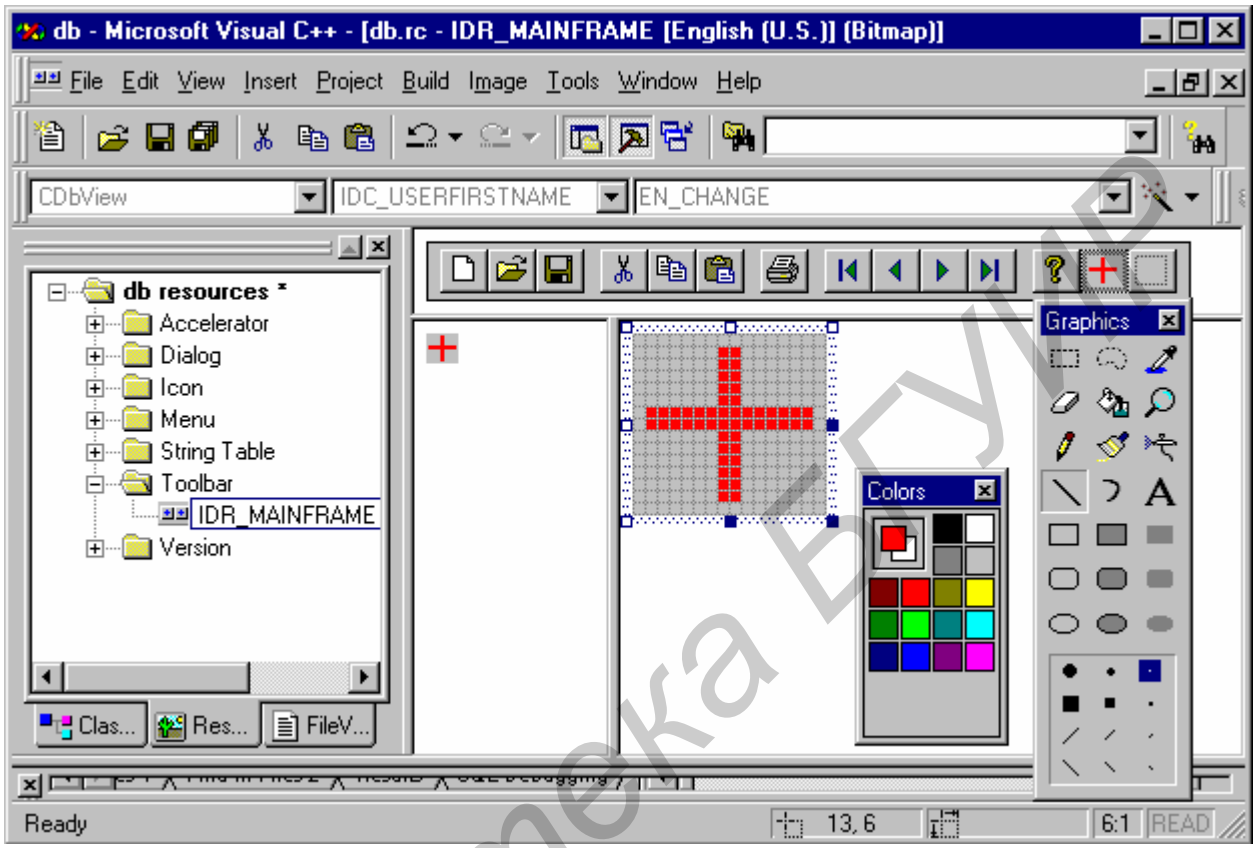

Рис. 4.6. Добавление пиктограммы на панель инструментов

Теперь, когда в меню уже добавлены новые команды и на панель инструментов помещены соответствующие пиктограммы, необходимо сформировать программный код, который будет обрабатывать командные сообщения, посылаемые, когда пользователь щелкает на пиктограмме или выбирает пункт меню. Так как в нашем приложении с базой данных связан класс представления, в нем следует организовать перехват этих сообщений. Выполните следующие операции:

1. Раскройте окно ClassWizard и выберите в нем вкладку Message Maps.

2. В списке Class Name выберите CDBView, а в списке Object IDS – значение ID_RECORD_ADD, после чего сделайте двойной щелчок на значении COMMAND в списке Messages. Раскроется диалоговое окно Add Member Function.

3. Щелкните на кнопке OK, приняв для новой функции имя, предложенное по умолчанию. Имя новой функции появится в списке Member Functions в нижней части окна ClassWizard.

4. Аналогичным образом добавьте метод для обработки команды ID_RECORD_DELETE. Закройте окно мастера CIassWizard, щелкнув на кнопке ОК.

5. В окне ClassView, дважды щелкнув на элементе CDBView, откройте файл DBView.h. В объявлении класса добавьте следующие строки в раздел Attributes:

```
protected:
        BOOL m_bAdding;
```

6. В окне ClassView сделайте двойной щелчок на конструкторе класса CDBView и добавьте следующую строку в конец этой функции:

```
m_bAdding = FALSE;
```

7. Сделайте двойной щелчок на функции OnRecordAdd() и отредактируйте ее текст так, как показано в листинге:

Листинг функции CDBView: :OnRecordAdd():

```
    void CDBView::OnRecordAdd()
{
    m_pSet->AddNew();
    m_bAdding = TRUE;
    CEdit* pCtrl = (CEdit*)GetDlgltem(IDC_USERID);
    int result = pCtrl->SetReadOnly(FALSE);
    UpdateData(FALSE);
}
```

8. В окне ClassView щелкните правой кнопкой мыши на элементе CDBView и выберите в раскрывшемся контекстном меню команду Add Virtual Function. В левом списке выберите значение OnMove, как показано на рис. 4.7, а затем щелкните на кнопке Add and Edit. В результате в класс будет добавлена функция и можно будет немедленно отредактироват заготовку ее текста.

9. Отредактируйте функцию OnMove() так, чтобы она содержала текст программы, приведенный в листинге.

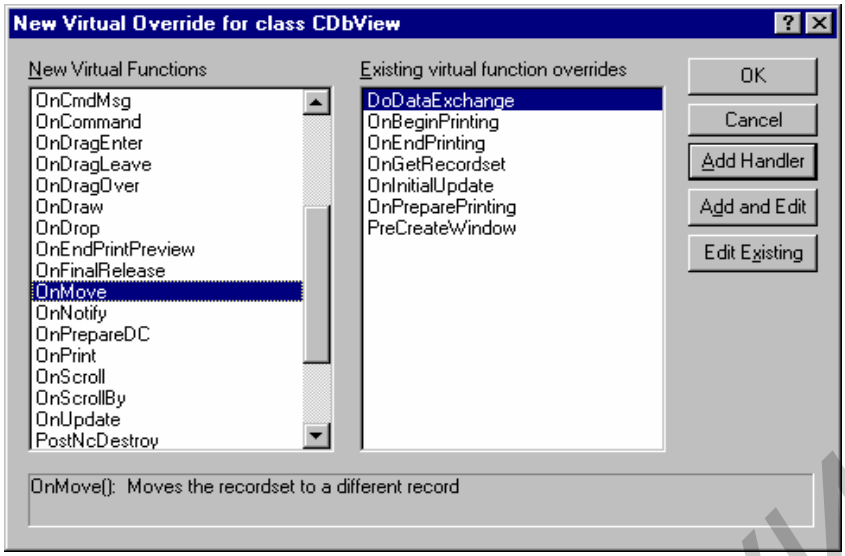

Рис. 4.7. Переопределение функции OnMove

Листинг функции CDBView: :OnMove():

```
BOOL CDBView::OnMove(UNIT nIDMoveCommand)
{
    if (m_bAdding) {
        m_bAdding = FALSE;
        UpdateData(TRUE);
        if (m_pSet->CanUpdate())
            m_pSet->Update();
        m_pSet->Requery();
        UpdateData(FALSE);
          CEdit* pCtrl = (CEdit*)GetDlgltem(IDC_USERID);
        pCtrI->SetReadOnIy(TRUE);
        return TRUE;
    } else
         return CRecordView::OnMove(nIDMoveCommand);
}
```

10. Сделайте двойной щелчок на функции OnRecordDelete() и отредактируйте ее так, чтобы ее код соответствовал приведенному ниже листингу. Пояснения к этому тексту будут даны в следующем разделе.

Листинг функции CDBView::OnRecordDelete():

```
void CDBView::OnRecordDelete()
{
   m_pSet->Delete();
    m_pSet->MoveNext();
   if (m_pSet->lsEOF())
```

```
        m_pSet->MoveLast();
    if (m_pSet->lsBOF())
        m_pSet->SetFieldNull(NULL);
    UpdateData(FALSE);
}
```

Мы модифицировали приложение, и теперь оно способно выполнять добавление и удаление записей, равно как и их обновление. Откомпилируйте приложение и запустите его на выполнение, выбрав в меню Visual Studio команду Builds Execute или нажав клавиши <Ctrl+F5>. Когда приложение начнет работу, на экране раскроется его главное окно, внешний вид которого не претерпел никаких изменений в сравнении с предыдущей версией приложения. Однако теперь у вас появилась возможность добавить в базу данных новую запись, щелкнув на пиктограмме добавления записи на панели инструментов (или выбрав команду Record\Add Record), либо удалить текущую запись из базы, щелкнув на пиктограмме удаления записи на панели инструментов (или выбрав команду Record\Delete Record).

После щелчка на пиктограмме добавления записи приложение отображает в экранной форме пустую запись. Заполните поля новой записи. При переходе к другой записи приложение автоматически внесет новую запись в базу данных. Для того чтобы запись удалить, просто щелкните на пиктограмме удаления. Текущая запись (та, которая отображена на экране) исчезнет, и на экран будет выведена следующая запись базы данных.

Анализ функции OnRecordAdd(). Вероятно, вам будет интересно узнать, как работают подпрограммы на C++, добавленные в приложение. Функция OnRecordAdd() начинает свою работу с вызова метода AddNew() класса CDBSet, производного от класса CRecordset. Вызванная функция формирует пустую запись, предназначенную для заполнения пользователем. Однако эта запись не появится на экране до тех пор, пока не будет вызван метод UpdateData() класса представления. Но прежде чем осуществить вызов этого метода, необходимо проделать подготовительные действия.

После того, как пользователь создаст новую запись, необходимо будет обновить базу данных. Установка в данной подпрограмме определенного флажка позволит подпрограмме пересылки определить, какое именно действие пользователя имеет место: перемещение к следующей записи базы данных от существовавшей ранее записи базы или же от вновь добавленной. Именно с этой целью переменной m_bAdding присваивается значение TRUE.

В данный момент, когда пользователю предоставляется возможность ввести новую запись, необходимо изменить статус поля кода служащего UserID, обычно имеющего атрибут “только чтение". Для снятия этого атрибута программе, прежде всего, необходимо с помощью функции GetDIgItem() получить указатель на соответствующий элемент управления, а затем вызвать метод SetReadOnly() для присвоения значения FALSE атрибуту “только чтение” этого элемента управления.

Теперь все готово к вызову функции UpdateData() для отображения на экране новой пустой записи.

Анализ функции OnMove(). Теперь, когда пустая запись выведена на экран, пользователю не составит большого труда заполнить поля ввода необходимыми данными. Для того чтобы новая запись действительно была помещена в базу данных, пользователю необходимо выполнить переход к другой записи базы. При этом будет вызван метод OnMove() класса представления. Обычно функция OnMove() не выполняет ничего, кроме отображения следующей записи базы данных. Сделанное нами переопределение этой функции дополнительно обеспечит и сохранение новой записи.

При вызове функция OnMove() прежде всего проверяет значение логической переменной m_bAdding и таким образом выясняет, от какой записи происходит переход: от существовавшей или от вновь добавленной. Если значение m_bAdding равно FALSE, то основное тело оператора IF пропускается и выполняется фрагмент программы, следующий за ELSE. При этом программа вызывает метод OnMove базового класса (CRecordView), который выполняет обычный переход на следующую запись.

Если переменная m_bAdding имеет значение TRUE, выполняется основное тело оператора IF. Здесь программа прежде всего сбрасывает флаг m_bAdding, а затем вызывает функцию UpdateData() для передачи данных из полей окна представления в буфер выбранных записей. Вызов функции CanUpdate() класса выборки данных определяет, можно ли обновлять источник данных, и, если можно, вызов функции Update(), являющейся членом этого же класса, добавляет новую запись к источнику данных.

Для формирования новой выборки данных программа должна вызвать функцию Requery(), являющуюся членом класса CRecordset, а затем вызовом метода класса окна представления UpdateData() поместить новые данные в элементы управления этого окна. И наконец, программа восстанавливает для поля кода служащего UserID атрибут "только чтение", еще раз вызвав функции GetDIgItem() и SetReadOnIy().

Анализ функции OnRecordDeIete(). Удаление записи выполняется достаточно просто. Функция OnRecordDe lete() вызывает функцию Delete(), являющуюся членом класса выборки данных. После выполнения удаления вызов метода MoveNext() класса выборки данных позволяет организовать переход к отображению следующей записи таблицы.

Однако здесь может возникнуть проблема, если удаляемая запись была в таблице последней или же единственной. Вызов метода IsEOF() класса CRecordset позволяет выяснить, достигнут ли конец последовательности записей. Если эта функция возвращает TRUE, то указатель записи нужно поместить на последнюю запись в текущей выборке. Для этого используется метод класса выборки данных MoveLast.

Когда все записи из текущей выборки данных будут удалены, указатель текущей записи будет находиться в начале выборки. Программа должна проверить наличие такой ситуации посредством вызова метода IsBOF() класса CRecordset. Если эта функция возвращает значение TRUE, то программа устанавливает значения полей текущей записи равными NULL.

Для завершения работы подпрограммы необходимо обновить содержимое окна представления, что осуществляется вызовом функции UpdateData().

Сортировка и фильтрация записей. Часто при работе с базой данных требуется изменить порядок, в котором записи отображаются на экране, или же осуществить поиск записей, удовлетворяющих определенному критерию. Существующие в MFC классы работы с базами данных ODBC располагают методами, позволяющими сортировать выбранные записи по любому из их полей. Кроме того, вызов определенных методов этих классов предоставит возможность ограничить набор отображаемых записей только такими, поля которых содержат указанную информацию, например конкретное имя или идентификатор. Данная операция называется *фильтрацией.* В этом разделе мы добавим функции сортировки и фильтрации в наше приложение. Выполните следующие действия:

1. Добавьте меню Sort (Сортировка) в основное меню приложения, как показано на рис.4.8. Предоставьте Visual Studio автоматически определить идентификаторы команд.

2. С помощью мастера ClassWizard организуйте в классе CDBView перехват четырех новых команд сортировки, используя имена функций, предложенные этим мастером. Окончательный вид окна ClassWizard показан на рис.4.9.

3. Добавьте меню Filter (Фильтрация) в строку меню приложения. Предоставьте Visual Studio установить идентификаторы команд.

4. С помощью мастера ClassWizard организуйте в классе CDBView перехват четырех новых команд фильтрации, используя имена функций, предложенные этим мастером.

5. Выберите команду Insert\Resource и создайте новое диалоговое окно, сделав двойной щелчок на элементе Dialog, а затем отредактируйте диалоговое окно так, чтобы оно выглядело, как показано на рис.4.10. Присвойте элементу управления – текстовому полю – идентификатор ID_FILTERVALUE.

Оставив новое диалоговое окно раскрытым на экране, запустите мастер ClassWizard. Раскроется диалоговое окно Adding a Class. Установите опцию Create a new class и щелкните на кнопке OK. Раскроется диалоговое окно New Class. В поле Name введите значение CFiIterDlg, как показано на рис. 4.11.

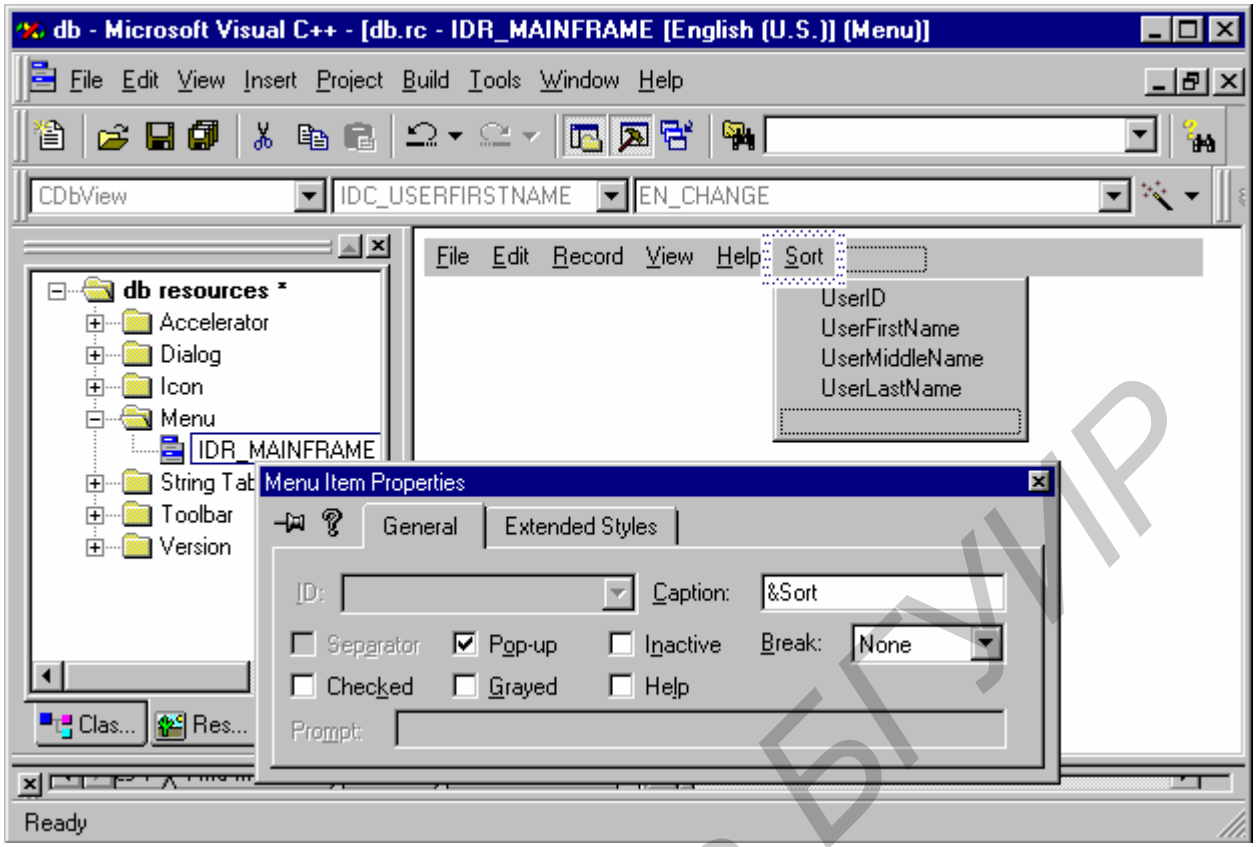

Рис.4.8.Создание меню Sort

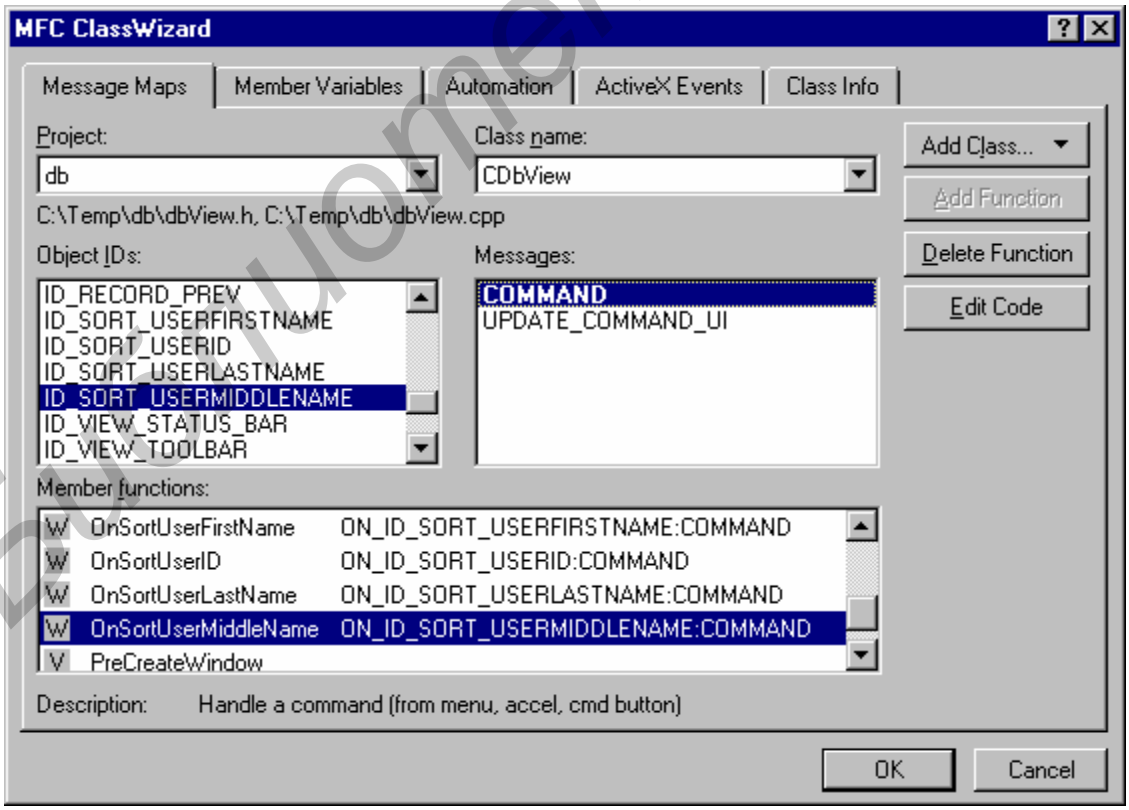

Рис. 4.9. Вид окна мастера ClassWizard после добавления четырех новых функций сортировки

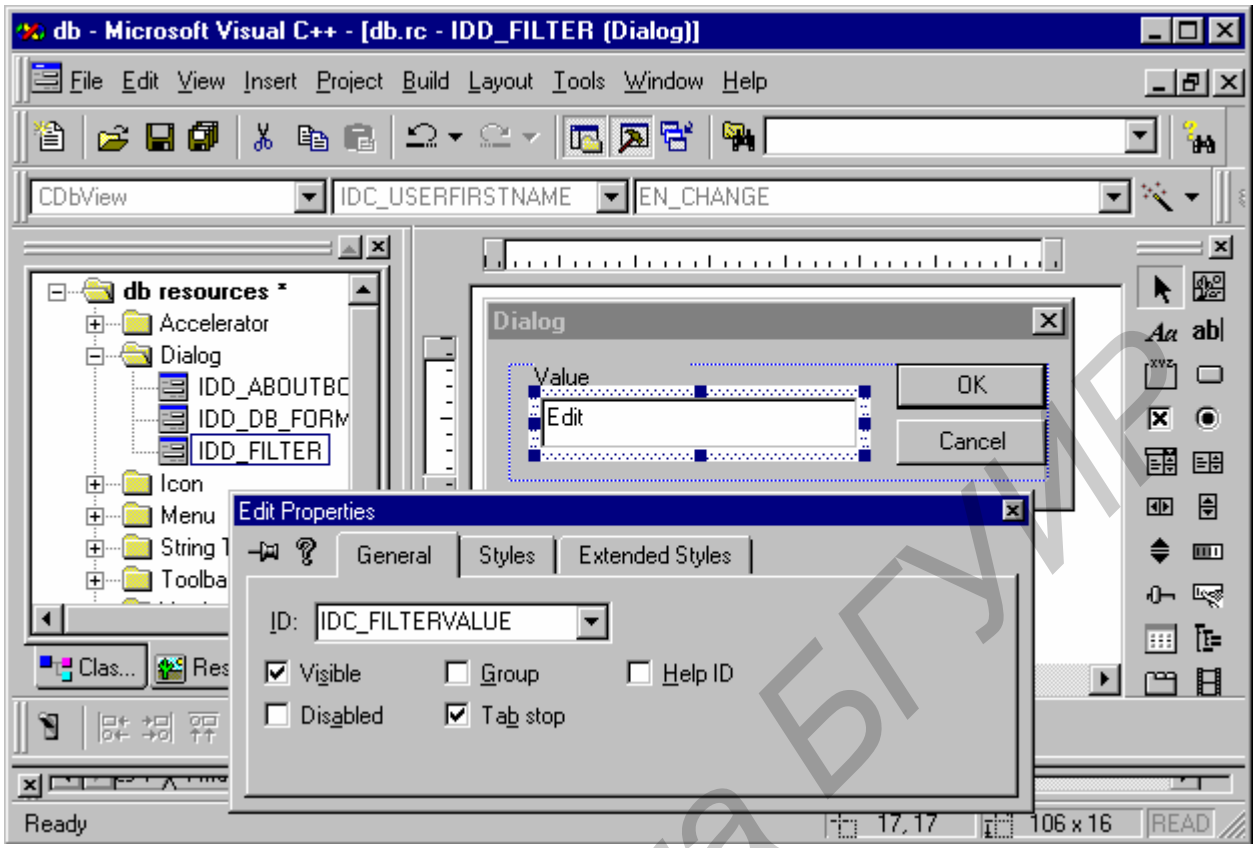

Рис. 4.10. Создание диалогового окна установки параметров фильтрации

6. В окне мастера ClassWizard щелкните на корешке вкладки Member Variables, свяжите элемент управления IDC_FILTERVALUE с переменной-членом m_fiIterValue. Завершите работу с мастером ClassWizard, щелкнув на кнопке OK.

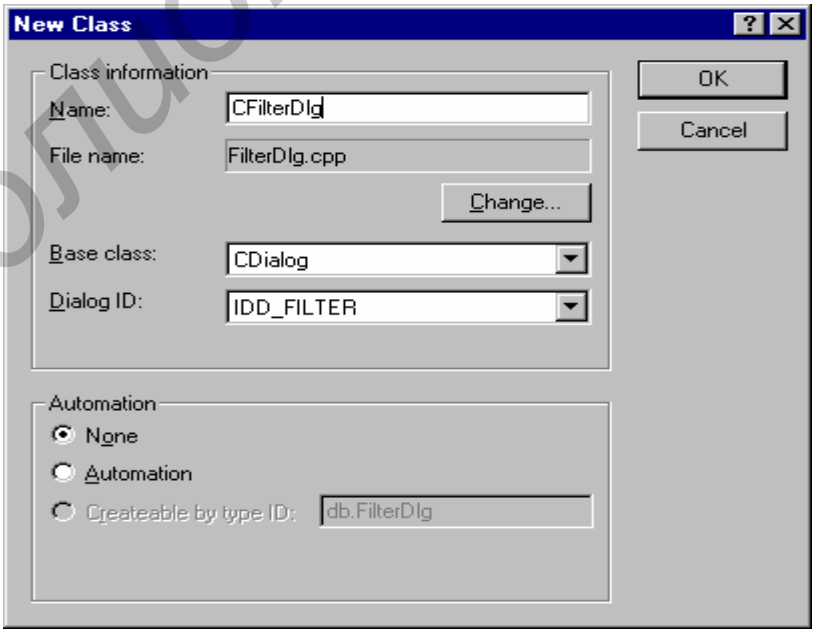

Рис. 4.11. Создание класса диалога для окна Filter

Теперь, когда меню и диалоговые окна уже созданы и связаны с заготовками функций, необходимо добавить в эти заготовки определенный программный код. На панели ClassView сделайте двойные щелчки на функциях, связанных с командами сортировки, отредактируйте их текст в соответствии с листингом:

```
void CDBView:: OnSortUserID ()
{
    m_pSet->Close();
    m_pSet->m_strSort = "UserID";
    m_pSet->Open():
    UpdateData(FALSE);
}

void CDBView: :OnSortFirstName ()

{
        m_pSet->Close();
        m_pSet->m_strSort = "UserFirstName";
        m_pSet->Open();
        UpdateData(FALSE);
}


void CDBView: : OnSortMiddleName()
 {
        m_pSet->Close();
        m_pSet->m_strSort = “UserMiddleName";
        m_pSet->Open();
        UpdateData(FALSE);
 }

void CDBView: : OnSortLastName()
 {
        m_pSet->Close();
        m_pSet->m_strSort = “UserLastName";
        m_pSet->Open();
        UpdateData(FALSE);
 }

void CDBView::OnSortEMail ()
{
        m_pSet->Close():
        m_pSet->m_strSort = "UserEMail";
```

```
        m_pSet->Open();
        UpdateData(FALSE);
}
```

Введите в начало файла DBView.cpp после уже имеющихся директив #include строку **#include "FilterDlg.h"**. Добавьте функции-обработчики:

```
void CDBView:: OnFilterUserID ()
{
    DoFilter("UserID");
}

    void CDBView: :OnFilterFirstName ()
{
    DoFilter("FirstName");
}

    void CDBView: : OnFilterMiddleName()
{
    DoFilter("MiddleName");
}

    void CDBView: : OnFilterLastName()
{
    DoFilter("LastName");
}

    void CDBView::OnFilterEMail ()
{
        DoFilter("EMail");
}
```

Все эти четыре функции вызывают функцию DoFilter(). Далее необходимо будет написать функцию, выполняющую фильтрацию записей базы данных, представленных в классе выборки данных. На панели ClassView щелкните правой кнопкой мыши на классе CDBView и выберите в раскрывшемся контекстном меню команду Add Member Function. Укажите в раскрывшемся диалоговом окне тип функции void и введите ее объявление как DoFilter(CString col). Данный метод должен быть защищенным, так как он вызывается только другими методами этого же класса CDBView. На панели ClassView сделайте двойной щелчок на функции DoFilter() и поместите в нее текст программы, показанный в листинге:

```
        void CDBView::DoFilter(CString col)
{
      CFiIterDlg dig;
       int result = dlg.DoModal():
            if (result == IDOK)
            CString str = col + " = " + dlg.m_fiIterValue;
            m_pSet->Close();
            m_pSet->m_strFiIter = str;
             m_pSet->Open();
             int recCount = m_pSet->GetRecordCount();
             if (recCount == 0) {
                    MessageBox("No matching records.");
                    m_pSet->Close();
                    m_pSet->m_strFiIter = "";
                    m_pSet->Open();
                       UpdateData(FALSE);
             }
}
```

Мы добавили к создаваемому приложению функции сортировки и фильтрации записей базы данных. Оттранслируйте приложение и запустите его на выполнение. На экране появится главное окно приложения, которое выглядит так же, как и раньше. Однако теперь можно сортировать записи по любому полю, для чего достаточно просто выбрать имя поля в меню Sort. Кроме того, появилась возможность задать фильтрацию отображаемых записей, выбрав имя требуемого поля в меню Filter, а затем введя значение фильтра в раскрывшемся диалоговом окне Filter. Определить, как записи отсортированы или какой для них задан фильтр, вы сможете, перемещаясь от одной записи к другой. Попробуйте, например, отсортировать записи по отделам или по зарплате, а затем установите фильтрацию по любому из отделов, который вы видели, просматривая базу.

Анализ функции OnSortFirstName(). Все функции сортировки имеют одинаковую структуру. Они закрывают выборку данных, устанавливают свои переменные-члены m_strSort в выборке и снова открывают выборку данных, а затем вызывают функцию UpdateData() для обновления окна представления данными из вновь полученной отсортированной выборки данных. Однако в тексте функций сортировки вы не найдете ни одного вызова функции, в названии которой было бы слово Sort. Когда же в таком случае выполняется сортировка? Она выполняется, когда выборка данных открывается заново.

Объект класса CRecordSet (как и объект любого другого класса, производного от CRecordSet, например объект класса CDBSet в этой программе) использует специальную строковую переменную m_strSort для определения способа упорядочения записей. Объект анализирует эту строковую переменную при

формировании выборки данных и соответственно упорядочивает выбранные из базы записи.

Анализ функции DoFilter(). Всякий раз, когда пользователь выбирает команду из меню Filter, управляющая программа вызывает соответствующий этой команде метод: OnFi lterDept(), OnFilterUserlD(), OnFiIterMiddleName() или OnFilterEMail(). Каждая из этих функций ничего не делает, кроме вызова метода DoFilter(), передавая ему в качестве параметра строковую переменную, определяющую поле, по которому требуется выполнить фильтрацию.

Функция DoFilter() независимо от того, какая именно команда была выбрана в меню, всегда отображает одно и то же диалоговое окно. Если значение *result* не равно IDOK, значит, пользователь выполнил щелчок на кнопке Cancel и весь оператор IF пропускается, а функции DoFilter() остается только закончить свою работу.

Внутри конструкции IF прежде всего создается строковая переменная, которая будет использоваться для фильтрации записей базы данных. Строковая переменная применяется для выполнения фильтрации записей так же, как это происходит при сортировке. В данном случае строковая переменная называется m_strFiIter. Строка, которая используется для фильтрации записей базы данных, должна иметь следующий формат:

ИдентификаторПоля = Значение

Здесь *ИдентификаторПоля* является аргументом типа CString функции DoFiIter(), а *Значение* вводится пользователем в диалоговом окне. Например, если пользователь выберет команду фильтрации по полю отдела и введет в диалоговом окне значение фильтра hardware, функция DoFiIter() должна будет создать строку.

Сформировав указанную строку, программа будет готова к выполнению фильтрации записей. Для этого, как и в случае сортировки, выборка данных должна быть закрыта, а затем, при ее повторном открытии, функция DoFilter() выполнит формирование выборки данных с учетом требуемой фильтрации.

Что произойдет, если в результате работы установленного фильтра не будет выбрано ни одной записи? Хороший вопрос. Функция DoFilter() обнаруживает подобную ситуацию, подсчитывая количество записей в создаваемой выборке и сравнивая затем это число с нулем. Если набор записей пуст, программа выводит окно сообщения, информирующее пользователя о сложившейся ситуации. Затем программа закрывает выборку, присваивает строковой переменной фильтра пустое значение и снова открывает выборку записей. Таким образом восстанавливается выборка, включающая все записи таблицы User.

И, наконец, независимо от того, удалось ли обнаружить записи, отвечающие заданному фильтру, или же выборка данных включает всю базу данных, программа должна заново отобразить данные на экране. Для этого вызывается функция UpdateData().

Выбор между классами ODBC и DAO. MSVC++ предоставляет набор классов для доступа к БД через ODBC и DAO. Во многих отношениях DAO является для классов ODBC суперклассом, включая большинство функциональных возможностей ODBC и добавляя при этом множество своих собственных. К сожалению, хотя классы DAO и могут работать с источниками данных ODBC,

для которых существуют ODBC-драйверы, такое их применение не особенно эффективно. По этой причине DAO-классы больше подходят для создания программных приложений, оперирующих файлами баз данных формата .mdb фирмы Microsoft, создаваемых приложением Microsoft Access. Файлы других форматов, с которыми можно работать напрямую, используя классы DAO, создаются приложениями FoxPro и Excel.

Классы интерфейса DAO, которые используют приложение Microsoft Jet Database Engine, настолько похожи на классы интерфейса ODBC, что в некоторых случаях можно путем простого изменения названия класса в тексте программы заменить интерфейс доступа с ODBC на DAO: CDatabase изменяется на CDaoDatabase, CRecordset – на CDaoRecordset, а CRecordView – на CDaoRecordView. Однако между классами ODBC и DAO имеется существенное различие в том, как реализуются системные библиотеки. ODBC-классы реализованы как набор модулей DLL, в то время как классы DAO реализованы в виде объектов OLE. Использование объектов OLE делает систему DAO несколько более современной в сравнении с ODBC, по крайней мере, в отношении архитектуры.

Хотя система DAO реализована в виде объектов OLE, вам не придется беспокоиться о работе с подобными объектами напрямую. Входящие в MFC классы DAO берут обработку всех деталей управления на себя, предоставляя данные и методы, обеспечивающие взаимодействие с объектами OLE. Класс CDaoWorkspace обеспечивает с помощью статических методов прямой доступ к объектам ядра базы данных DAO. Хотя MFC берет управление рабочей областью на себя, можно использовать ее данные и методы для непосредственной инициализации связи с базой данных.

Еще одно отличие состоит в том, что классы DAO предоставляют более мощный набор функций, которые можно использовать для манипулирования базой данных. Эти более мощные методы позволяют выполнять с базами данных сложные операции, используя небольшой объем исходного текста на C++ или SQL-выражения, написанные непосредственно разработчиком:

- обе системы (ODBC и DAO) могут работать с ODBC-источниками данных, но в этом случае DAO менее эффективна, так как ориентирована для работы с базами данных формата . mdb;
- мастер AppWizard может создать заготовку БД-приложения, используя либо классы ODBC, либо классы DAO. Выбор типа создаваемого приложения зависит частично от баз данных, с которыми вы будете работать;
- обе системы – и ODBC и DAO – используют для соединения с базой данных, к которой осуществляется доступ, объекты классов баз данных MFC. В ODBC такой класс базы данных называется CDatabase, а в системе DAO – CDaoDatabase. Хотя эти классы и имеют разные названия, они содержат множество похожих членов;
- обе системы, ODBC и DAO, используют объекты класса выборки данных для хранения записей, выбранных на текущий момент. В ODBC такой класс выборки данных называется CRecordset, а в системе DAO – CDaoRecordset. DAO-класс выборки данных содержит практически все члены класса ODBC. Кроме того, DAO-класс имеет большой набор дополнительных методов;

- системы ODBC и DAO используют схожие методики просмотра содержимого источника данных, а именно: в обеих системах приложение должно создать объект базы данных, объект выборки данных, а затем вызвать методы соответствующего класса для манипулирования базой данных.

Различия между системами ODBC и DAO состоят в следующем:

- хотя входящие в MFC классы ODBC и DAO похожи (иногда даже очень), некоторые аналогичные методы имеют разные имена. Кроме того, классы DAO включают много методов, которым нет аналогов в классах ODBC;
- в системе ODBC для определения опций, которые могут использоваться при открытии выборки данных, используются макросы и перечисления, тогда как в DAO для этих целей определены константы;
- большое количество существующих ODBC-драйверов делает систему ODBC пригодной для работы с множеством файлов баз данных различных форматов, в то время как система DAO больше подходит для приложений, работающих только с файлами формата . mdb;
- система ODBC реализована в виде набора DLL-модулей, а DAO реализована как набор объектов OLE;
- в ODBC объект класса CDatabase напрямую взаимодействует с источником данных. DAO-объект класса CDaoWorkspace занимает промежуточное положение между объектами классов CDaoRecordset и CDaoDatabase, что дает возможность рабочей среде взаимодействовать со многими объектами класса баз данных.

OLE DB – это совокупность интерфейсов OLE, которые упрощают доступ к данным, сохраненным приложениями, не являющимися СУБД, например, хранящимся в почтовых ящиках электронной почты или линейных файлах. Приложение, использующее OLE DB, может интегрировать информацию из таких СУБД, как Oracle, SQL Server и Access, с информацией из систем, не являющихся СУБД, но использующих возможности OLE.

Для использования этого мощного инструмента вы должны уверенно чувствовать себя при работе с интерфейсами OLE. Если вам раньше приходилось создавать приложения OLE (ActiveX) только с помощью MFC и мастера App-Wizard, вас может шокировать знакомство с тем, что Microsoft полагает "упрощенным". Вы встретитесь с множеством вызовов функции Querylnterface() и множеством переменных с именами наподобие pIColslnfo и rgCol Info. Все же, разобравшись во всех интерфейсах и всех установках, вы сможете вызвать такую функцию, как GetData(), и получить информацию из приложения, не являющегося СУБД, как будто это база данных. В результате получается существенный выигрыш во времени.

Варианты задания:

1. Разработать приложение управления базой данных сдачи студентами экзаменационной сессии.
2. Разработать приложение управления базой данных студенческого общежития.
3. Разработать приложение управления базой данных выданных студентам курсовых.
4. Разработать приложение управления базой данных посещения студентами занятий.
5. Разработать приложение управления базой данных студенческой библиотеки.
6. Разработать приложение управления базой данных студенческой бухгалтерии для выдачи стипендий.
7. Разработать приложение управления базой данных распределения путевок студенческого профкома.
8. Разработать приложение управления базой данных, содержащей расписание студенческих занятий.
9. Разработать приложение управления базой данных коммунальных платежей за квартиру.
10. Разработать приложение управления базой данных учета времени доступа к Интернету.
11. Разработать приложение управления базой данных учета времени переговоров по телефону.
12. Разработать приложение управления базой данных учета страховых полисов.

В разрабатываемом приложении обеспечить добавление, редактирование и удаление записей из базы данных, сохранение результатов в файле (создание текстового отчета).

Литература

Грекори К. Использование Visual C++ 6. – М.: Вильямс, 1999.

Круглински Д. Основы Visual C++. Версия 4. – М.: Русская редакция, 1997.

Приложение 1

Мастер создания приложений AppWizard

Мастер создания приложений AppWizard позволяет создавать различные типы приложений, но обычно большинством программистов используются исполняемые программы (файл приложения с расширением .exe). Кроме того, желательно получить от AppWizard готовые фрагменты программного кода – классы, объекты, функции, которые присутствуют едва ли не в каждой стандартной программе. Для создания программы подобного типа, необходимо выбрать File\New, а затем – вкладку Projects в окне New, как это показано на рис.1.1.

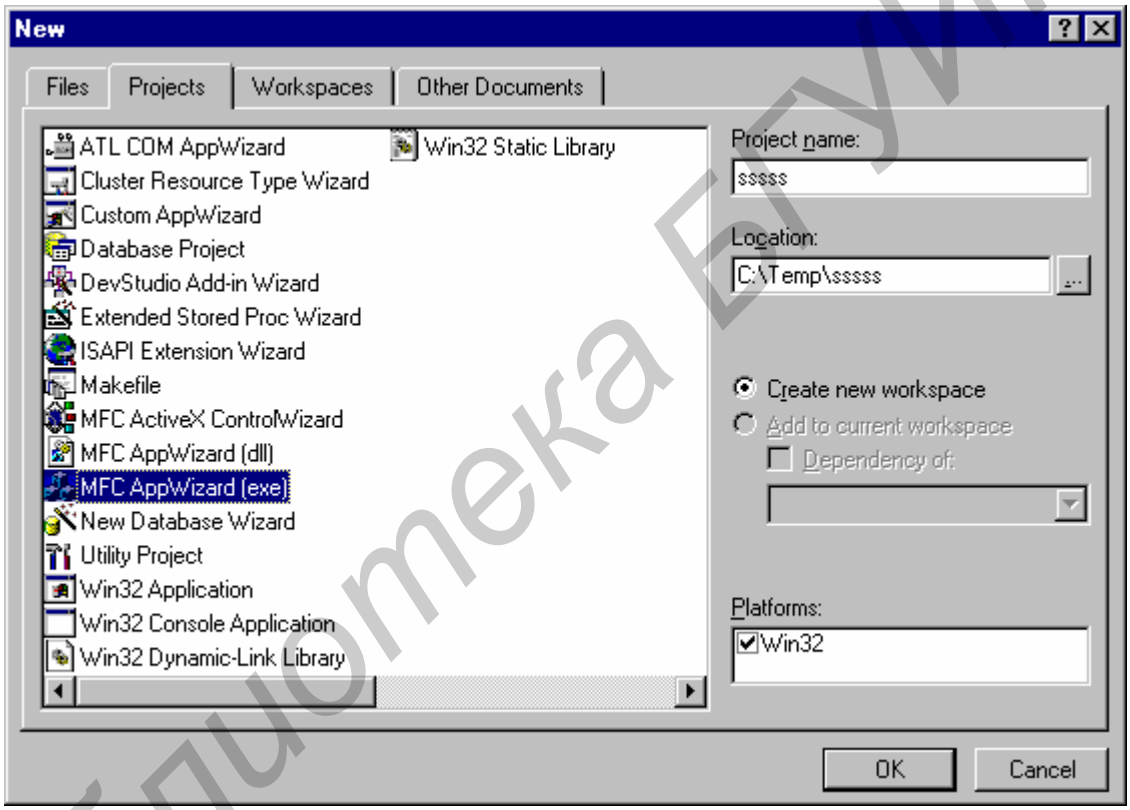

Рис. 1.1. Выбор типа приложения, которое вы хотите создать

В левой части окна находится список возможных типов проектов. Для создания типового приложения необходимо выбрать MFC AppWizard (.exe). Также необходимо указать имя проекта в поле Project name, а в поле Location – каталог, в котором будет находиться проект, (например, мы создали приложение с именем sssss в каталоге c:\temp). Дальнейшие действия AppWizard пронумерованы как отдельные этапы (step), причем номер текущего этапа – всегда в строке заголовка окна MFC AppWizard. Данный тип проекта использует библиотеку классов Microsoft Foundation Classes (MFC). На каждом этапе программист может изменить некоторые параметры создаваемого приложения. Для перехода на следующий этап необходимо щелкнуть на кнопке Next, для перехода к предыдущему этапу – щелкнуть на кнопке Back. При нажатии на кнопке Cancel про-

цесс создания приложения вообще будет прерван. Оперативная справка по текущему этапу вызывается на экран с помощью кнопки Help. Кнопка Finish позволяет завершить сеанс настройки, пропустив последующие этапы и настроить все оставшиеся параметры в состоянии по умолчанию. Рассмотрим более подробно этапы создания приложения AppWizard:

Шаг 1. Выбор типа приложения. Первое, что должен определить программист, приступая к работе в AppWizard, – сколько документов будет поддерживать будущее приложение, т.е. будет ли оно MDI-приложением, SDI-приложением или простым диалоговым приложением. Для каждого из этих типов приложений AppWizard создает различные классы. Окно MFC AppWizard при этом будет выглядеть так, как показано на рис.1.2. Подробности о каждом из этих типов приложений приведены ниже:

- SDI-приложение *(SD I– Single Document Interface,* интерфейс с единственным документом) позволяет в каждый момент времени иметь открытым только один документ, однако количество открытых окон не ограничено. Примером может служить известный каждому редактор Notepad. Если вы выберете в таком приложении File\0pen, то открытый в текущий момент файл будет закрыт прежде, чем откроется новый. Создание SDI-приложения настраивается в окне MFC AppWizard переключателем Single document.

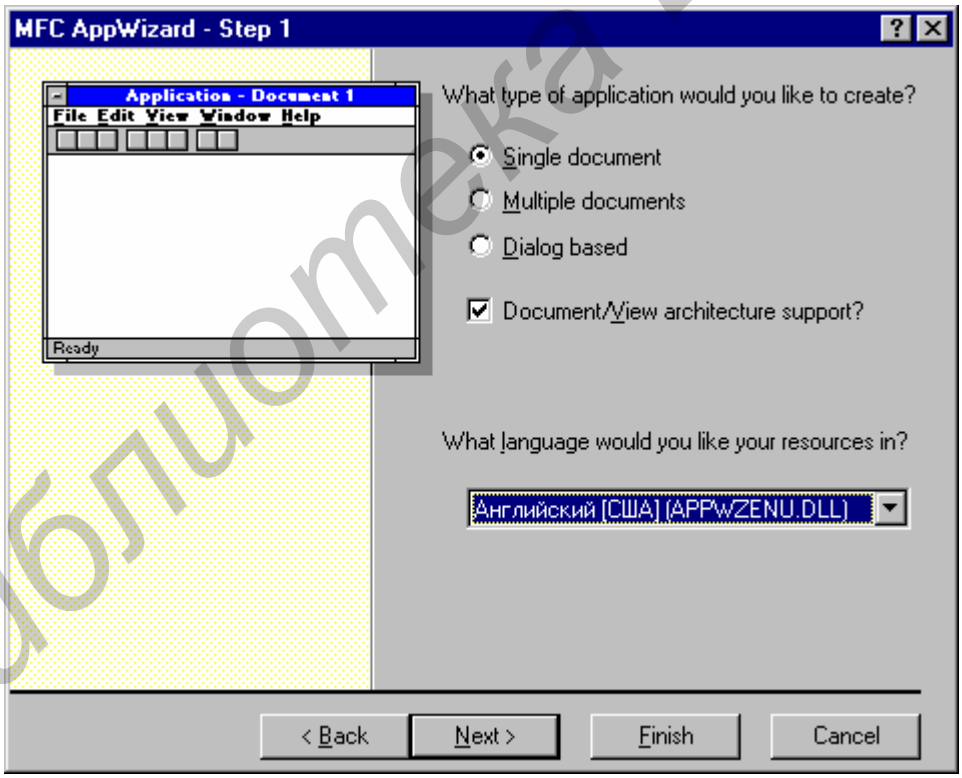

Рис. 1.2. Первый этап создания типового приложения с помощью AppWizard – выбор варианта интерфейса пользователя

- MDl-приложение (*MDI – Multiple Document Interface,* дословно – ”многодокументный интерфейс”) может одновременно держать открытыми не-

сколько документов, каждый из которых представлен отдельным файлом, примеры – Excel, Word и другие хорошо знакомые многим аналогичные приложения. Такие приложения обязательно имеют в главном меню пункт Window, а в меню File – пункт Close. Создание MDI-приложения настраивается в окне MFC AppWizard переключателем Multiple documents.

- Простое диалоговое приложение, как правило, вообще не открывает документов. Примером могут служить приложение Character Map (Таблица символов) и множество других простых приложений, которые входят в базовый комплект Windows. Такие приложения не имеют меню. Приложение Character Map, скорее всего, находится в папке Accessories (Стандартные), которую можно запустить, щелкнув на кнопке Start (Пуск). Возможно, вам понадобится установить его на свой компьютер, тогда воспользуйтесь функцией Add/Remove programs программы Control Panel.

Создание приложения этого типа настраивается в окне MFC AppWizard переключателем Dialog based. В левой части диалогового окна после выбора переключателя типа приложения появится соответствующий образец вашего будущего приложения. Ниже этой группы переключателей в диалоговом окне находится флажок Document/View architecture support (Поддержка архитектуры документ/представление). Пользователи, которые имеют большой опыт разработки приложений в среде Visual C++, могут отключить поддержку этой архитектуры мастером, но для большинства разработчиков она будет отнюдь не лишней. Поэтому в дальнейшем, если не будет оговорено особо, будем считать, что флажок Document/View architecture support установлен.

Еще ниже в диалоговом окне находится раскрывающийся список для выбора языка, который вы собираетесь использовать при написании текста программы. Если системный язык операционной среды, не заданный по умолчанию English (United States) – американский вариант английского, не забудьте сделать такой же выбор и в списке. Иначе можете в дальнейшем столкнуться с совершенно неожиданными эффектами в работе с ClassWizard. (Конечно, если вы создаете приложения для заказчика, который пользуется американским английским, у вас не остается иного выбора, как изменить системный язык с помощью программы Control panel.)

Шаг 2. Базы данных. Второй этап создания приложения Windows с помощью AppWizard – выбор уровня поддержки операций с базами данных. Окно MFC AppWizard при этом будет выглядеть так, как показано на рис.1.3.

Здесь вам на выбор предлагаются четыре варианта уровня поддержки:

- если работа с базами данных в приложении не планируется, выберите переключатель None (Никакой);
- если предполагается иметь доступ к базам данных, но для этого не будут использованы классы просмотра, производные от CFormView, или нет необходимости в меню Record (Запись), выбирайте переключатель Header files only (Только файлы заголовков);
- если вы планируете разрабатывать классы просмотра базы данных в приложении как производные от CFormView и иметь меню Record, но не нуждаетесь в средствах сохранения-восстановления (сериализации) документов, выбирайте переключатель Database view without file support (Просмотр базы данных без

поддержки операций с файлами). Записи в базе данных можно будет обновлять с помощью CRecordset – класса MFC доступа к базам данных;

- если помимо всего что задано в предыдущем случае, вы планируете и сохранение-восстановление документов на диске (возможно, это будет одна из опций пользователя), выберите Database view with file support (Просмотр базы данных и поддержка операций с файлами).

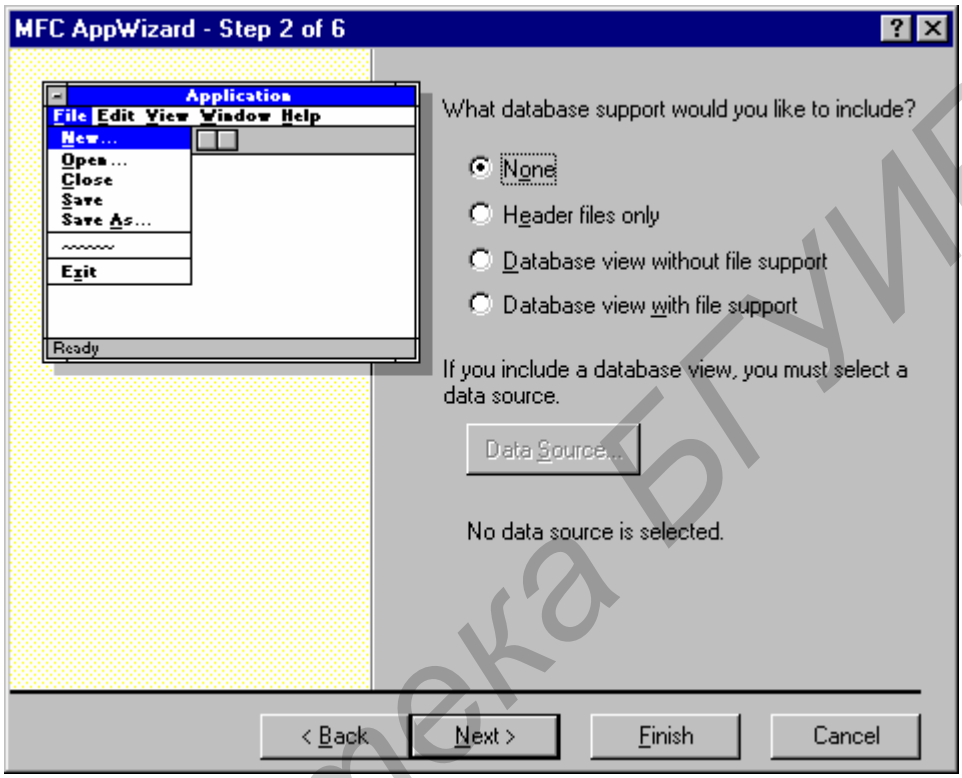

Рис. 1.3. Второй этап создания типового приложения с помощью AppWizard – выбор варианта поддержки операции с базами данных

Если вы выбрали один из вариантов с использованием базы данных, в этом же окне задайте базу, которая будет источником данных. Для этого нужно щелкнуть на кнопке Data Source (Источник данных).

Картинка в левой части окна MFC AppWizard меняется после задания любого из предложенных вариантов обращения к базе данных, демонстрируя последствия сделанного выбора.

Шаг 3. Поддержка составных документов. Третий этап создания выполняемого приложения Windows с помощью AppWizard – выбор уровня поддержки операции с составными документами. Окно MFC AppWizard при этом будет выглядеть так, как показано на рис.1.4.

На выбор предлагается пять вариантов поддержки:

- если не планируется создание OLE-приложения, выберите переключатель None (Никакой);
- если вы хотите, чтобы в приложении использовались связанные или внедренные объекты OLE (например, такие, как документы Word или рабочие лис-

ты Excel), выберите переключатель Container (Контейнер);

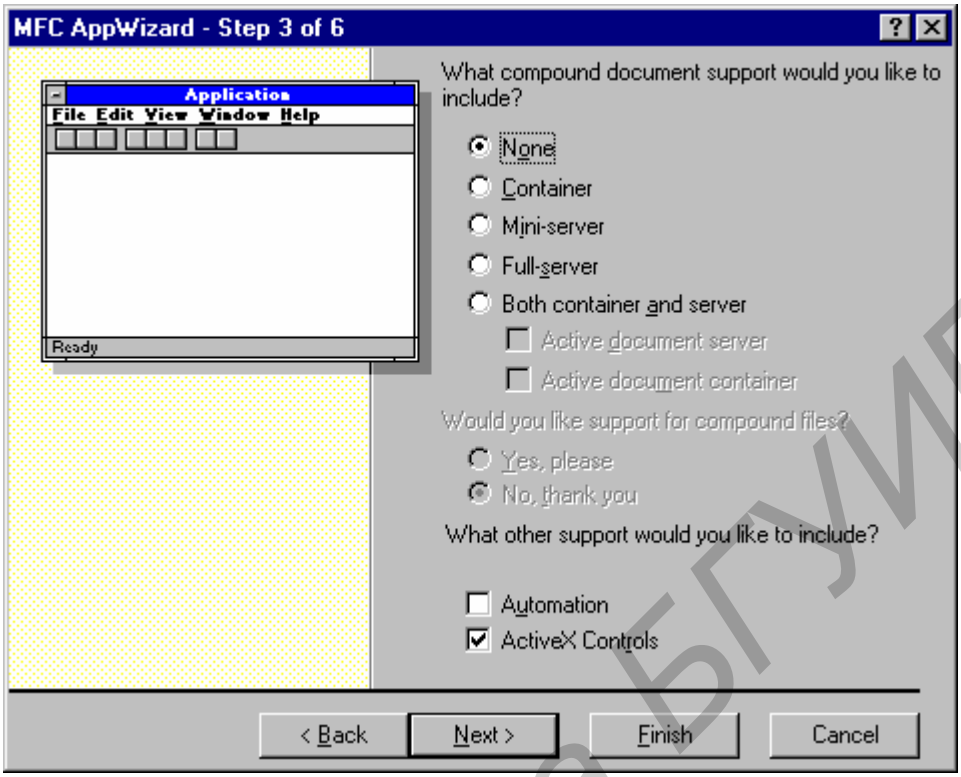

Рис.1.4. Третий этап создания типового приложения с помощью AppWizard – выбор варианта поддержки составных документов

- если планируется создание приложения, документы которого могли бы быть внедрены в другое приложение, но при этом само приложение не будет использоваться автономно, выберите переключатель Mini-server (Мини-сервер);
- если ваше будущее приложение будет не только служить сервером для других приложений, но и сможет работать автономно, выберите переключатель Full-server (Полный сервер);
- если создаваемое приложение должно обладать способностью включать документы других приложений и само обслуживать их своими объектами, выбирайте переключатель Both container and server (и контейнер, и сервер).

Если вы выбрали какой-либо из вариантов поддержки составных документов, то придется поддерживать и *составные файлы (compound files)*. Составные файлы содержат один или более объектов ActiveX и сохраняются на диске в особом формате, так что один из объектов может быть заменен без переписи всего файла. Таким образом удается сберечь довольно много времени. В середине диалогового окна Step 3 имеется группа из двух переключателей – “Would you like to support compound files?” (Необходима ли поддержка составных файлов?)

Если вы хотите, чтобы создаваемое приложение могло передавать управление другому приложению через механизм автоматизации ActiveX, установите флажок Automation (Автоматизация). Если планируется использовать в прило-

жении элементы управления ActiveX, установите флажок ActiveX Controls (Элементы управления ActiveX).

Если хотите, чтобы создаваемое приложение само было элементом управления ActiveX, то все описываемое в этой главе вас не касается, поскольку вам не нужно заказывать типовое приложение (Exe-файл).

Шаг 4. Внешний вид приложения и другие опции. Четвертый этап создания выполняемого приложения Windows с помощью AppWizard – выбор опций, определяющих внешний вид элементов пользовательского интерфейса. MFC AppWizard при этом будет выглядеть так, как показано на рис.1.5.

Диалоговое окно MFC AppWizard – Step 4 Of 6 содержит много переключателей-флажков, соответствующих предлагаемым опциям оформления:

- *Docking toolbar* (Фиксируемая панель инструментов). В приложение будет добавлена панель инструментов, которая может быть пристыкована (зафиксирована) к одной из границ окна (затем можно будет удалить ненужные пиктограммы из панели или добавить новые, связанные с теми пунктами меню, которые вы посчитаете нужными включить в свое приложение);
- *Initial Status* bar (Панель состояния). В приложение будет добавлена панель состояния, в которой можно будет выводить подсказки соответственно выбранным пунктам меню и другие сообщения;

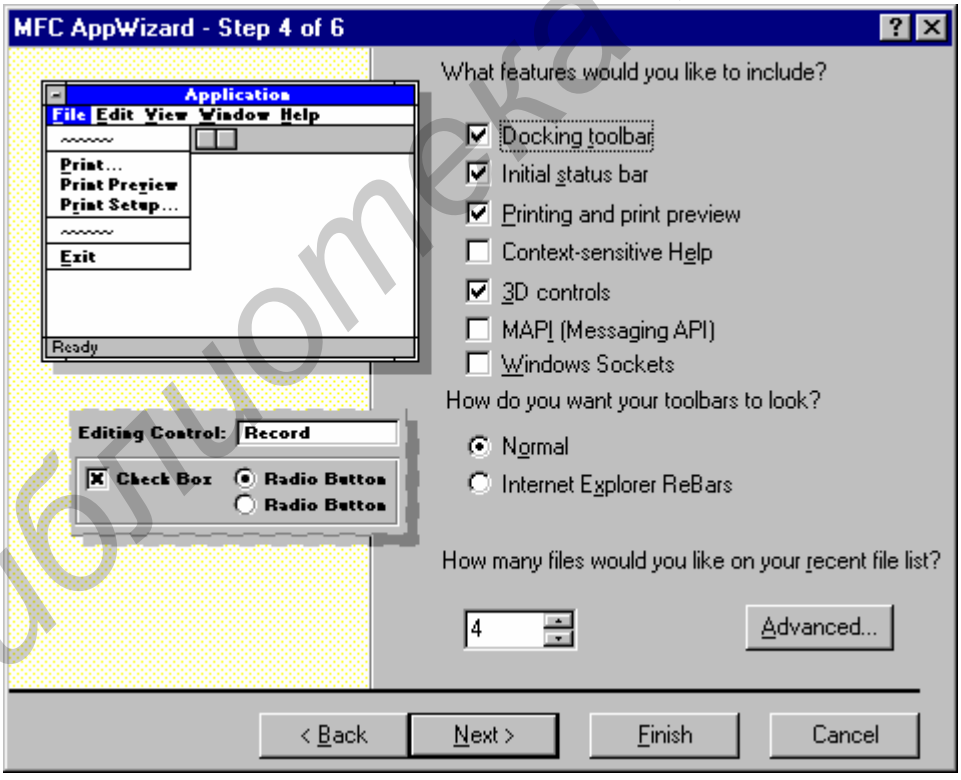

Рис.1.5.Установка некоторых опций пользовательского интерфейса с помощью AppWizard

- *Printing and print preview* (Печать и предварительный просмотр печати). Приложение при выборе этой опции будет иметь пункты Print и Print preview в

меню File, и AppWizard включит в приложение большую часть программного кода, связанного с выполнением этих операций;

- *Context sensitive Help* (Контекстная справка). Меню Help в приложении будет иметь опции Index и Using Help, а значительная часть программного кода, необходимого для организации контекстной справки в приложении, будет включена в него мастером AppWizard;
- *3D controls* (Объемный дизайн элементов управления). При установке этой опции дизайн приложения будет полностью соответствовать стилю, принятому в фирменных приложениях Windows 95. Если вы откажетесь от этой опции, то фон диалоговых окон будет белым, а такие элементы, как текстовые поля, переключатели и вкладки, не будут отбрасывать тени;
- MAPI (Messaging API – почтовый интерфейс). При установке этой опции приложение сможет обмениваться сообщениями по электронной почте;
- *Windows Sockets*. Если эта опция будет установлена, приложение сможет иметь непосредственный доступ к Internet через такие протоколы, как FTP и HTTP (протокол World Wide Web). Можно создать Internet-программу и без поддержки Windows Sockets, если использовать классы Winlnet.

С помощью группы переключателей "*How do you want your toolbars to look?"* мастеру AppWizard можно заказать создание панелей инструментов в традиционном стиле, как в Word или в самом продукте Visual C++ (переключатель Normal), или в новом стиле оформления, принятом в Internet Explorer (переключатель Internet Explorer ReBars). Можно также установить длину списка последних открываемых файлов в поле меню File создаваемого приложения. Для этого служит раскрывающийся список *"How many files would you like on your recent file list?"* По умолчанию этот параметр имеет значение 4 и менять его не рекомендуется без очень весомых причин.

После щелчка на кнопке Advanced (Дополнительно) в нижней части диалогового окна MFC AppWizard Step 4 на экран будет выведено новое диалоговое окно Advanced Options (Дополнительные опции), которое имеет две вкладки. На рис.1.6 показана одна из них – Document Template Strings (Строковые шаблоны

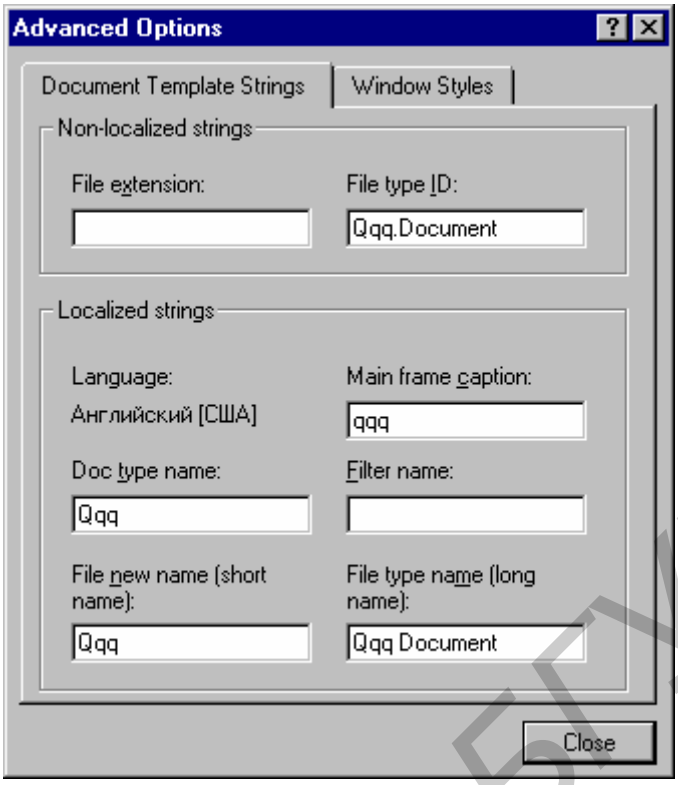

Рис.1.6. Вкладка Document Template Strings диалогового окна Advanced Options

документов). Дело в том, что AppWizard формирует многочисленные запросы и идентификаторы, принимая в качестве главного переменного элемента имя приложения, и иногда ему необходимы аббревиатуры этого имени. Здесь же их можно при желании откорректировать, а также уточнить надпись, которая будет выведена в строке заголовка главного окна создаваемого приложения. Расширения имени файла, если вы установите его в поле File extension, будут автоматически добавляться к именам всех файлов, которые записываются на диск приложением. Аналогично по команде File\Open в соответствующем диалоговом окне будут выведены по умолчанию только файлы с заданным расширением.

На рис.1.7 показана другая вкладка – Window Styles (Стили оформления окон). Это окно позволяет изменить внешний вид окон приложения. Первый флажок – Use Split Window (Использование разделения окна). При его установке в приложение включается весь программный код, необходимый для организации разделения окна приложения таким же образом, как это сделано, например, в редакторе программного кода из комплекта средств Visual Studio. Остальные элементы диалогового окна устанавливают параметры, определяющие оформление фрейма (рамки) главного окна приложения, а для MDI-приложений – фреймов дочерних окон (child frames). Фрейм является весьма важным элементом окна. Системное меню, строка заголовка, кнопки минимизации и максимизации, собственно границы – все это свойства фрейма как объекта. Фрейм главного окна содержит всё SDI-приложение. MDl-приложение имеет несколько дочерних окон (по одному на каждый документ), которые размещаются в пределах главного, родительского, окна.

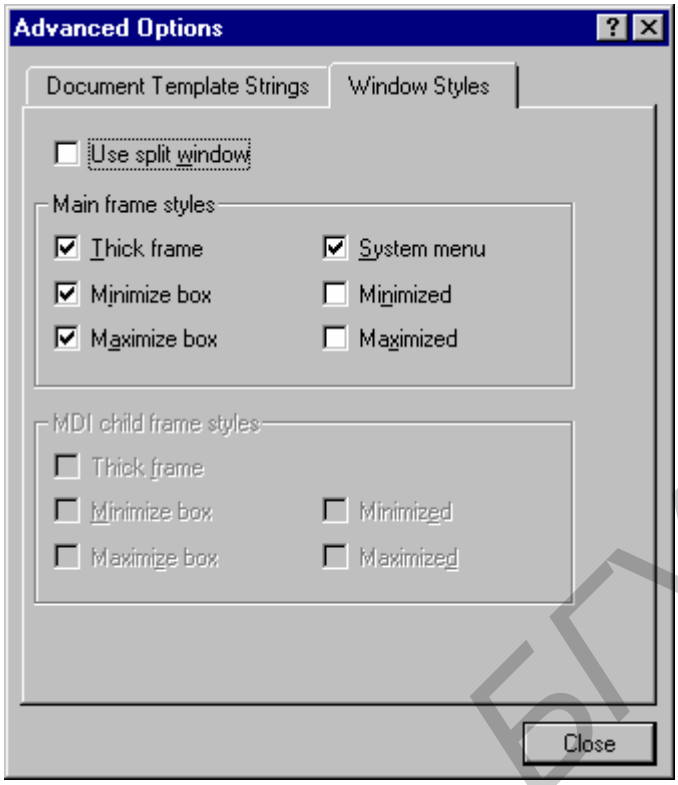

Рис.1.7. Вкладка Window Styles диалогового окна Advanced Options.

Ниже перечислены свойства фрейма, которые можно настраивать во вкладке, о которой идет речь:

- *Thick frame* (утолщенная рамка) – кромки окна утолщены, и можно будет изменять размеры окна стандартным для Windows способом;
- *Minimize box* (кнопка минимизации) – окно имеет кнопку минимизации в правой части строки заголовка.
- *Maximize box* (кнопка максимизации) – окно имеет кнопку максимизации в правой части строки заголовка.
- *System menu* (системное меню) – в левом верхнем углу окна будет установлена пиктограмма вызова системного меню;
- *Minimized* – при запуске приложения окно сворачивается в пиктограмму. Для SDI-приложений выбор этой опции не будет иметь никаких последствий при выполнении приложения в среде Windows 95;
- *Maximized* – при запуске приложения окно разворачивается на весь экран. Для SDI-приложений выбор этой опции не будет иметь никаких последствий при выполнении приложения в среде Windows 95.

После завершения всех манипуляций щелкните на Close для возврата в окно MFC AppWizard – Step 4 of 6.

Шаг 5. Другие опции. Пятый этап создания выполняемого приложения Windows с помощью AppWizard – выбор опций, которые нельзя было отнести по

назначению ни к одному из предыдущих этапов. Диалоговое окно AppWizard – Step 5 of 6 при этом будет выглядеть так, как показано на рис.1.8. Будете ли вы включать в формируемый текст программ приложения комментарии? Редко кто отказывается от этого, тем более что задать такой режим не составляет никакого труда – нужно просто выбрать один из переключателей группы “Would you like to generate source file comments?”

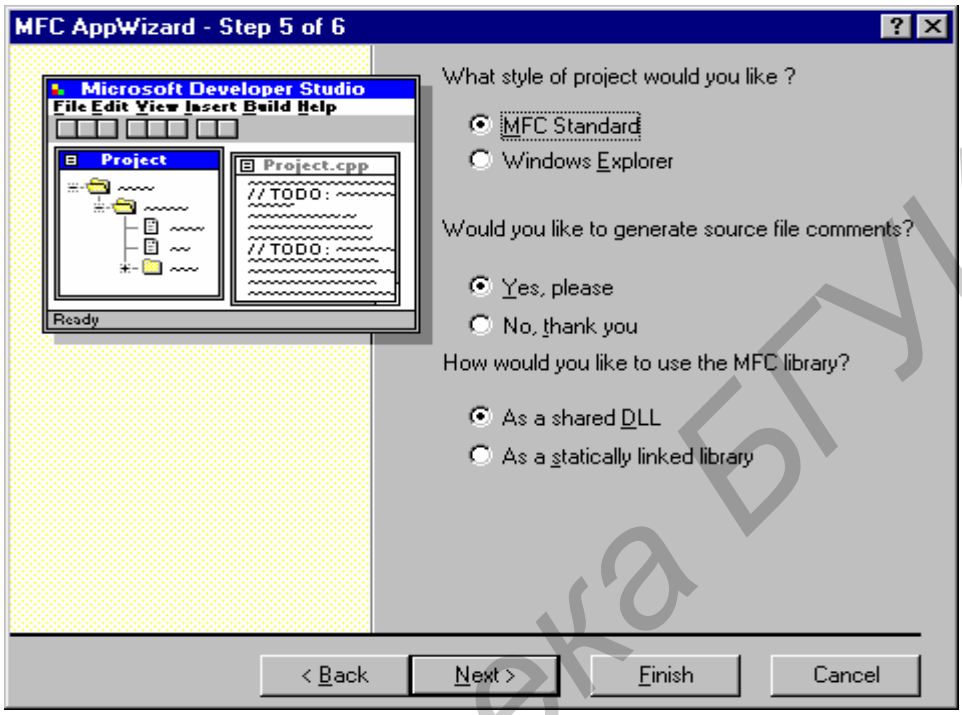

Рис.1.8. Установка опций, определяющих оформление формируемого текста программы и метод связывания программы с модулями библиотеки MFC

Ответ на второй вопрос в этом окне: “Желаете ли вы, чтобы библиотека MFC *была разделяемой, динамически связываемой библиотекой* (shared DLL) или *статически прикомпилированной* (statically linked library)?” – не так очевиден. Динамически связываемая библиотека (DLL – Dynamic-Link Library) представляет собой множество функций, используемых самыми разными приложениями. Использование DLL сокращает объем программы, но несколько усложняет установку продукта. Если вы просто перенесете на другой компьютер выполняемый файл программы, то скорее всего приложение работать не будет, поскольку оно нуждается еще и в соответствующих DLL-файлах. Если же модули библиотеки прикомпонованы статически к выполняемому файлу, то приложение легко перемещается с одного компьютера на другой, поскольку весь выполняемый код сосредоточен в одном файле.

Шаг 6. Имена файлов и классов. И, наконец, последний этап создания выполняемого приложения Windows с помощью AppWizard – подтверждение имен классов и имен файлов, которые создает для вас AppWizard, как это показано на рис.1.9. AppWizard использует имя проекта (в данном случае – sssss) для

формирования имен классов и имен файлов. Нет никакой нужды их изменять. Если в приложении используются классы представления, можно изменить имя класса, наследниками которого являются вновь создаваемые классы. По умолчанию базовым является CView, но многие разработчики предпочитают CScrollView или CEditView. После завершения работы в диалоговом окне Step 6 of 6 необходимо нажать на кнопку Finish.

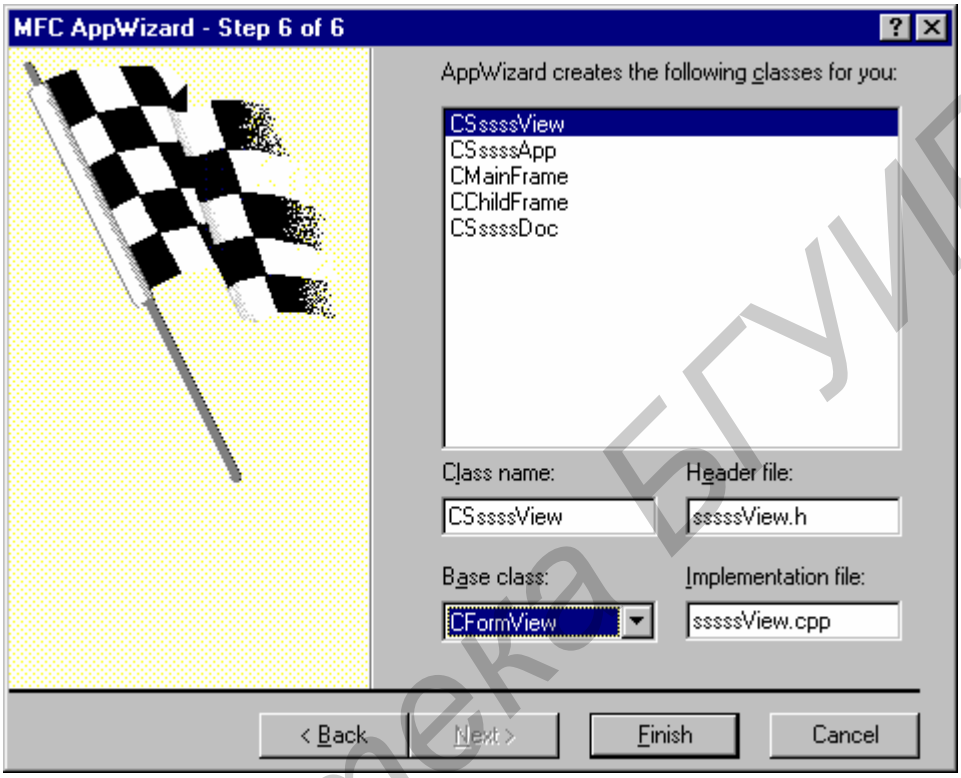

Рис.1.9. Подтверждение имен классов и файлов на последнем этапе создания типового приложения

Шаг 7. Создание приложения. После того, как вы щелкнете на Finish, AppWizard покажет вам в окне New Project Information информацию о новом проекте (рис.1.10). Если что-либо не устраивает, можно вернуться, нажав на кнопку Cancel, и затем последовательно двигаться в обратном порядке по окнам этапов настройки, пока не будет найдено то окно, в котором возможно изменение данной настройки. После уточнения настройки можно повторить путь по шагам AppWizard'а либо сразу принять оставшиеся установки. После чего можно опять взглянуть на окно New project information и дать согласие на генерацию классов. AppWizard создаст необходимые классы и ресурсы.

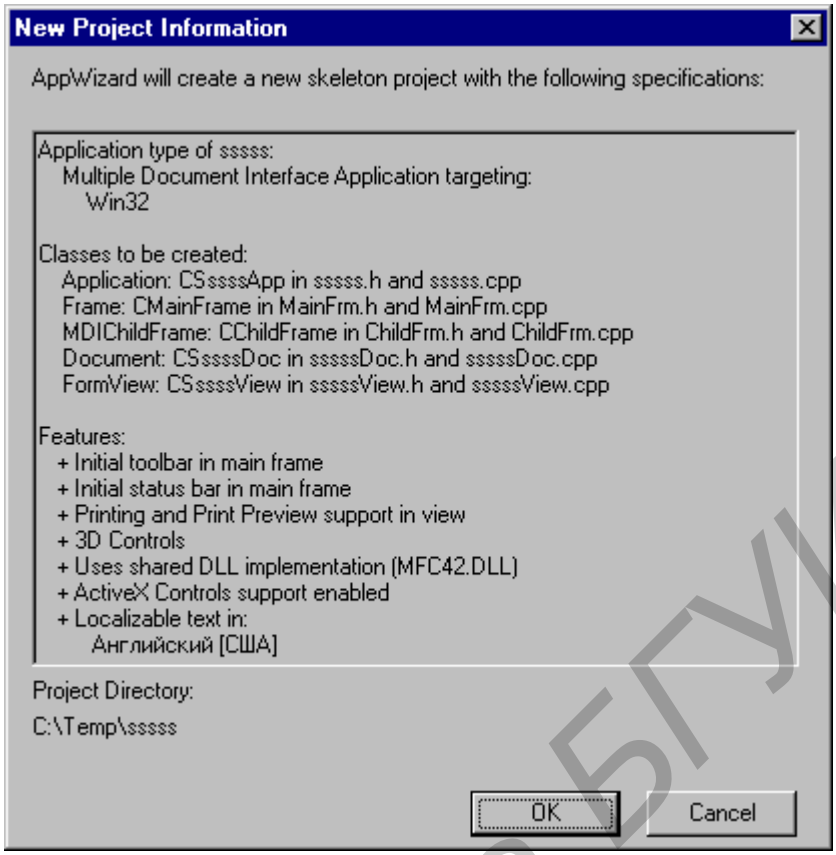

Рис.1.10. Завершающий диалог AppWizard

Приложение 2

Диалоговые окна

Приложение может иметь любое количество диалоговых окон, в которых происходит ввод данных пользователем. Как правило, для каждого диалогового окна в приложении существуют ресурс диалога и класс.

В класс окна включены переменные и функции-члены, ответственные за работу диалога.

Ресурсы диалога создаются посредством редактора ресурсов, с помощью которого возможно включать в его состав необходимые элементы управления и размещать их в необходимом порядке. Класс создается при помощи ClassWizard. Как правило, класс диалогового окна в проекте является производным от класса CDialog, входящего в MFC. ClassWizard также позволяет облегчить работу с элементами управления, расположенными на диалоговом окне. Обычно каждый элемент управления, включенный в состав ресурсов диалога, имеет в классе окна соответствующий член-переменную. Для того чтобы вывести диалоговое окно на экран, нужно вызвать функцию- член его класса. Для того чтобы установить или считать значения элементов управления, необходимо обращаться к членам-переменным класса.

Формирование нового ресурса диалогового окна. Первый шаг процесса организации диалогового окна в приложении – формирование ресурса окна.

Чтобы приступить к формированию ресурсов, необходимо выбрать пункт Insert Resource из меню Visual Studio – появится диалоговое окно Insert Resource, показанное на рис.2.1. Дважды щелкните на элементе Dialog в окне Resource Type – этим вы вызываете редактор диалогового окна, который выводит на экран заготовку окна, как это показано на рис.2.2.

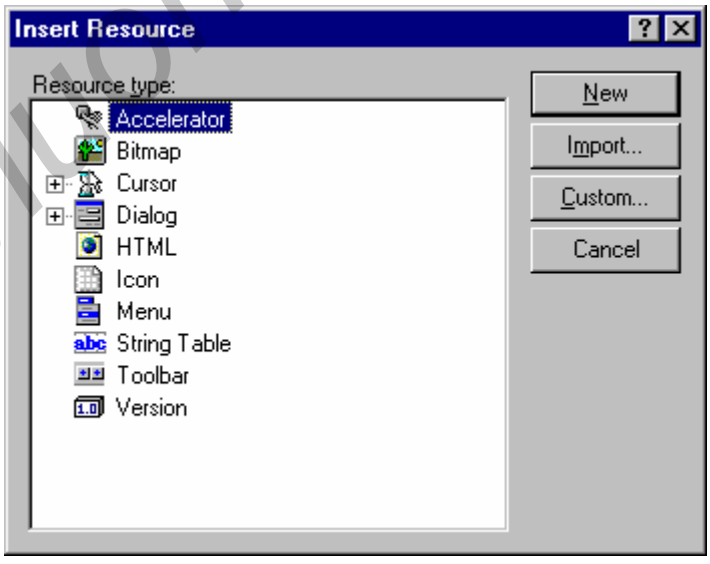

Рис. 2.1. Добавление ресурса диалогового окна

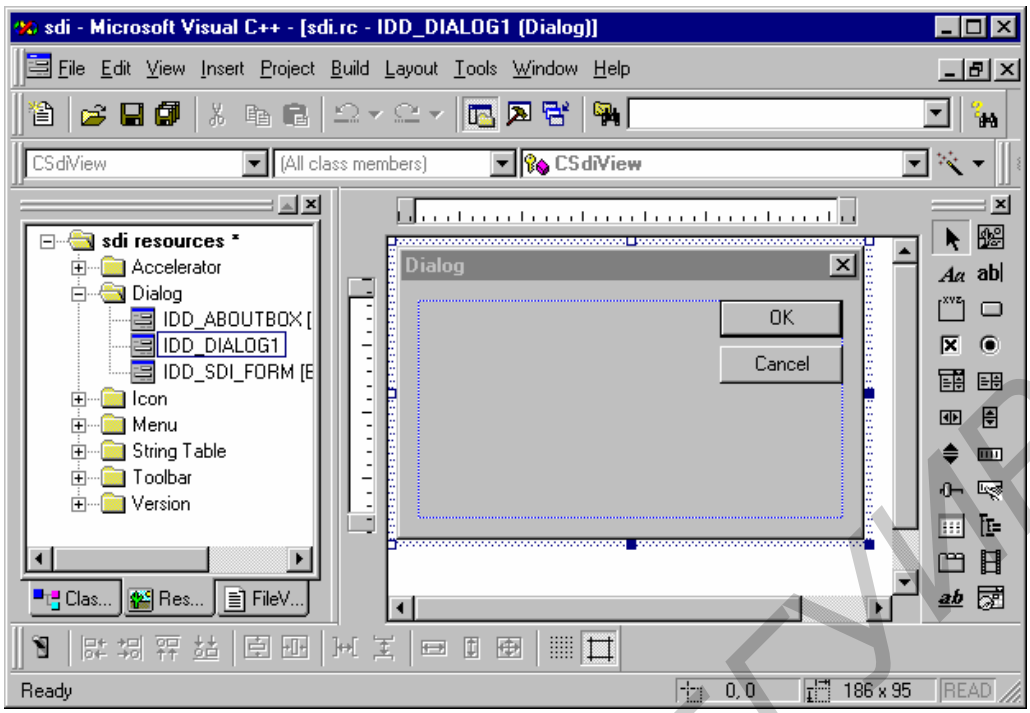

Рис.2.2. Заготовка диалогового окна

Вызовите на экран диалоговое окно Dialog Properties для вновь создаваемого диалогового окна, выбрав в меню View\Properties или щёлкнув правой кнопкой мыши на диалоговом окне. В поле Caption (Надпись) введите заголовок диалога, как это показано на рис.2.3.

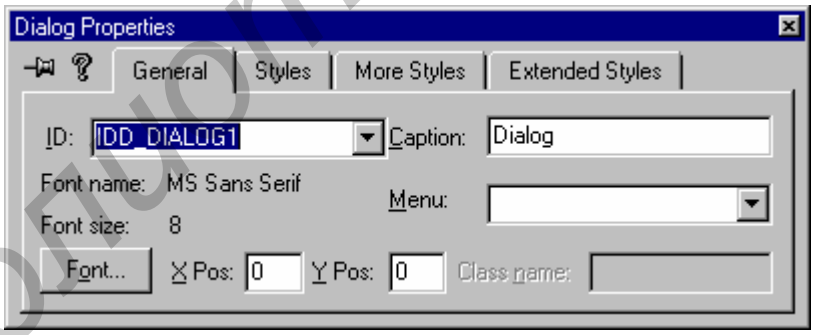

Рис.2.3. Окно свойств (Dialog Properties) создаваемого диалогового окна

Создание класса диалогового окна. Когда формирование ресурсов диалогового окна будет завершено, вызовите на экран диалоговое окно мастера ClassWizard. Для этого нужно выбрать View\ClassWIzard. Мастер ClassWizard обнаружит новый диалог и предложит создать новый класс, как это показано на рис. 2.4. Установите переключатель *Create a new class* (создать новый класс) и щелкните на OK. Появится новое диалоговое окно New Class (новый класс), которое показано на рис.2.5. В поле Name (имя) введите имя нового класса (на-

пример, CSDIDialog) и щелкните на OK. После этого ClassWizard создаст новый класс, подготовит файл текста программы SDIDialog.cpp и файл заголовка SDIDialog.h и включит их в состав проекта.

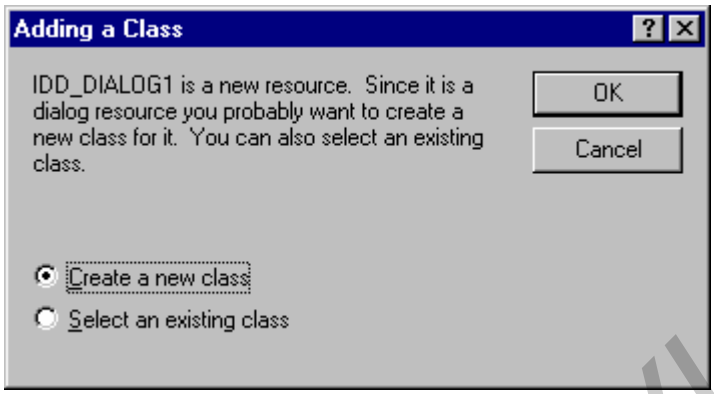

Рис.2.4.Диалог предложения создать или выбрать класс для нового диалогового окна

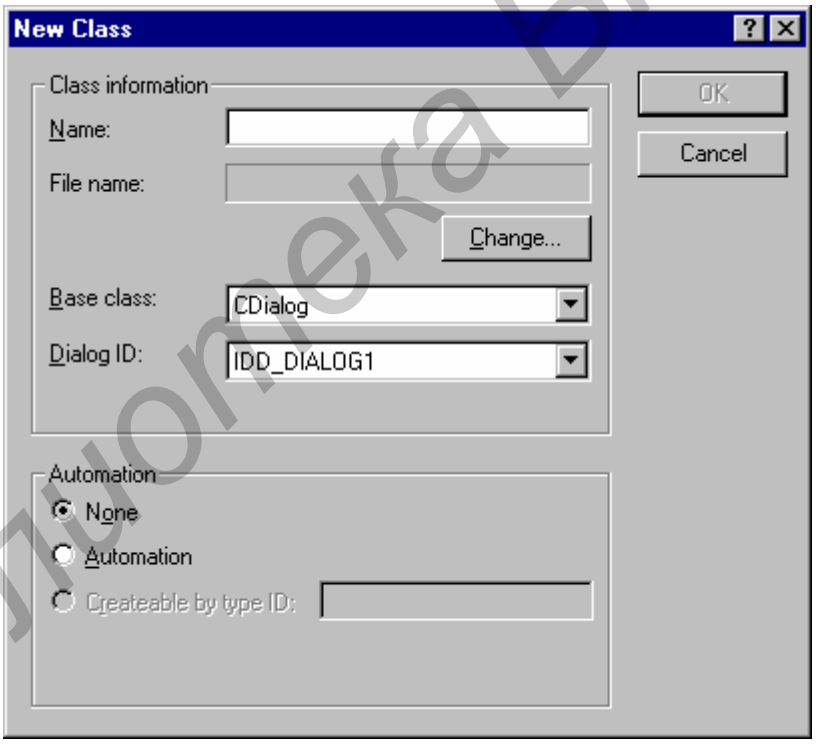

Рис.2.5.Создание класса для диалогового окна

Модальные и немодальные диалоговые окна. Большинство диалоговых окон, которые приходится включать в состав приложения, относятся к так называемым *модальным* окнам. Модальное окно выведено *всегда* поверх всех остальных окон на ране. Пользователь должен поработать в этом окне и обязательно закрыть его, прежде чем приступить к работе в любом другом окне этого же приложения. Примером может служить окно, которое открывается при выборе

команды File\Open любого приложения Windows.

Немодальное диалоговое окно позволяет пользователю, не закончив работы с ним, работать в других окнах приложения, выполнить там необходимые действия, затем снова вернуться в немодальное окно и продолжить работу. Типичными немодальными окнами являются окна, которые открываются при отработке команд Edit\Find (Правка\Поиск) и Edit\Replace (Правка\Замена) во многих приложениях Windows.

Приложение 3

Меню

Создание меню. AppWizard формирует в заготовке для MDI приложения два меню (для SDI одно). Если ни один файл не открыт в приложении, выводится меню IDR_MAINFRAME; если же хотя бы один документ открыт, выводится меню IDR_MDITYPE. Обратите внимание, что меню IDR_MAINFRAME не имеет пунктов (выпадающих меню) Edit и Window, а само меню File значительно короче, чем в IDR_MDITYPE. В первом варианте в нем есть только пункты New (Новый), Open (Открыть), Print Setup (Настройка принтера), Recent File (Последние файлы) и Exit (Выход) (рис.3.1).

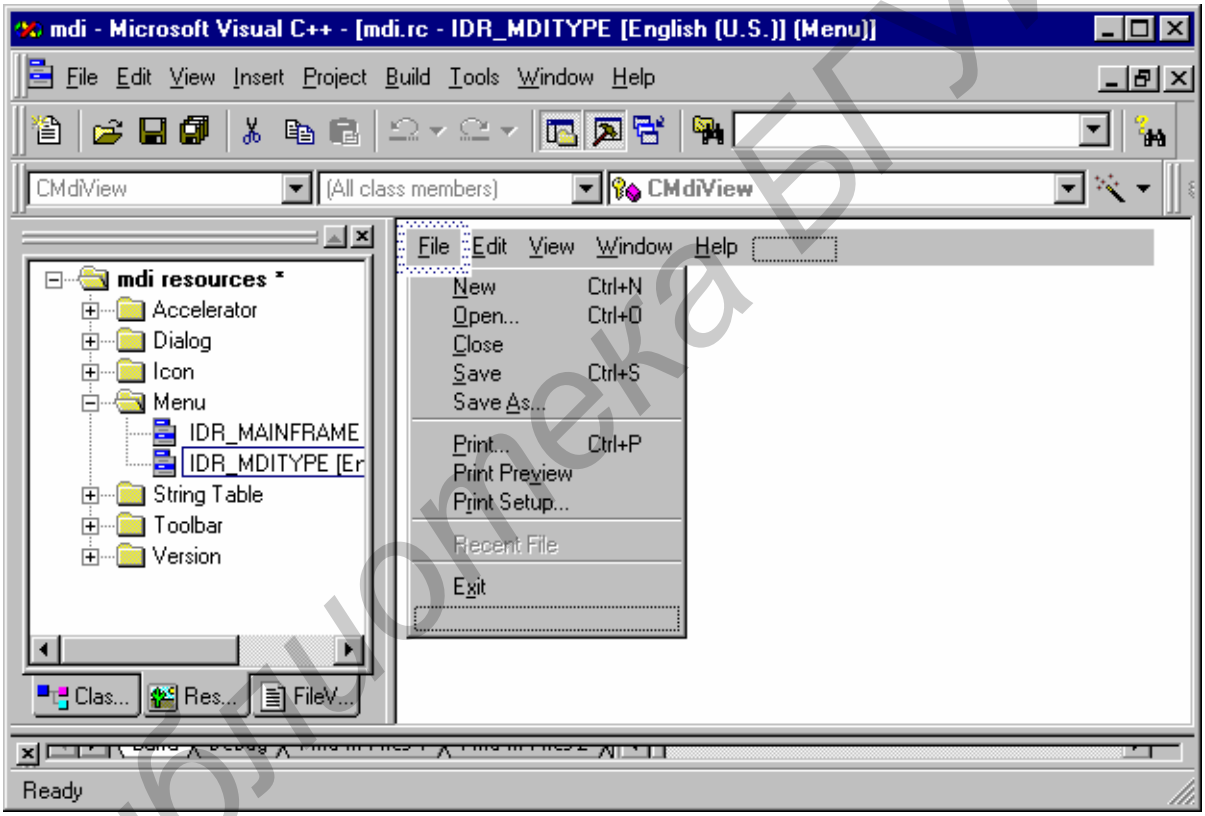

Рис.3.1.Редактирование ресурсов типа меню

Выберите из списка ресурсов меню. Появляется изображение меню. Необходимо выбрать пункт, в который необходимо добавить новую команду. В конце дочерних пунктов есть свободное место для нового пункта. Вызывается выпадающее (popup) меню путем нажатия правой кнопки мыши. Далее выбирается пункт Properties. В поле Caption вводится текст, который будет отображаться при выводе меню на экран.

Каждый пункт меню имеет свой идентификатор, посредством которого программа связывается с ним. Данный идентификатор вводится в поле ID диа-

лога Properties. Если оставить это поле пустым, то Visual Studio само автоматически назначит уникальный идентификатор.

Для создания и подключения клавиш-акселераторов к пунктам меню (и другим элементам управления) необходимо развернуть узел Accelerator в дереве ресурсов и выбрать необходимую таблицу акселераторов. Далее требуется найти пустую строку в таблице и выбрать пункт Properties в popup-меню. В поле *ID* необходимо выбрать идентификатор требуемого пункта меню. В поле *Key* вводится символ, который будет использоваться в акселераторе. Также необходимо указать клавиши-модификаторы (Ctrl, Alt, Shift), которые тоже используются в акселераторе.

Приложение 4

Панель инструментов

Генерируемая AppWizard панель инструментов (Toolbar) содержит пиктограммы для наиболее распространенных команд. Однако при разработке часто требуется удалить или добавить пиктограммы. Для работы с панелью инструментов необходимо развернуть узел ToolBar в дереве ресурсов и выбрать необходимую панель – откроется окно редактирования панели инструментов (рис. 4.1).

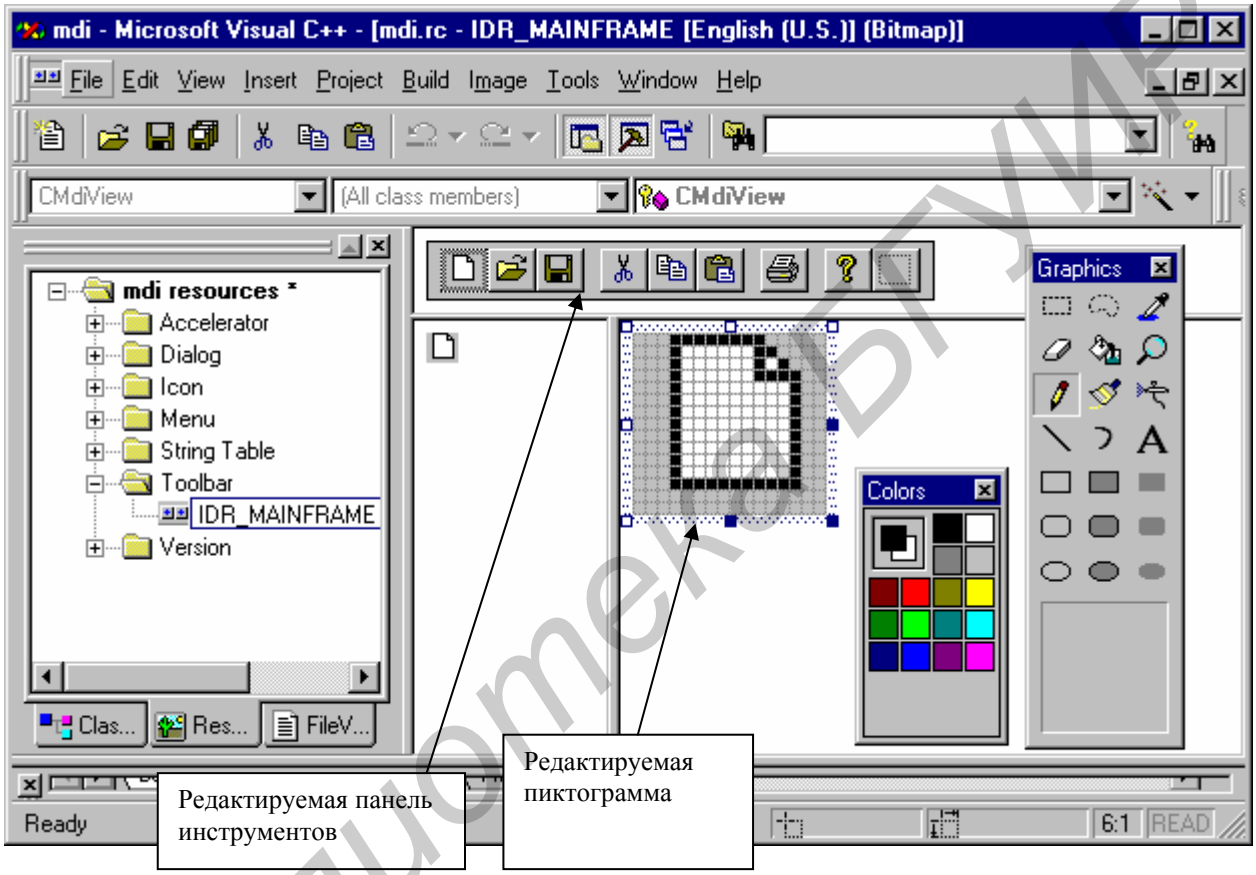

Рис.4.1. Редактирование панели инструментов

После того, как окно редактора панелей инструментов будет открыто, удаление пиктограмм с панели инструментов сводится к простому перетаскиванию их с панели на свободное место в окне.

Добавление пиктограмм на панель инструментов. Процедура включения новой пиктограммы в панель инструментов состоит из двух этапов. На первом из них следует нарисовать изображение пиктограммы, а на втором – вы должны связать команду с новой пиктограммой. Приступая к созданию изображения новой пиктограммы, необходимо выбрать заглушку пустой пиктограммы, расположенной на формируемой панели инструментов. Изображение пустой пиктограммы в увеличенном масштабе появится в окне редактирования.

Далее выведите на экран окно свойств Properties и назначьте пиктограмме соответствующий идентификатор команды (рис.4.2).

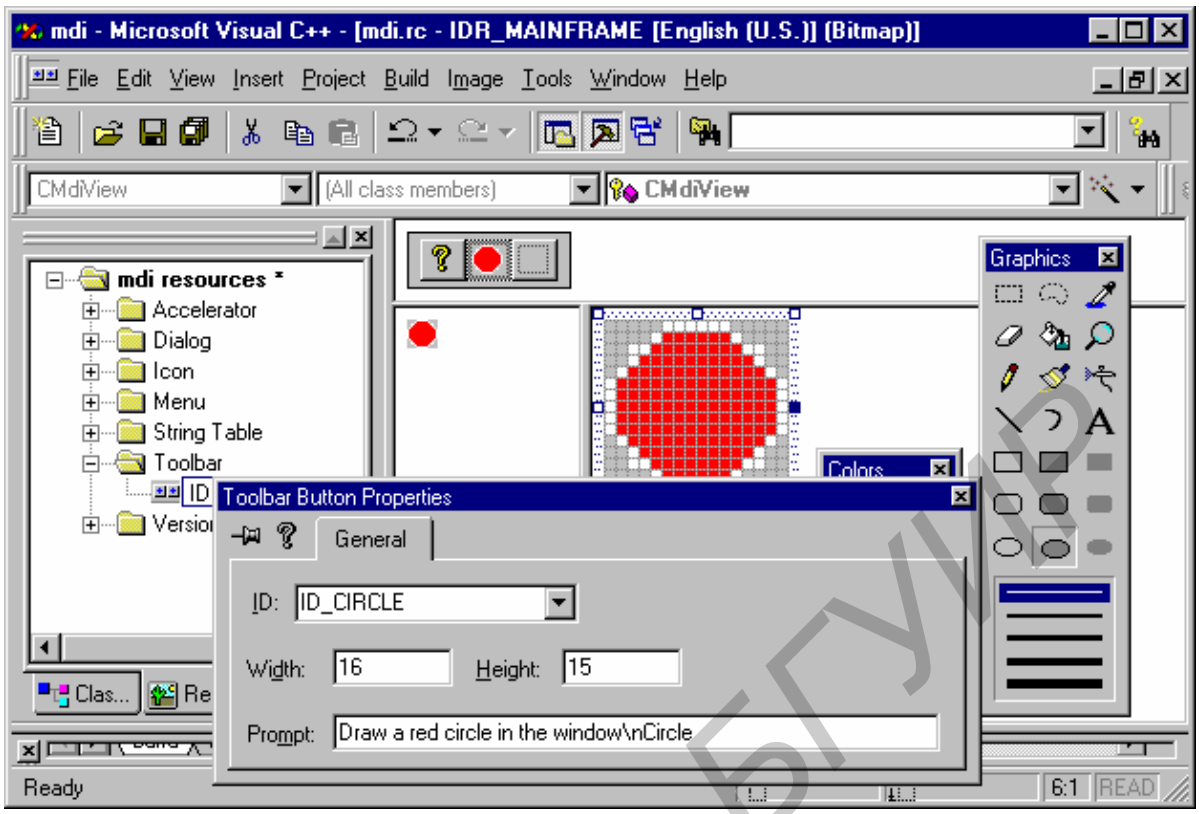

Рис.4.2. Определение свойств пиктограммы

Приложение 5

Генерация приложения, связанного с базой данных

Процесс создания приложений для работы с базой данных отличается от процесса создания простого приложения лишь выбором на втором этапе App-Wizard.

В данном диалоговом окне необходимо выбрать либо *Database view without file support* (если нет необходимости работать с другими файлами), либо *Database view with file support* (рис.5.1). Для определения источника данных необходимо нажать кнопку Data Source – появится диалог Database option. На данном этапе необходимо определить способ доступа к данным (ODBC, DAO, OLEDB), после чего в соответствующем поле выбрать источник данных (рис.5.2).

После окончания работы с диалогом необходимо выбрать таблицы, данные которых будут доступны программе (рис.5.3). После выбора источника данных и таблиц процесс выбора возвращается ко второму этапу AppWizard.

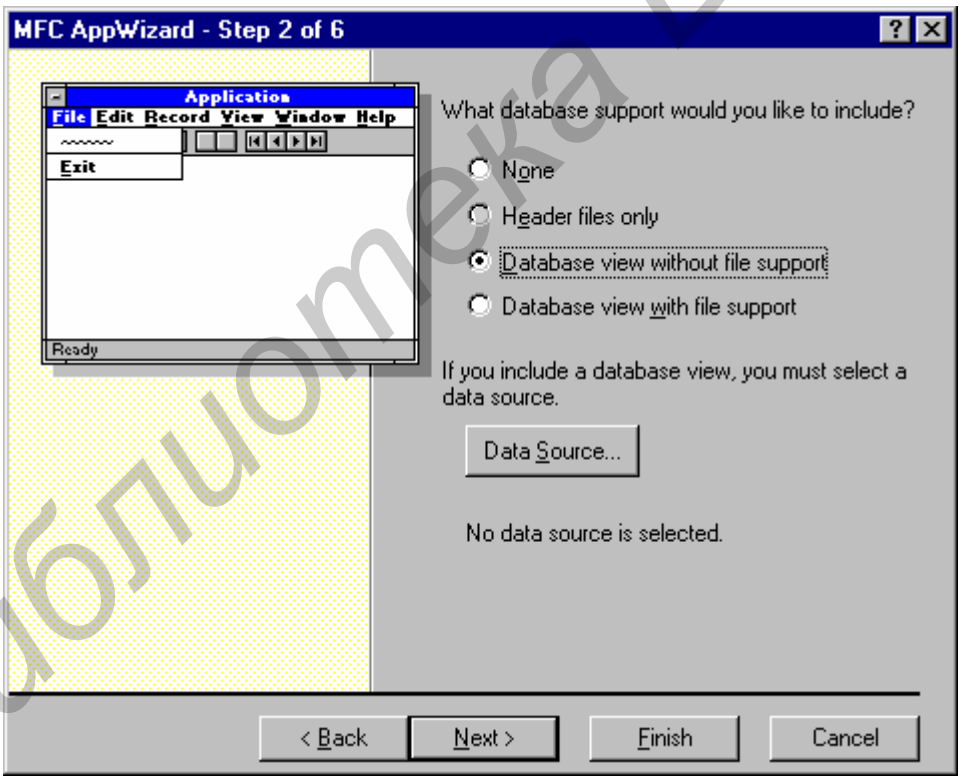

Рис. 5.1. Диалог поддержки работы проекта с базой данных

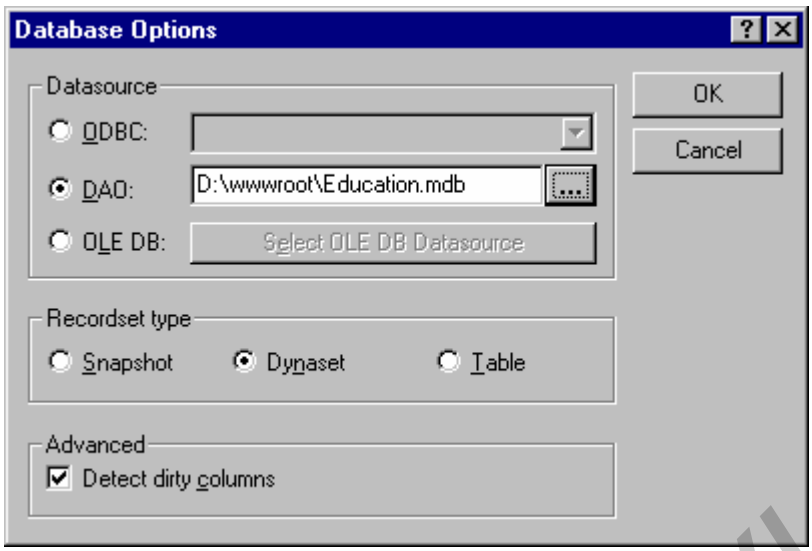

Рис.5.2. Выбор базы данных Education.mdb

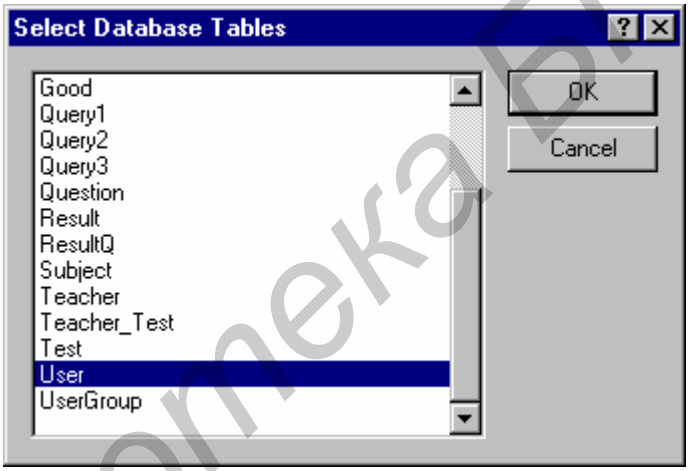

Рис.5.3. Выбор из источника данных таблицы User, которая будет использоваться в создаваемом приложении

Св. план 2001, поз. 88

Учебное издание

Комличенко Виталий Николаевич,
Живицкая Елена Николаевна,
Соколов Сергей Александрович и др.

ЛАБОРАТОРНЫЙ ПРАКТИКУМ
по курсу **«Визуальные средства разработки приложений»**
для студентов специальности 40 01 02-02
«Информационные системы и технологии в экономике»

Редактор Н.А. Бебель
Корректор Е.Н. Батурчик

Подписано в печать | | Формат 60x84 1/16.
Бумага | Печать офсетная. | Усл. печ. л.
Уч.-изд. л. 5,5 | Тираж 150 экз. | Заказ

Издатель и полиграфическое исполнение:
Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
Лицензия ЛП №156 от 05.02.2001
Лицензия ЛВ №509 от 03.08.2001
220013, Минск, П. Бровки, 6