Пользовательский интерфейс

ЧАСТЬ

I

В этой части...

ГЛАВА

1

Первое приложение MFC

В этой главе...

Являясь сторонником обучения программированию на примерах создания реальных приложений, автор не любит длинных и нудных глав, во всех подробностях описывающих каждое окно и каждую кнопку панели инструментов среды разработки. Но в связи с тем, что иногда очень полезно иметь краткое описание среды разработки, рассмотрим ее на примере создания простого демонстрационного приложения.

Как уже было сказано, эта глава предназначена для тех, кто не обладает достаточным опытом работы в среде Visual Studio. Тем, кто таким опытом обладает и желает сразу перейти к изучению библиотеки MFC, можно переходить непосредственно к главе 2, "Документы и представления SDI", но если среда разработки Visual C++ или Visual Studio .NET знакома читателю недостаточно, то ознакомиться с этой главой было бы очень полезно, поскольку здесь рассматривается множество задач, которые неоднократно будут встречаться на протяжении всей книги. К таким задачам относятся: работа со страницей Start Page (Начальная страница), создание диалогового приложения, добавление обработчика события, добавление элемента управления в шаблон диалогового окна при помощи редактора диалоговых окон (Dialog Editor), использование DDX для связи элемента управления с переменной-членом, отладка и исправление простых ошибок.

Поприветствуем новый Visual Studio

Сначала запустим редактор Visual Studio (он должен находиться в меню кнопки Start (Пуск), пункты Programs (Программы) Microsoft Visual Studio .NET). Тех, кто знаком с предыдущими версиями Visual Studio, ожидает множество неожиданностей как приятных, так и не очень. На рис. 1.1 можно заметить первое радикальное изменение среды разработки Visual Studio — страницу Start Page (Начальная страница).

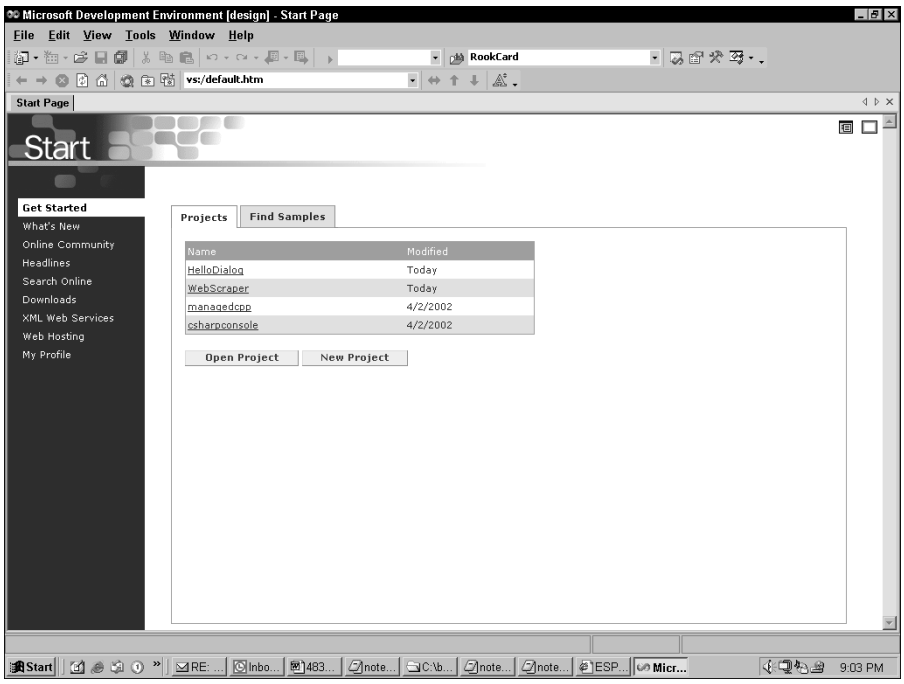


Рис. 1.1. Все начинается со страницы Start Page

Хотя для опытного разработчика начальная страница не особенно полезна, начинающим программистам Visual Studio она способна оказать неоценимую помощь. Начальная страница способна даже отобразить сообщение, оповещающее о выходе новой версии Visual Studio (или сервисного пакета), которой можно будет заменить текущую версию. В первую очередь обратите внимание на вертикальные вкладки, расположенные по левой стороне страницы. Ниже приводится краткое описание наиболее важных из них.

- Get Started (Начать). На первоначальном этапе изучения Visual Studio эта вкладка, вероятно, будет иметь первостепенную важность, поскольку содержит еще две вкладки, которые программисты, начинающие изучение данной среды разработки, используют особенно часто: Projects (Проекты) и Find Samples (Поиск примеров).
  - Как можно заметить на рис. 1.1, вкладка Projects (Проекты) содержит список проектов, над которыми работает пользователь. Кроме того, на этой вкладке расположены две кнопки, позволяющие открыть существующий проект или создать новый. Довольно скоро этими кнопками предстоит воспользоваться.
  - Возможно одним из самых интересных нововведений страницы Start Page является вкладка Find Samples (Поиск примеров) (рис. 1.2). Вспомните, как часто при разработке проекта приходится рыться в библиотеке MSDN (Microsoft Developer Network), которая занимает теперь два компакт-диска, чтобы найти необходимый пример кода. Эта вкладка позволяет не только осуществлять поиск в тексте статей библиотеки, но и отфильтровывать статьи по языкам программирования. Выбрав переключатель Type (Тип), можно использовать в качестве критерия фильтра такие категории, как Controls, Forms или Web. Как можно заметить, корпорация *Microsoft* потратила немало усилий, чтобы сделать эту среду разработки по возможности более легкой в употреблении, а следовательно, и более продуктивной.

На рис. 1.2 представлен результат поиска примеров применения расширений управляемого кода для языка Visual C++. (*Расширения управляемого кода* (managed extensions) — это средства создания кода для платформы .NET в среде разработки Visual C++. Этой теме посвящены главы 41–44.)

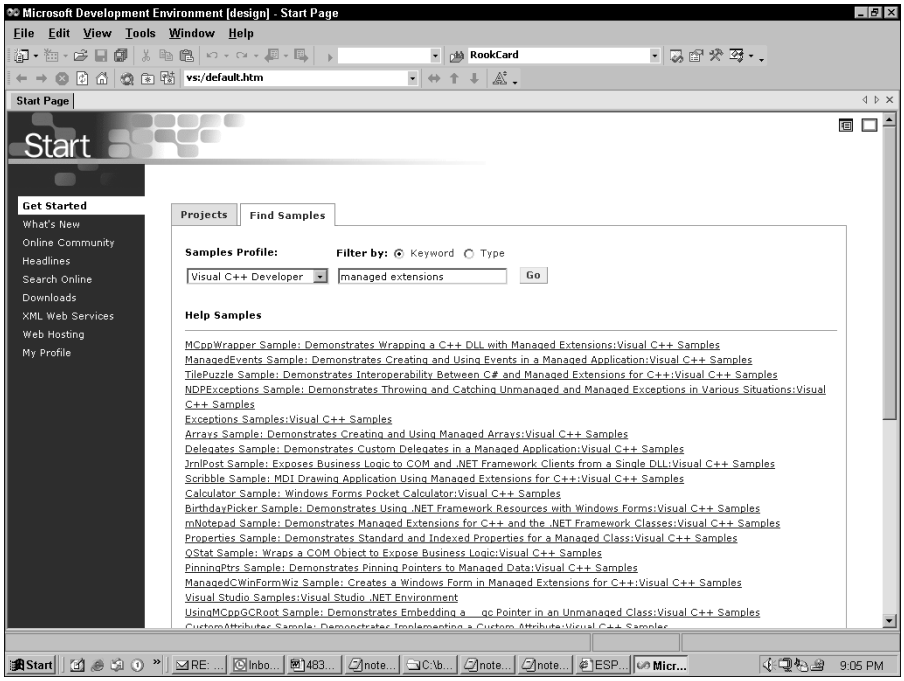
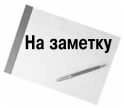


Рис. 1.2. Вкладка Find Samples *(Поиск примеров) позволяет быстро найти примеры для различных языков программирования*



Большинство вкладок на странице Start Page требуют подключения к Internet, поскольку они не только отображают, но и позволяют искать самую свежую информацию.

- What's New (Что нового). На самом деле замечательным свойством этой вкладки является то, что она, подключившись к Web-сайту *Microsoft*, отображает только самую свежую информацию, в которой заинтересованы разработчики для Windows или Web. Это существенно экономит время, избавляя от необходимости кропотливо перерывать этот чрезмерно перенасыщенный сайт. Здесь кроме списка ресурсов по Visual Studio содержится вкладка Partner Resources (Ресурсы партнеров) и, что более важно для разработчиков, вкладка Product Information (Информация о продуктах), которая содержит информацию о различных инструментальных средствах разработки для Visual Studio и о функциях API. Обратите также внимание, вкладка What's New содержит раскрывающийся список Filter (Фильтр), позволяющий ограничить отображаемую информацию определенным языком или инструментом.
- Online Community (Сетевое сообщество). Кто не испытывал мучений пытаясь вспомнить точный адрес необходимой группы новостей или Web-сайта? Практически все. И случается это обычно именно тогда, кода ответ на возникший вопрос нужен

срочно, *прямо сейчас*. Вкладка Online Community (рис. 1.3) содержит перечень всех групп новостей, Web-сайтов и даже адресов чат сообщества. Как и на предыдущих вкладках, здесь тоже можно отбирать содержимое списков при помощи фильтра, расположенного вверху страницы. В списке указан даже новый сайт автора, The Code Channel (адрес `http://www.theCodeChannel.com`), на котором можно найти полезные советы, демонстрационные приложения, а также список опечаток, обнаруженных в этой книге.

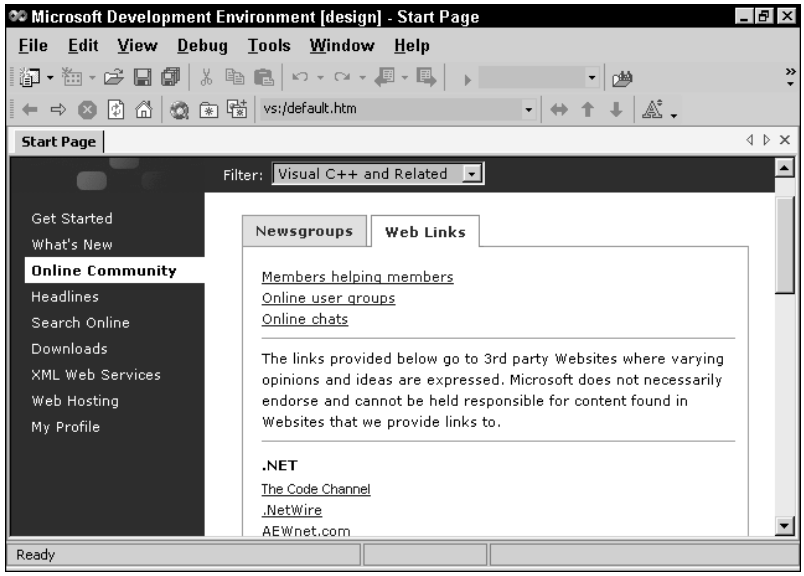


Рис. 1.3. Вкладка Online Community обеспечивает доступ к Web-ресурсам

- Headlines (Заголовки). Настоящая вкладка содержит последние новости и информацию от *Microsoft*. Здесь расположено еще две вкладки: Technical Articles (Технические статьи) и Knowledgebase Articles (Статьи базы знаний). Теперь можно не искать эти неуловимые статьи в MSDN!
- Search Online (Поиск по сети). Это обычная поисковая страница, осуществляющая поиск в библиотеке MSDN.
- Downloads (Загрузка). Данная страница содержит три вкладки: Downloads (Загрузка), Code Samples (Примеры кода) и Reference (Ссылки). Все они содержат ссылки на загружаемые ресурсы. На этой странице, в основном, представлена та же информация, которую при желании можно найти на странице Downloads библиотеки MSDN, но здесь поиск организован несколько лучше.
- XML Web Services (Web-службы XML). Не знаете, что такое Web-служба? Подыскиваете Web-службу для своей компании? Эта вкладка содержит все необходимое, чтобы узнать об этой новой технологии. Здесь также расположена вкладка, на которой можно зарегистрировать в поисковой системе собственноручно созданные Web-службы, предоставив их для поиска и использования другими компаниями!
- Web Hosting (Размещение в Web). Немного "меркантильная", по мнению автора, вкладка, содержащая информацию для компаний и частных лиц, собирающихся создать и разместить собственный Web-сайт.

- My Profile (Мой профиль). Страница My Profile позволяет настроить среду разработки Visual Studio так, чтобы она соответствовала вкусам и предпочтениям конкретного разработчика. Страница содержит параметры, позволяющие выбрать один из базовых профилей разработчика (C++, C#, студент и т.д.), раскладку клавиатуры, размещение окон и фильтр тем помощи. Можно выбрать страницу, которая будет отображаться по умолчанию при загрузке Visual Studio. Это может быть либо рассматриваемая в данный момент страница Start Page, либо последний проект, либо диалоговое окно New Project (Новый проект).

Создание первого проекта Visual Studio

Теперь приступим к созданию кода! Сначала создадим очень простое приложение типа Hello World. Для этого используем (читатель уже догадался) страницу Start Page. Выберите страницу Get Started (Начать), перейдите на вкладку Projects (Проекты) и щелкните на кнопке New Project (Новый проект). Вместо этого можно было бы выбрать в меню File (Файл) пункты New (Создать), Project (Проект) или нажать комбинацию клавиш <Ctrl+Shift+N>. В любом случае на экране появится диалоговое окно New Project (рис. 1.4), позволяющее создавать все типы проектов Visual Studio.

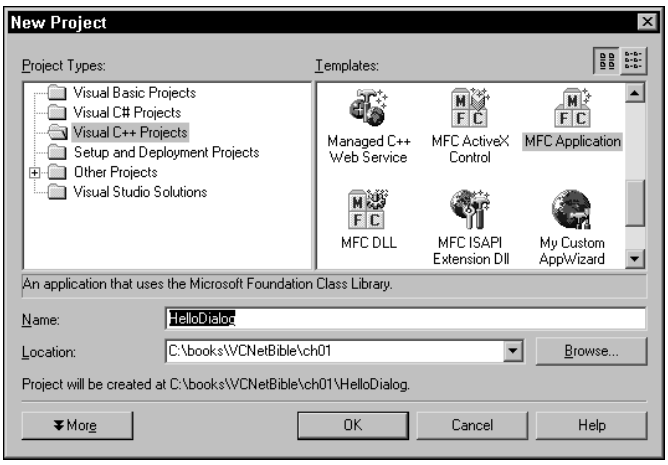


Рис. 1.4. Это окно позволяет создать около сотни различных типов проектов

1. Вначале необходимо выбрать тип проекта. В данном случае предстоит создать проект Visual C++, поэтому выберите его в списке Project Types (Тип проекта). (Тем, кто плохо знаком с графическим интерфейсом, не стоит отчаиваться, на рис. 1.4 продемонстрированы все параметры, которые предстоит выбрать.)
2. Прокрутите список Templates (Шаблоны) вниз и выберите из него шаблон MFC Application (Приложение MFC).
3. Введите в поле Name (Имя) имя проекта. В данном случае выбрано **`HelloDialog`**.
4. Подтвердите отображенное в поле Location (Расположение) расположение создаваемого проекта или укажите новое, введя его с клавиатуры или выбрав с помощью кнопки Browse (Выбрать). Выбор места хранения проекта зависит от предпочтений пользователя. В данном случае все проекты сохраняются в папке текущей главы.

5. Подтвердите это, выбрав переключатель Close Solution (Закрыть решение). Этот элемент управления устанавливает параметр сохранения вновь созданного проекта. *Решение* (solution) можно рассматривать как контейнер для одного и более проектов. Как правило, решения содержат только один проект. Но более опытные программисты при создании достаточно сложных приложений предпочитают хранить все проекты, предназначенные для данной системы, в приделах одного решения. В данном случае в демонстрационных целях оставьте переключатель Close Solution выбранным, что обеспечит создание нового решения и отказ от добавления данного проекта к текущему решению.
6. После щелчка на кнопке OK будет запущен мастер, ассоциированный с выбранным типом проекта, в данном случае MFC Application Wizard (мастер приложений MFC), показанный на рис. 1.5.

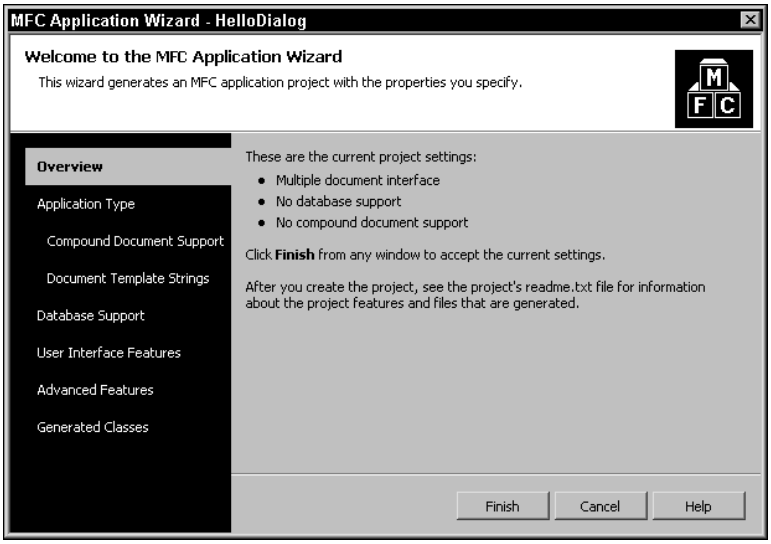


Рис. 1.5. Мастер MFC Application Wizard позволяет при помощи нескольких щелчков мышью создать новый проект, обладающий базовыми функциональными возможностями

Первая страница мастера MFC Application Wizard называется Overview (Обзор). Она отображает параметры создаваемого проекта, установленные в настоящий момент, позволяя пользователю удостовериться в их корректности, прежде чем он щелкнет на кнопке Finish (Готово). Остальные вкладки, расположенные вертикально с левой стороны диалогового окна мастера, предназначены для изменения различных параметров создаваемого проекта.

Не будем на данном этапе подробно рассматривать каждый из параметров, поскольку большинство из них сейчас не нуждается в изменении, а впоследствии, по мере приобретения опыта работы с Visual C++, изучим их досконально. Сейчас ограничимся лишь кратким обзором параметров, наиболее часто встречающихся на протяжении книги.

1. Перейдите на вкладку Application Type (Тип приложения) и выберите в группе Application type переключатель Dialog-based (Диалоговое) (рис. 1.6). Более подробная информация по этой теме приведена в главах 2, “Документы и представления SDI” и 3, “Приложения MDI”.

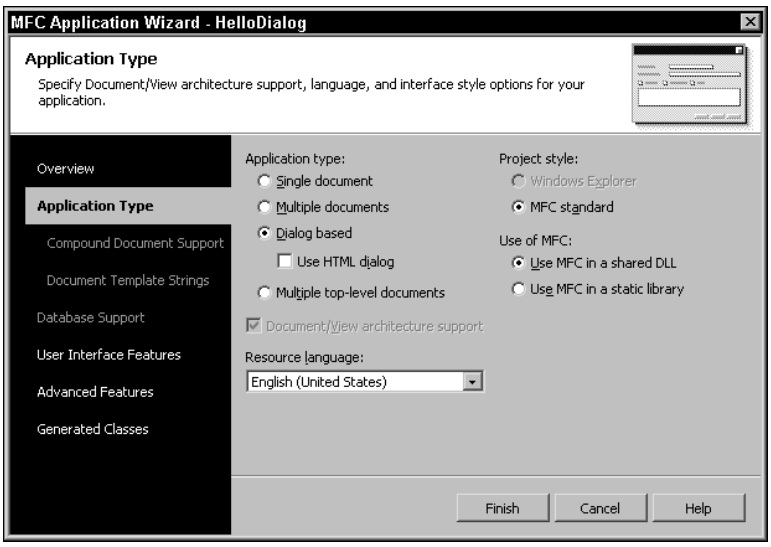


Рис. 1.6. Создавая приложение MFC с помощью мастера, можно выбрать любой из четырех основных пользовательских интерфейсов

Группа переключателей Project style (Стиль проекта), изменение которой окажется невозможным при выборе диалогового проекта, представляет интерес только в том случае, если создается приложение SDI (Single Document Interface — *однодокументный интерфейс*) или MDI (Multiple Document Interface — *многодокументный интерфейс*). Этот параметр позволяет создать пользовательский интерфейс, похожий на проводник Windows (Explorer), где с одной стороны окна расположено древовидное представление (tree view), а с другой — представление в виде списка (list view). Более подробная информация об этих элементах управления приведена в главе 7, "Элементы управления древовидное представление и список".

2. Перейдите на вкладку User Interface Features (Возможности пользовательского интерфейса) (рис. 1.7). В зависимости от типа проекта здесь будут доступны те или иные параметры. Поскольку создаваемое приложение является диалоговым, группы Child frame styles (Стили дочерних форм) и Toolbars (Панели инструментов) окажутся заблокированы. Единственными доступными окажутся параметры границ диалогового окна, системного меню и заголовка. Единственное изменение, которое необходимо сделать на этой вкладке (рис. 1.7), — заполнить поле Dialog title (Заголовок диалогового окна). В данном случае введите: "**`My First MFC Application`**" (Мое первое приложение MFC).
3. Чтобы создать новый проект, щелкните на кнопке Finish (Готово).

При создании диалогового приложения среда разработки Visual Studio автоматически отобразит панели Resource View (Ресурсы), Dialog Editor (Редактор диалоговых окон) и Toolbox (Панель инструментов) (рис. 1.8). (Обратите внимание, если пользователь уже запускал среду Visual Studio и переместил эти окна, то они могут выглядеть иначе.) Применение редактора диалоговых окон и панели инструментов обсудим несколько позже, а пока ознакомимся с разнообразными представлениями (views), которыми можно воспользоваться для просмотра нового проекта.

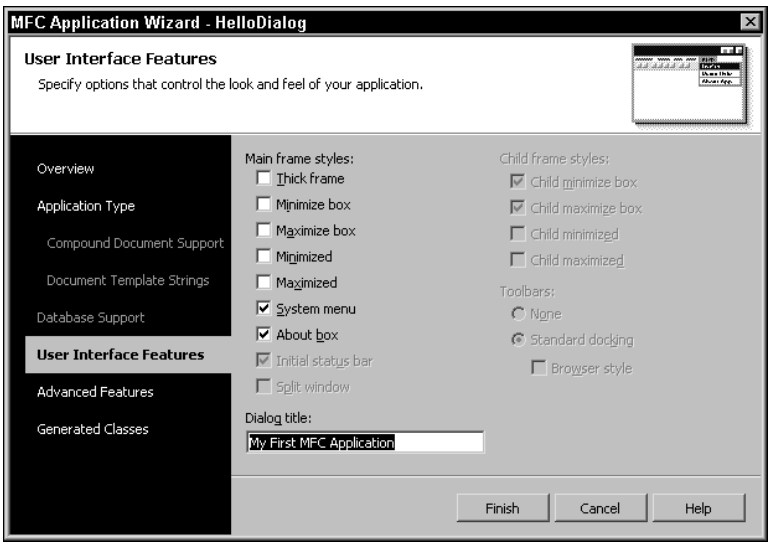


Рис. 1.7. Создавая приложение MFC с помощью мастера, можно установить стили границ окна, отобразить или скрыть такие элементы пользовательского интерфейса, как кнопки минимизации и развертывания окна, системное меню, а также окно About (О программе)

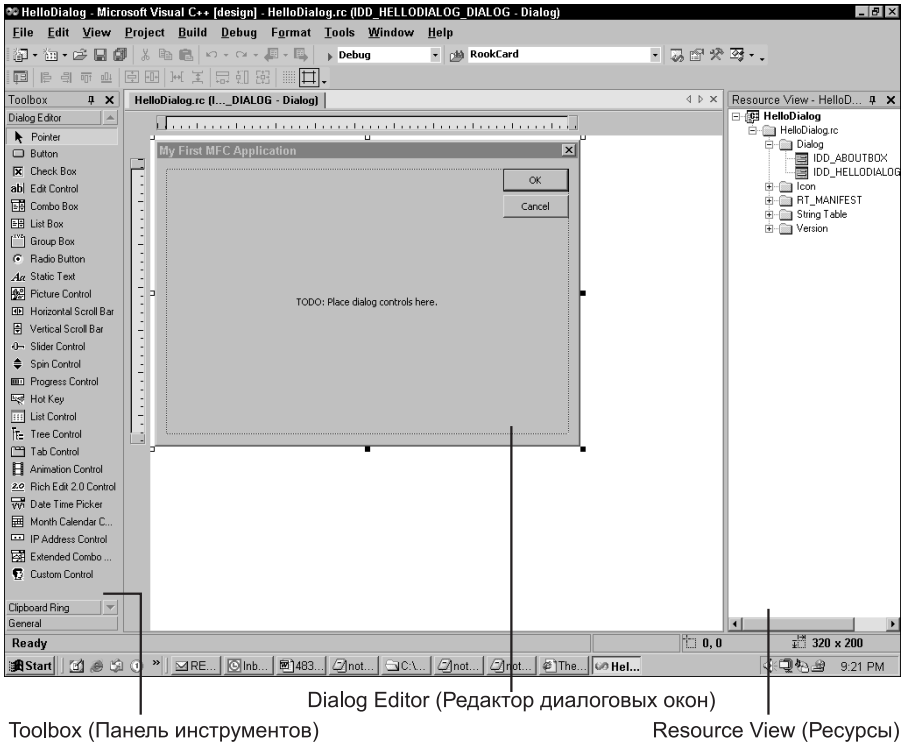
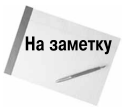


Рис. 1.8. Обычно, при создании диалогового приложения, среда Visual Studio открывает панели Resource View, Dialog Editor и Toolbox, чтобы пользователь сразу мог приступить к редактированию стандартного диалогового окна

Обзор представлений

Решения и проекты Visual Studio состоят из очень большого количества различных файлов и классов, которые автоматически создаются от имени разработчика. В этом разделе описывается, как с помощью разнообразных представлений (доступных в меню View (Вид)) можно быстро и эффективно разобраться со всеми ними.

- Resource View (Ресурсы). По умолчанию это представление (см. рис. 1.8) выглядит как докер, расположенный в левой части окна Visual Studio и отображает все ресурсы текущего проекта (диалоговые окна, меню, текстовые строки и т.д.).
- Solution Explorer (Проводник решений). Данное представление (рис. 1.9) используют чаще всего. Проводник решений находится в меню View и отображает все файлы, связанные с текущим решением или проектом. Файлы разделены по категориям: *файлы исходного кода* (source files), *файлы заголовков* (header files) и *файлы ресурсов* (resource files), что позволяет упростить поиск. Открыть файл очень просто, достаточно дважды щелкнуть на нем мышью. Подобно всем представлениям Visual Studio, для элементов Solution Explorer поддерживается контекстное меню. Следовательно, если щелкнуть правой кнопкой мыши на элементе представления Visual Studio (в данном случае имени файла), то появится контекстное меню, специфическое именно для этого элемента. Например, если необходимо удалить файл из проекта, то достаточно открыть Solution Explorer, найти файл, щелкнуть на его имени правой кнопкой мыши и в появившемся контекстном меню выбрать пункт Delete (Удалить). Теперь, когда в тексте встретится фраза “выберите в контекстном меню пункт”, читатель будет знать, что это означает.



Обратите внимание, удаление файла из проекта лишь выводит его из состава проекта, но не удаляет физически.

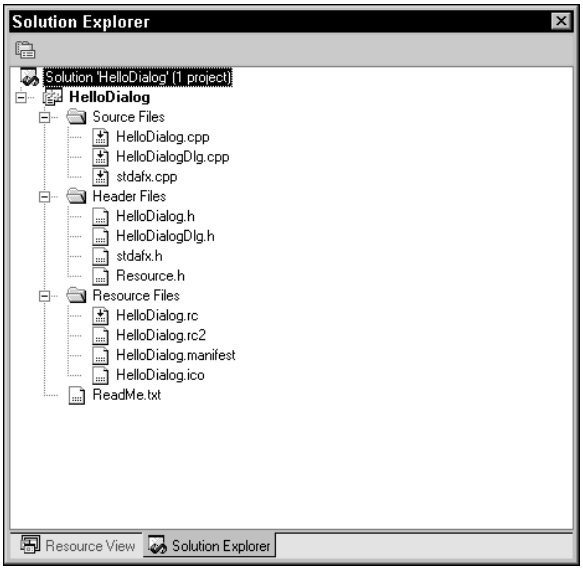


Рис. 1.9. Представление Solution Explorer (показанное здесь в отдельном окне) позволяет быстро находить, добавлять и удалять файлы из проектов текущего решения

- Class View (Классы). Еще одним встроенным представлением Visual Studio является Class View (рис. 1.10). Фактически, это представление используется чаще всех остальных, поскольку обеспечивает не только достаточно простые средства быстрого поиска классов и их членов, но и определенные манипуляции с ними. Если вкладка Class View не доступна внизу окна, добавьте ее, выбрав в меню View (Вид) среды разработки Visual Studio пункт Class View (Классы).

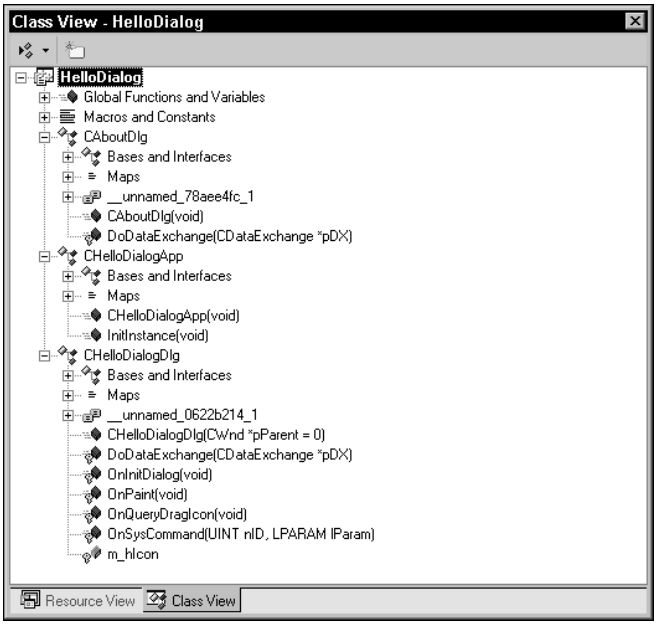


Рис. 1.10. Представление Class View (также представленное в отдельном окне) позволяет быстро находить и манипулировать классами (и их членами) в составе проекта текущего решения

Добавление обработчиков событий кнопки

Теперь заставим это демонстрационное приложение что-нибудь делать. Например, отображать при щелчке на кнопке окно, содержащее простое текстовое сообщение. В процессе реализации этой задачи рассмотрим операции, которые придется неоднократно выполнять в процессе изучения и применения Visual C++.

- Добавим обработчик события, реагирующий на действия пользователя.
- Организуем отображение сообщения.
- Откомпилируем и запустим на выполнение приложение Visual Studio.

Вначале предпримем следующие действия:

1. Откройте представление Resource View (в случае необходимости используйте меню View), как показано на рис. 1.8.
2. Раскройте папку Dialog (Диалоговые окна) и найдите элемент `IDD_HELLODIALOG_DIALOG`. Это *идентификатор ресурса* (resource ID), присвоенный диалоговому окну данного приложения. Дважды щелкните на имени `IDD_HELLODIALOG_DIALOG`, чтобы открыть этот ресурс.

3. Среда Visual Studio откроет редактор диалоговых окон (окно Dialog Editor), а панель инструментов (панель Toolbox) изменит таким образом, чтобы она содержала элементы управления, предназначенные для размещения в диалоговых ресурсах. Если окно Resource View занимает слишком много места на экране, закройте его.

   Как можно заметить на рис. 1.11, автоматически созданное диалоговое окно содержит лишь кнопки OK и Cancel, а также статический текст, оповещающий о том, что именно здесь и нужно размещать элементы управления диалогового окна.

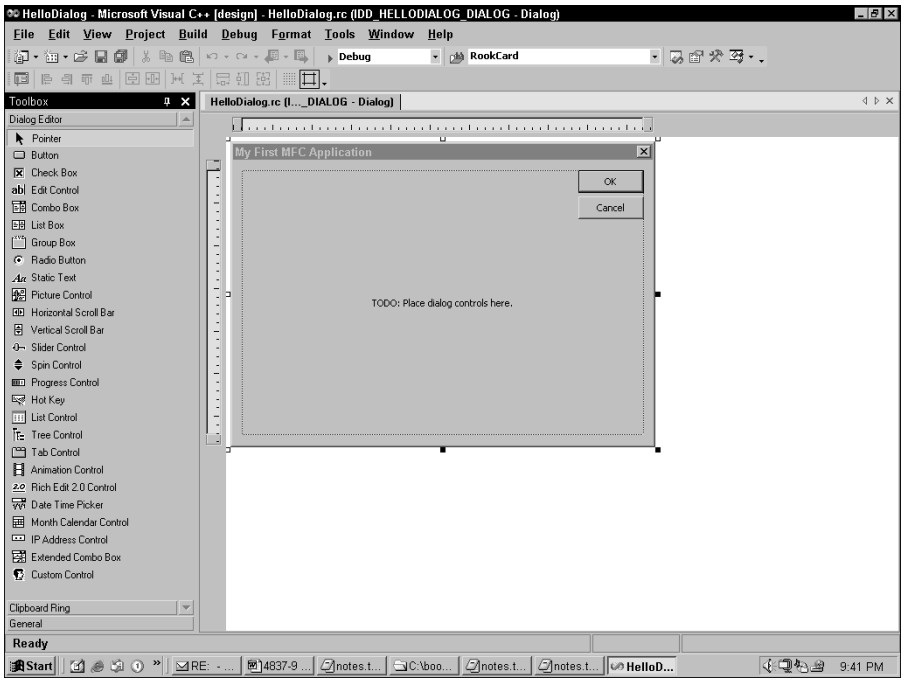


Рис. 1.11. Простое, стандартное диалоговое окно, созданное мастером Visual Studio по умолчанию

4. Добавьте для одной из кнопок обработчик события. Чтобы при щелчке на кнопке (например OK) на экране появилось сообщение, необходимо выполнить определенный код. Именно этот код и называется *обработчиком события* (event handler).

   Существует два способа назначить элементу управления обработчик события. Можно просто дважды щелкнуть на кнопке, событие `Click` (Щелчок) которой необходимо обрабатывать. В результате автоматически создается функция-член класса диалогового окна, вызываемая каждый раз при щелчке на этой кнопке. Но что если необходимо обрабатывать другие события, например, щелчок правой кнопкой мыши?

   В этом случае необходим мастер Event Handler Wizard (мастер обработчиков событий), который помогает добавлять такие обработчики событий элементов управления диалоговых окон, как *щелчок правой кнопкой мыши* (right-click) и *двойной щелчок* (double-click). Мастер Event Handler Wizard можно вызвать, щелкнув правой кнопкой мыши на элементе управления и выбрав затем в появившемся контекстном меню пункт Add Event Wizard (Мастер добавления событий).

Щелкните правой кнопкой мыши на кнопке OK и в появившемся контекстном меню выберите пункт Add Event Wizard.

5. Выберите в поле со списком Message type (Тип сообщения) элемент `BN_CLICKED` и убедитесь, что в поле со списком Class (Класс) выбран класс `CHelloDialogDlg`.

   Мастер Event Wizard позволяет выбирать как тип обрабатываемого сообщения, так и имя функции, отвечающей за обработку. При желании, разработчик может изменить автоматически созданные мастером имена функций-обработчиков, но обычно их оставляют установленными по умолчанию.

6. Щелкните на кнопке Add and Edit (Добавить и отредактировать). Откроется редактор исходного кода, который самостоятельно перейдет к функции `CHelloDialogDlg::OnBnClickedOk` (подразумевается, что имя функции осталось принятым по умолчанию). Измените эту функцию так, чтобы она выглядела следующим образом:

```
void CHelloDialogDlg::OnBnClickedOk()
{
   AfxMessageBox("You clicked the Ok button");
   // OnOK();
}
```

   Обратите внимание, обращение к функции `OnOK` базового класса закомментировано, т.к. стандартное поведение кнопки OK подразумевает, что после щелчка на ней диалоговое окно необходимо закрыть (при помощи функции `CDialog::EndDialog`). Поскольку данное диалоговое окно является главным окном, это приведет к завершению работы приложения.

7. Создайте приложение и запустите его, что можно сделать двумя способами:
   - нажать кнопку <F5>, чтобы сразу запустить приложение. Будет задан вопрос: “Следует ли создавать приложение?” Щелкните на кнопке Yes (Да), и компилятор Visual Studio, создав приложение (если проект не содержит ошибок), сразу запустит его;
   - альтернативный подход состоит из двух этапов: сначала в меню Build (Построить) выберите пункт Build Solution (Построить решение), а затем нажмите кнопку <F5>.

8. Проверьте приложение. После щелчка на кнопке OK должно появиться окно сообщения, аналогичное представленному на рис. 1.12.

Рис. 1.12. Демонстрационное приложение в действии

На настоящий момент рассмотрено создание диалогового приложения, а также связывание события и функции, отображающей окно сообщения. Далее будет рассматриваться использование редактора диалоговых окон для добавления элементов управления в диалоговое окно.

Использование редактора диалоговых окон и панели инструментов

Продолжим создание простого демонстрационного приложения, добавив в диалоговое окно пару элементов управления. При этом рассмотрим применение редактора диалоговых окон. Прежде чем предпринять следующие действия, закройте демонстрационное приложение (если оно до сих пор запущено).

1. Раскройте ресурс шаблона диалогового окна (ресурс `IDD_HELLODIALOG_DIALOG`), если он еще не открыт, и отобразите панель инструментов (нажав комбинацию клавиш <Ctrl+Alt+X>).
2. Удалите из диалогового окна статический текст с глупой надписью "`TODO: ...`" (Что делать: ...). Для этого достаточно щелкнуть на элементе управления (на любом из его слов), а затем нажать клавишу <Delete>.
3. Измените размеры диалогового окна. Щелкните мышью на поверхности диалогового окна. По сторонам изображения появятся небольшие темные квадраты, называемые *маркерами размера* (sizing gripper) (рис. 1.13). Щелкните мышью на нижнем правом маркере и, перемещая его, измените размеры диалогового окна — сделайте его поменьше, стандартный размер слишком велик для этого приложения.

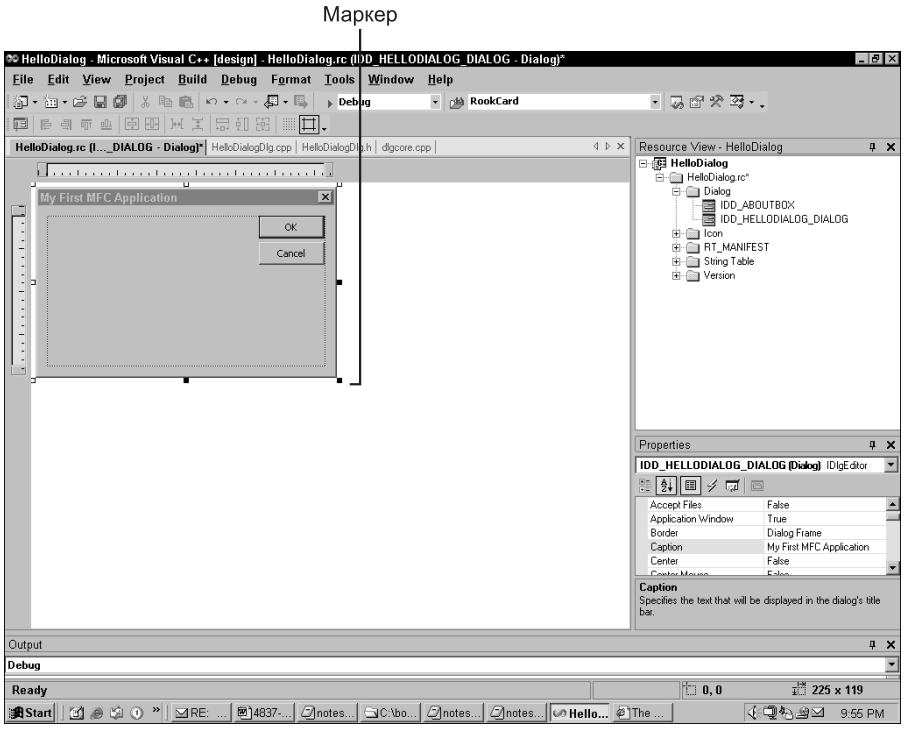


Рис. 1.13. Маркеры позволяют легко изменять размеры диалоговых окон и элементов управления

4. Используя панель инструментов, добавьте в диалоговое окно *элемент управления статический текст* (static text control) и *элемент управления поле ввода* (edit control). Элемент управления можно добавить в диалоговое окно, щелкнув вначале на его пиктограмме в панели инструментов, а затем на диалоговом окне приблизительно в том месте, где его необходимо разместить. Элемент управления почти никогда не попадает туда, куда нужно, поэтому его необходимо переместить затем в надлежащее место диалогового окна. Для более точного размещения элементов управления можно воспользоваться кнопками со стрелками на клавиатуре.
5. Измените текст элемента управления статический текст на "`Message text:`" (Текст сообщения:). Для этого достаточно щелкнуть на элементе управления и набрать текст. Подобное поведение связано с тем, что по умолчанию для элемента управления статический текст заданно свойство `Caption` (Подпись). Чтобы просмотреть все свойства элемента управления, достаточно щелкнуть на его изображении и нажать клавишу <F4>. Появится окно Properties (Свойства), предоставляющее доступ ко всем свойствам элемента управления (рис. 1.14).

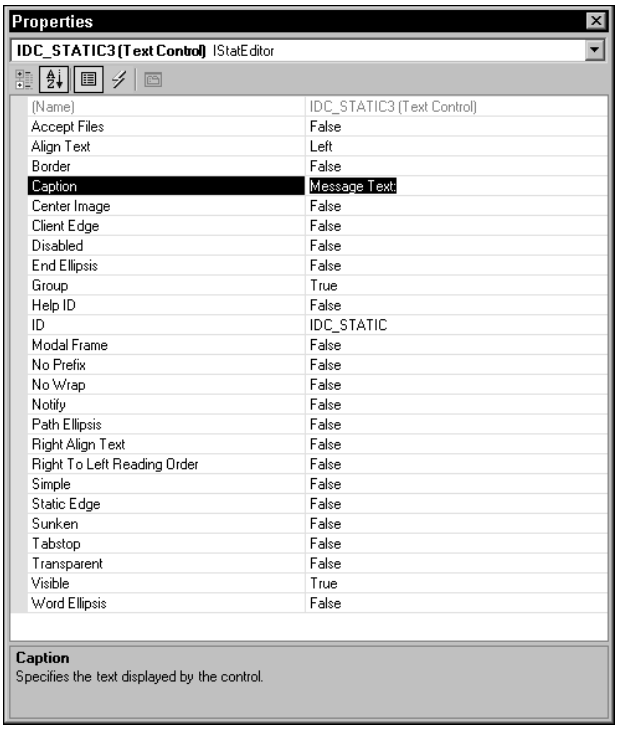


Рис. 1.14. Окно Properties *(представленное здесь отдельно) содержит в верхней части раскрывающийся список, позволяющий выбрать любой элемент управления диалогового окна, список свойств которого необходимо отобразить*

Прежде чем продолжать изучение среды разработки, ознакомимся с панелью инструментов редактора диалоговых окон, чтобы не испытывать неудобств, когда дело дойдет до изменения диалоговых окон. Эта панель инструментов (рис. 1.15) применяется для проверки диалогового окна, установки последовательности переходов (при нажатии кнопки <Tab>), а также для установки размеров элементов управления и их выравнивания относительно друг друга.

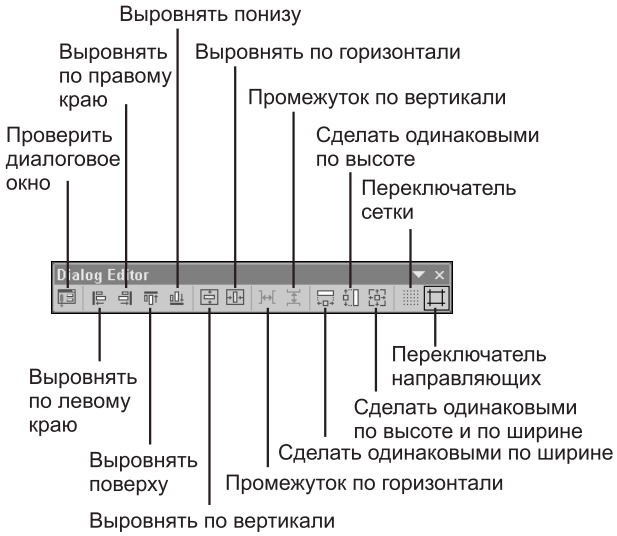


Рис. 1.15. Панель инструментов редактора диалоговых окон позволяет выполнять множество разнообразных задач, включая выравнивание элементов управления, изменение их размеров, позиционирование, последовательность перехода и проверку работоспособности диалогового окна

Теперь окно имеет поле ввода, в которое пользователь может ввести текст. Но как им воспользоваться? Ответ прост — с помощью динамического обмена данными.

Использование динамического обмена данными

Динамический обмен данными (DDX — Dynamic Data Exchange) — это средство, при помощи которого можно легко передавать данные между элементами управления диалогового окна и переменными-членами приложения. Чтобы создать переменную DDX, достаточно щелкнуть[1] на элементе управления в ресурсах шаблона диалогового окна и в появившемся контекстном меню выбрать пункт Add Variable (Добавить переменную). Для элемента управления Edit Control (Поле ввода) это запустит мастер Add Member Variable Wizard (Мастер добавления переменной-члена), изображенный на рис. 1.16.

Можно создать два типа переменных DDX.

- *Управляющая переменная DDX* (DDX Control Variable). Управляющая переменная связывает элемент управления диалогового окна с переменной, являющейся экземпляром класса-оболочки данного элемента управления в библиотеке MFC. Примером такого подхода является класс `CEdit`. Наличие переменной-члена типа `CEdit`, привязанной к полю ввода, позволяет манипулировать этим полем с помощью функций-членов класса `CEdit`, избежав необходимости использовать более трудоемкий способ создания сообщений Windows. Классы элементов управления рассматриваются в главе 10 “Элементы управления”. Обычно этот тип переменной DDX используется в том случае, когда необходимо тем или иным образом манипулировать самим элементом управления, например, перемещать или изменять его размер.

[1] *Вероятно, правой кнопкой мыши. — **Прим. ред.***

- *Переменная значения DDX* (DDX Value Variable). Переменная значения связывает элемент управления диалогового окна с переменной MFC, содержащей значение этого элемента управления. Например, если имеется поле ввода, содержащее число, то можно создать переменную значения DDX типа `CInt`, которая будет содержать это значение.

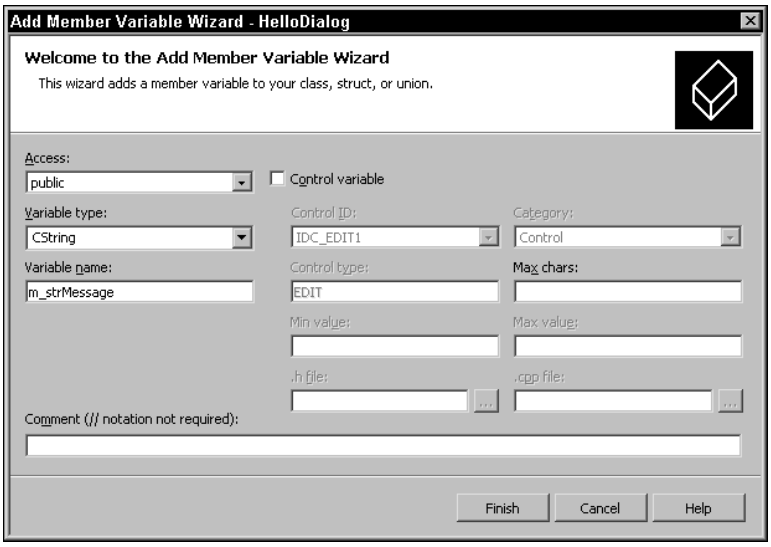


Рис. 1.16. Мастер Add Member Variable Wizard позволяет создавать как управляющие переменные DDX, так и переменные значения DDX

Поскольку речь о манипулировании самим элементом управления пока не идет, сбросим флажок Control Variable (Управляющая переменная). Затем в раскрывающемся списке Variable type (Тип переменной) выберите тип `CString`. (В той версии Visual Studio, с которой работал автор, типа `CString` не было в списке, поэтому его приходилось вводить вручную. Если в более новой версии этот тип не появился, то сделайте то же самое.) Теперь введем имя переменной-члена, по которому можно обратиться к значению элемента управления. В данном случае было выбрано имя **`m_strMessageText`** (см. рис. 1.16). Еще раз проверив правильность введенной информации, щелкните на кнопке Finish (Готово).

Теперь, когда переменная значения DDX создана и подключена к полю ввода, осталось организовать передачу ее значения во время выполнения. Для этого применяется функция `CDialog::UpdateData`. Чтобы использовать эту функцию, достаточно вызвать ее с логическим значением `True` в качестве аргумента, если необходимо передать данные из элементов управления диалогового окна в переменные, или значением `False`, если необходимо передать данные из переменных в элементы управления диалогового окна. По умолчанию принято значение `True`. Следовательно, в функцию `OnBnClickedOk` диалогового окна необходимо внести изменения. Если окно редактора кода уже закрыто, то чтобы открыть его, достаточно выбрать в панели Class View класс `CHelloDialogDlg` и дважды щелкнуть на имени функции `OnBnClickedOk`. После внесения изменений функция должна выглядеть следующим образом:

```
void CHelloDialogDlg::OnBnClickedOk()
{
  UpdateData(TRUE);
```

```
  AfxMessageBox(m_strMessageText);
  // OnOK();
}
```

Обратите внимание на обращение к функции `UpdateData`. Сейчас она передает данные из диалогового окна в переменные DDX. Это означает, что переменная `m_strMessageText` получит значение, введенное пользователем в поле ввода диалогового окна, которое впоследствии будет отображено при обращении к функции `AfxMessageBox`. Откомпилируйте и запустите приложение. Результат должен выглядеть подобно представленному на рис. 1.17.

Рис. 1.17. DDX позволяет быстро создать код, связывающий элементы управления с переменными-членами

Исправление ошибок компиляции

До сих пор рассматривалось выполнение лишь тех программ Visual Studio, которые не содержали ошибок. Но что если в код вкралась ошибка и его нельзя откомпилировать? Давайте рассмотрим эту ситуацию, введя в код небольшую синтаксическую ошибку.

Еще раз модифицируем функцию `OnBnClickedOk`, но на сей раз внесем явную ошибку — опечатку в имени функции `AfxMessageBox`. Теперь функция должна выглядеть следующим образом:

```
void CHelloDialogDlg::OnBnClickedOk()
{
  UpdateData(TRUE);
  xAfxMessageBox(m_strMessageText);
  // OnOK();
}
```

Попытаемся откомпилировать приложение теперь. Когда компилятор (или компоновщик) сталкивается с ошибками, он отображает сообщения о них в списке окна Task List (Список задач), как показано на рис. 1.18.

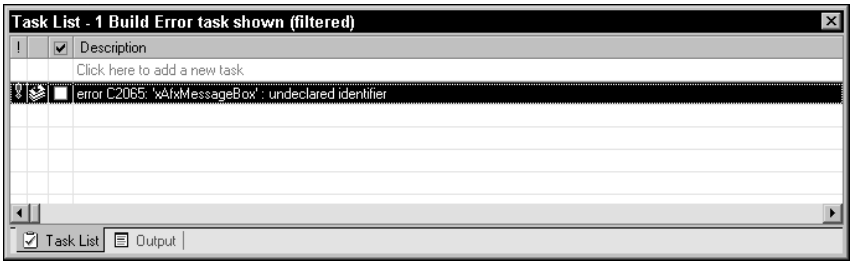


Рис. 1.18. Список окна Task List позволяет просматривать сообщения об ошибках периода компиляции, а также легко переходить к участку кода, являющемуся их причиной

Окно Task List (Список задач) содержит таблицу, столбцы которой содержат описание ошибок, имена файлов, в которых они находятся, и номера строк, содержащих ошибки. В данном случае отображено следующее сообщение об ошибке: "`error C2065: 'xAfxMessageBox' : undeclared identifier'`" (ошибка номер `C2065`: `xAfxMessageBox` — необъявленный идентификатор).

Причина данной конкретной ошибки очевидна; но что, если сообщение об ошибке не столь однозначно? Чтобы получить более подробное описание ошибки, достаточно просто нажать клавишу <F1>, это приведет к отображению текста из файла помощи, ассоциированного с номером ошибки (в данном случае `C2065`).

Для устранения проблемы достаточно дважды щелкнуть на строке, содержащей запись об ошибке, и Visual Studio автоматически откроет необходимый файл (если он еще не открыт) и переместит курсор на строку кода, содержащую ошибку. Теперь можно отредактировать строку, заменив имя функции `xAfxMessageBox` на `AfxMessageBox`, и пытаться снова откомпилировать проект.

Отладка в Visual Studio

Следующей подлежащей рассмотрению темой является отладка простого приложения Visual Studio. Преднамеренно внесем в исходный код функции `OnBnClickedOk` ошибку, изменив аргумент функции `UpdateData` с `True` на `False`. Эта строка кода выделена полужирном шрифтом:

```
void CHelloDialogDlg::OnBnClickedOk()
{
  UpdateData(FALSE);
  AfxMessageBox(m_strMessageText);
  // OnOK();
}
```

Полагаю, читатель помнит, к чему приведет такое изменение: теперь DDX вместо передачи данных из элемента управления в переменную `m_strMessageText` будет передавать содержимое переменной `m_strMessageText` в элемент управления. В результате окно сообщения не сможет правильно отображать информацию. Внесите это изменение, откомпилируйте и запустите приложение, чтобы убедиться в сказанном выше. Обратите внимание, теперь окно сообщения не только содержит пустую строку (которой переменная `m_strMessageText` инициализируется в конструкторе диалогового окна), но и текст, внесенный пользователем в элемент управления, также заменяется пустой строкой после каждого щелчка на кнопке OK — это тоже результат вызова функции `UpdateData(FALSE)`. Итак, причина ошибки известна, но как исправить ее?

Установка контрольных точек

Когда содержащая ошибку строка кода известна, необходимо остановить выполнение приложения и выяснить, что именно там происходит. Для этого следует найти функцию `CHelloDialogDlg::OnBnClickedOk` и установить *контрольную точку* (breakpoint) на обращении к функции `UpdateData`. Чтобы установить контрольную точку, достаточно щелкнуть мышью на поле, расположенном слева от текста программы (или выбрать в меню Debug (Отладка) пункт New Breakpoint (Новая контрольная точка), или, щелкнув правой кнопкой мыши на необходимой строке кода, выбрать из появившегося контекстного меню пункт New Breakpoint (Новая контрольная точка)). Автор полагает, что устанавливать

контрольную точку щелчком на левом поле быстрее и проще, поскольку в отличие от других способов это позволяет избежать вызова диалогового окна New Breakpoint. (Чтобы установить контрольную точку в текущей строке можно также нажать кнопку <F9>.)

Как только контрольная точка установлена, в левом поле строки кода появится красная точка. На рис. 1.19 изображено поле, на котором необходимо щелкнуть, а также внешний вид установленной контрольной точки.

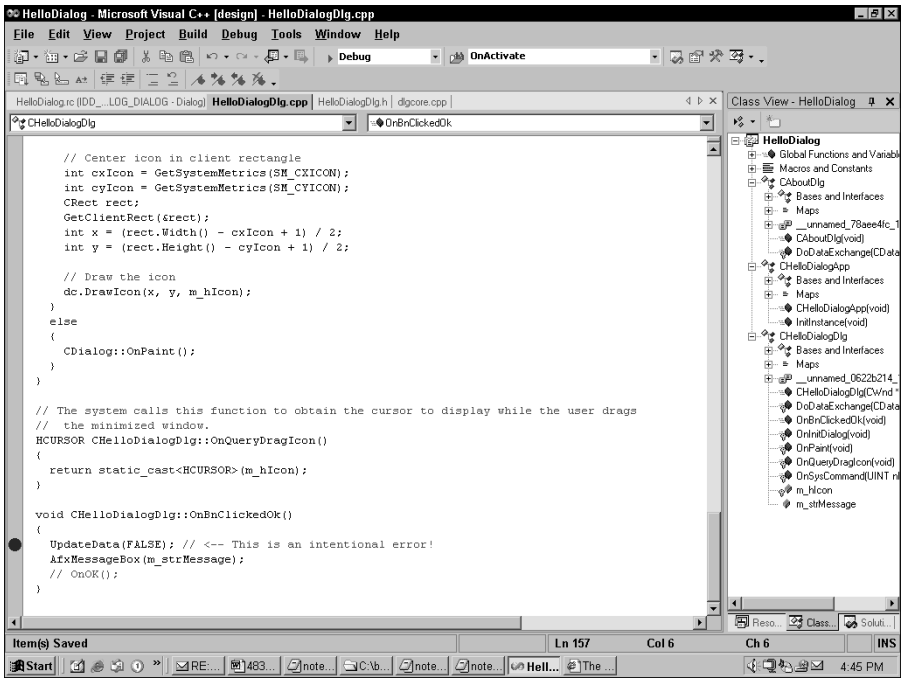


Рис. 1.19. С помощью контрольной точки можно легко установить место, где выполнение приложения должно остановиться

Откомпилируйте и запустите приложение, нажав кнопку <F5>. На сей раз, когда пользователь введет текст и щелкнет на кнопке OK, выполнение кода остановится на строке, содержащей вызов функции `UpdateData`. (Текущую строку выполняемого кода отмечает желтая стрелка.)

Чтобы проконтролировать значение переменной `m_strMessageText`, достаточно установить курсор мыши на ее имени в текущей строке выполняемого кода. Среда разработки Visual Studio автоматически отобразит адрес в памяти, занимаемый этой переменной, а также ее текущее значение. В настоящий момент переменная `m_strMessageText` пуста.

Для перехода к следующей строке кода достаточно нажать кнопку <F10>. Текущей строкой выполняемого кода станет обращение к функции `AfxMessageBox`. Теперь можно снова просмотреть значение переменной `m_strMessageText` и выяснить, что вызов функции `UpdateData` по неизвестным причинам не модифицировал переменную! Следовательно, во всем виновата функция `UpdateData`. Безусловно, причина неполадки была известна заранее, но рассматриваемая ситуация мало отличается от реальных условий отладки. Итак, настало время переходить к исходному коду MFC.

Исходный код MFC

Автор не забыл своего обещания сделать эту главу по возможности проще и не собирался настолько подробно останавливаться на этой теме в самом начале книги. Но данный раздел важен не только потому, что всем разработчикам MFC рано или поздно придется столкнуться с отладкой кода MFC, но и потому, что это весьма полезно для понимания внутренних процессов работы приложений и окажется весьма не лишним, когда дело дойдет до сакраментальной фразы "Я понятия не имею, почему это не работает".

Чтобы вернуться к контрольной точке (и продолжить выполнение кода), достаточно нажать клавишу <F5>. Далее наберите в поле ввода какой-либо текст и щелкните на кнопке OK. Но прежде чем сделать это, ознакомьтесь с интересной информацией. Версия компилятора Visual C++ 6 обладала одной уникальной способностью, которой ранее (среди компиляторов *Microsoft*) был наделен лишь компилятор Visual Basic — возможностью устанавливать следующий выполняемый оператор.

Таким образом, несмотря на то, что текущая выполняемая строка (обращение к функции `AfxMessageBox`) расположена ниже той, к которой необходимо перейти, вовсе не обязательно запускать программу с начала, достаточно указать компилятору Visual Studio, что следующей выполняемой строкой кода должно быть обращение к функции `UpdateData`. Для этого необходимо вызвать контекстное меню данной строки кода и выбрать в нем пункт Set Next Statement (Установить следующим оператором). Желтая стрелка появится напротив обращения к функции `UpdateData`.

Ну как, неплохо "перешли к" функции? Кроме того, переход можно осуществить при помощи клавиши <F11> (или меню Debug (Отладка), пункт Step Into (Перейти к)). Как только это будет сделано в первый раз, Visual Studio загрузит в редактор файл исходного кода функции `UpdateData` (`wincore.cpp`). При этом подразумевается, что при установке Visual C++ исходный код классов MFC также был установлен. (Данный параметр при установке задан по умолчанию.)

Исходный код функции `CWnd::UpdateData` выглядит следующим образом (некоторые строки кода пришлось перенести, чтобы их размер соответствовал размеру страниц книги, а комментарии переведены на русский язык):

```
BOOL CWnd::UpdateData(BOOL bSaveAndValidate)
{
 // Вызов UpdateData ранее Modal?
 ASSERT(::IsWindow(m_hWnd));

 CDataExchange dx(this, bSaveAndValidate);

 // Предотвратить передачу уведомлений элемента управления
 // в течение выполнения UpdateData.
 _AFX_THREAD_STATE* pThreadState = AfxGetThreadState();
 HWND hWndOldLockout = pThreadState->m_hLockoutNotifyWindow;
 ASSERT(hWndOldLockout != m_hWnd);   // избежать рекурсии
 pThreadState->m_hLockoutNotifyWindow = m_hWnd;

 BOOL bOK = FALSE;       // возможен отказ
 TRY
 {
  DoDataExchange(&dx);
  bOK = TRUE;          // сработало
 }
 CATCH(CUserException, e)
 {
```

```
  // Проверка не пройдена - пользователь уже оповещен
  ASSERT(!bOK);
  // Замечание: вызов DELETE_EXCEPTION_(e) необязателен
 }
 AND_CATCH_ALL(e)
 {
  // Проверка не пройдена в связи с OOM или другим отказом ресурса
  e->ReportError(MB_ICONEXCLAMATION,
                 AFX_IDP_INTERNAL_FAILURE);
  ASSERT(!bOK);
  DELETE_EXCEPTION(e);
 }
 END_CATCH_ALL

 pThreadState->m_hLockoutNotifyWindow = hWndOldLockout;
 return bOK;
}
```

На первый взгляд здесь ничто не указывает на способ передачи данных (от элемента управления к переменной и наоборот). Имеется лишь две подозрительные функции: конструктор `CDataExchange` и обращение к функции `DoDataExchange`. (Обе они выделены полужирным шрифтом.)

Теперь можно установить на обеих строках кода контрольные точки и нажать клавишу <F5>, чтобы получить возможность прочитать их значения (в данном конкретном случае нельзя воспользоваться пунктом Set Next Statement, поскольку необходимые локальные переменные не будут инициализированы). Вместо клавиши <F5> можно воспользоваться клавишей <F10> (это приведет к пошаговому выполнению строк кода) и постепенно перейти к необходимой строке.

Вне зависимости от способа перехода к строке конструктора `CDataExchange`, нажмите клавишу <F11>, чтобы войти внутрь этой процедуры.

Конструктор `CDataExchange` находится в том же файле и выглядит следующим образом:

```
CDataExchange::CDataExchange(CWnd* pDlgWnd, BOOL bSaveAndValidate)
{
  ASSERT_VALID(pDlgWnd);
  m_bSaveAndValidate = bSaveAndValidate;
  m_pDlgWnd = pDlgWnd;
  m_idLastControl = 0;
}
```

Итак, это — тупик. Здесь нет ничего кроме присвоения значений нескольких переменных. Следовательно, данный блок кода необходимо быстро покинуть, сохранив при этом порядок выполнения кода. Как это сделать? Нажмите комбинацию клавиш <Shift+F11>. По окончании выполнения текущей функции управление перейдет к строке кода, расположенной непосредственно за обращением к этой функции, и выполнение кода снова остановится. Теперь доберемся до обращения к функции `DoDataExchange` и, как и в прошлый раз, осуществим вход в функцию (<F11>).

```
void CHelloDialogDlg::DoDataExchange(CDataExchange* pDX)
{
  DDX_Text(pDX, IDC_EDIT1, m_strMessageText);
  CDialog::DoDataExchange(pDX);
}
```

Мм-да. Этот код автоматически создан мастером специально для данного приложения (обратите внимание на имя класса). Кстати, здесь кроме всего прочего впервые упоминается пользовательская переменная `m_strMessageText`. Осуществив вход в функцию `DDX_Text`, можно увидеть следующий код (здесь строки тоже перенесены):

```
void AFXAPI DDX_Text(CDataExchange* pDX,
                     int nIDC,
                     CString& value)
{
 HWND hWndCtrl = pDX->PrepareEditCtrl(nIDC);
 if (pDX->m_bSaveAndValidate)
 {
  int nLen = ::GetWindowTextLength(hWndCtrl);
  ::GetWindowText(hWndCtrl,
                  value.GetBufferSetLength(nLen),
                  nLen+1);
  value.ReleaseBuffer();
 }
 else
 {
  AfxSetWindowText(hWndCtrl, value);
 }
}
```

Выглядит как дремучий лес! Но можно заметить, что код проверяет значение `pDX->m_bSaveAndValidate`, и если оно равно `TRUE`, то применяется функция `::GetWindowText` (получающая значение из окна и передающая его в переменную). Это именно то, что нужно. Но если, используя клавишу <F10>, пройти по коду, то можно заметить, что значением `pDX->m_bSaveAndValidate` является `FALSE`, а следовательно вызвана будет другая функция — `AfxSetWindowText`. Вот почему в элемент управления передается значение переменной `m_strMessageText`.

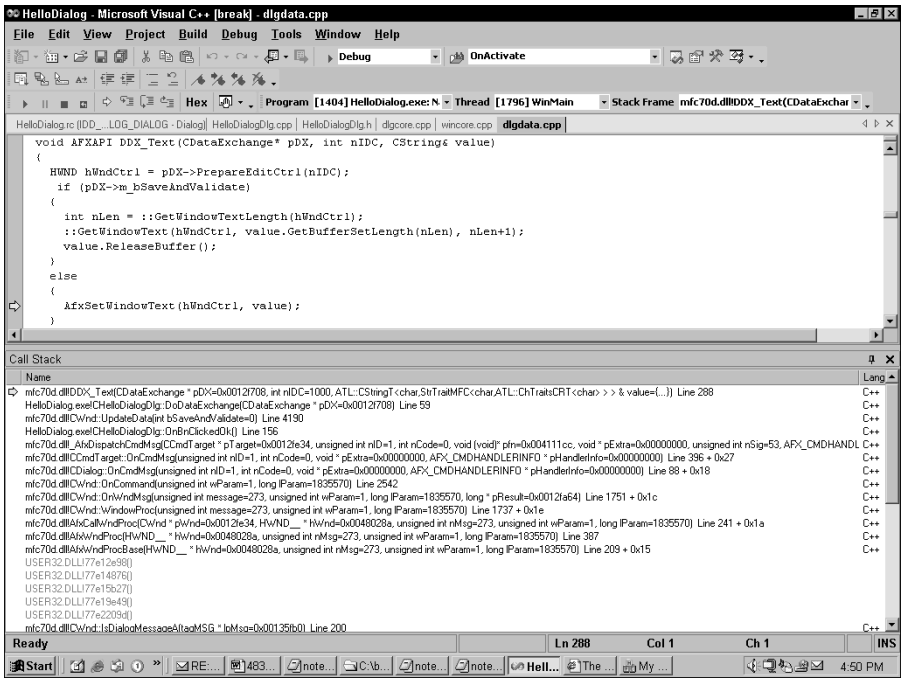


Рис. 1.20. По стеку вызовов можно легко перемещаться вверх и вниз, находя информацию, необходимую для отладки кода

Таким образом, осталось выяснить, почему переменная pDX->m_bSaveAndValidate содержит значение FALSE, и проблема будет решена. Предположим, что необходимо просмотреть предыдущий фрагмент кода, тот, который обсуждался несколько минут назад, но повторно запускать для этого весь процесс прокрутки кода еще раз не хочется. Для этого достаточно открыть окно Call Stack (меню Debug (Отладка) пункты Windows (Окно), Call Stack (Стек вызовов)). Окно содержит стек (список) функций, выполняющихся в настоящий момент (рис. 1.20).

Стек вызовов отображает информацию не только о текущей функции, но и обо всех предыдущих функциях, расположенных в текущем стеке выполнения. Если щелкнуть в этом окне на третьей функции сверху (строка начинается с надписи mfc70d.dll!CWnd::UpdateData), то можно перейти к следующей строке кода:

```
CDataExchange dx(this, bSaveAndValidate);
```

Исходя из этого, можно предположить, что значение bSaveAndValidate (переданное из функции CHelloDialogDlg::OnBnClickedOk) — это именно то значение, которое используется для передачи в переменную pDX->m_bSaveAndValidate. Следовательно, чтобы приложение снова заработало правильно, необходимо просто модифицировать данное обращение и передать в качестве аргумента значение TRUE вместо значения FALSE.

Вполне очевидно, что в реальном мире профессиональной разработки программного обеспечения задачи отладки могут быть намного сложнее. Но независимо от того, насколько сложна проблема, которую придется решать в ходе отладки, основы знаний об установке контрольных точек (как минимум), проверки значений переменных, пошагового выполнения, перемещения по коду (включая код библиотеки MFC), установки следующего оператора, отображения стека вызовов и перехода к функциям по стеку вызовов окажутся весьма полезны.

Применение обработчиков сообщений

Как известно, операционная система Windows использует модель, *управляемую событиями* (event-driven). Это означает, что вместо последовательного набора команд приложение содержит *цикл сообщений* (message loop), который обрабатывает сообщения (или события), передаваемые приложению операционной системой Windows. Как только сообщение получено, цикл сообщений берет на себя всю ответственность за его обработку (обычно для этого применяется *обработчик сообщения* (message handler) или функция, специально написанная для обработки конкретного сообщения). Цикл сообщений выполняется на протяжении всего периода работы приложения. Выполнение цикла завершается при поступлении сообщения WM_QUIT. Но где же этот цикл в демонстрационном приложении? Если "копать" библиотеку MFC достаточно глубоко, то он обнаружится в функции CWnd::RunModalLoop, расположенной в файле thrdcore.cpp. Этот цикл выглядит следующим образом:

```
int CWnd::RunModalLoop(DWORD dwFlags)
{
 ASSERT(::IsWindow(m_hWnd)); // окно должно быть создано
 ASSERT(!(m_nFlags & WF_MODALLOOP));

 // для отслеживания состояния ожидания
 BOOL bIdle = TRUE;
 LONG lIdleCount = 0;
 BOOL bShowIdle = (dwFlags & MLF_SHOWONIDLE)
  && !(GetStyle() & WS_VISIBLE);
 HWND hWndParent = ::GetParent(m_hWnd);
 m_nFlags |= (WF_MODALLOOP|WF_CONTINUEMODAL);
 MSG *pMsg = AfxGetCurrentMessage();
```

```
// получать и пересылать сообщения, пока состояние остается модальным
for (;;)
{
 ASSERT(ContinueModal());

 // фаза1: выяснить, нельзя ли выполнить эту работу в режиме ожидания
 while (bIdle &&
 !::PeekMessage(pMsg, NULL, NULL, NULL, PM_NOREMOVE))
 {
  ASSERT(ContinueModal());

  // отобразить диалоговое окно, если очередь сообщений
  // не пуста и обнаружен режим ожидания
  if (bShowIdle)
  {
   ShowWindow(SW_SHOWNORMAL);
   UpdateWindow();
   bShowIdle = FALSE;
  }

  // пока установлено состояние bIdle, обращаться к OnIdle
  if (!(dwFlags & MLF_NOIDLEMSG)
  && hWndParent != NULL && lIdleCount == 0)
  {
   // переслать родителю сообщение WM_ENTERIDLE
   ::SendMessage(hWndParent,
                 WM_ENTERIDLE,
                 MSGF_DIALOGBOX,
                 (LPARAM)m_hWnd);
  }
  if ((dwFlags & MLF_NOKICKIDLE) ||
   !SendMessage(WM_KICKIDLE, MSGF_DIALOGBOX, lIdleCount++))
  {
   // остановить обработку в режиме ожидания до следующего раза
   bIdle = FALSE;
  }
 }

 // фаза2: передавать сообщения, пока это возможно
 do
 {
  ASSERT(ContinueModal());

  // передать сообщение и выйти (WM_QUIT)
  if (!AfxPumpMessage())
  {
   AfxPostQuitMessage(0);
   return -1;
  }

  // отобразить окно, если получено определенное сообщение
  if (bShowIdle &&
  (pMsg->message == 0x118
  || pMsg->message == WM_SYSKEYDOWN))
  {
   ShowWindow(SW_SHOWNORMAL);
   UpdateWindow();
   bShowIdle = FALSE;
  }
```

```
    if (!ContinueModal())
     goto ExitModal;

    // передав сообщение "нормальный",
    // сбросить состояние "не ожидание"
    if (AfxIsIdleMessage(pMsg))
    {
     bIdle = TRUE;
     lIdleCount = 0;
    }

   } while (::PeekMessage(pMsg, NULL,
                          NULL, NULL,
                          PM_NOREMOVE));
 }

 ExitModal:
  m_nFlags &= ~(WF_MODALLOOP|WF_CONTINUEMODAL);
  return m_nModalResult;
}
```

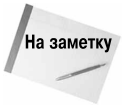
Выше приведен большой фрагмент сложного кода. Не стоит расстраиваться, если он пока не до конца понятен. На данном этапе рассмотрим лишь следующее основные моменты.

- За операторами инициализации следует бесконечный цикл `for (;;)`, предназначенный для обработки сообщений.
- Внутри этого цикла `for` расположен цикл `while`, который сначала определяет, нельзя ли выполнить эту работу в режиме *ожидания* (idle). Как известно, приложения MFC перехватывают сообщения постоянно (в фоновом режиме) даже тогда, когда казалось бы приложение не делает ничего. Таким образом, цикл `while` проверяет сначала состояние флага `bIdle` и выясняет, достаточно ли времени, чтобы выполнить эту часть работы в фоновом режиме. Для подтверждения того, что приложение не завершило работу, осуществляется вызов функции `PeekMessage`, которая возвращает значение `0`, если в очереди нет никаких сообщений. Следовательно, если флаг `bIdle` установлен в состояние `1` и в очереди есть сообщения, то выполняется код, посылающий окну сообщение о том, что оно способно выполнить обработку в фоновом режиме.
- Следующий цикл (`do`) фактически обрабатывает сообщения. Это и есть *цикл сообщений* (message loop), который иногда называют *конвейером сообщений* (message pump), именно он и осуществляет в приложении фактическую обработку сообщений. Обратите внимание, после обращения к функции `ContinueModal` (позволяющей удостовериться в том, что модальное состояние должно быть сохранено) код проверяет наличие в очереди сообщения `WM_QUIT` (в этом случае функция `AfxPumpMessage` возвращает значение `0`). Если это так, то осуществляется выход из функции и завершение работы приложения. В противном случае цикл продолжает получать и пересылать сообщения.

Несмотря на то, что это достаточно поверхностное описание процесса обработки сообщений в приложении MFC, оно затрагивает такую фундаментальную концепцию MFC, как передача сообщений, обеспечивающую прием сообщений, посланных приложению из операционной системы Windows, и их распределение между обработчиками сообщений. Но как перехватить определенное сообщение? Реализуя обработчик события для кнопки OK, расположенной в диалоговом окне, разработчик фактически организует перехват соответствующего сообщения, хоть это и не вполне очевидно. Для этого в *карте сообщений* (message map) создается запись о том, что сообщения кнопки `BN_CLICKED` необходимо передавать на обработку пользовательской функции `OnBnClickedOk`. Чтобы просмотреть карту сообщений, открой-

те файл HelloDialogDlg.cpp и отыщите строку, начинающуюся словами BEGIN_MESSAGE_MAP. Там расположен следующий код:

```
BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
  ON_BN_CLICKED(IDOK, OnBnClickedOk)
END_MESSAGE_MAP()
```



Для успешной работы с библиотекой MFC достаточно знать, что существуют карты сообщений, связывающие сообщения и функции, а также разнообразные мастера, которые позволяют создавать эти функции. Более подробная информация о внутренней организации и функционировании таких макросов библиотеки MFC, как BEGIN_MESSAGE_MAP, приведена в книге *Скота Винго* (Scot Wingo) *MFC Internals*.

Теперь рассмотрим создание обработчика сообщения для события, не относящегося к *графическому интерфейсу пользователя* (GUI — Graphical User Interface). Создать обработчик сообщения BN_CLICKED для кнопки OK не так уж и сложно: достаточно дважды щелкнуть в редакторе диалогового окна на кнопке OK. Но что делать с другими сообщениями, для доступа к которым редактор диалогового окна не поможет? Например, если необходимо обрабатывать события перемещения или изменения размера диалогового окна? Ниже приведена последовательность действий, позволяющая легко и просто создать обработчик, который отобразит соответствующее сообщение, если пользователь дважды щелкнет мышью в клиентской области.

1. Откройте представление Class View (Классы).
2. Найдите класс, отвечающий за то диалоговое окно, обработчик сообщений которого необходимо создать. В данном случае это класс CHelloDialogDlg.
3. Щелкните правой кнопкой мыши на имени класса и в появившемся контекстном меню выберите пункт Properties (Свойства).
4. В диалоговом окне Properties (Свойства) щелкните на пиктограмме Messages (Сообщения) (расположенной в верхней части диалогового окна). Окно будет разделено на два столбца. Первый из них содержит список всех сообщений, для которых можно создать обработчики, а второй — перечень имен всех уже созданных обработчиков.
5. Щелкните в поле справа от идентификатора сообщения WM_LBUTTONDBLCLK. Поле идентификатора сообщения окажется выделенным, а в поле справа появится кнопка со стрелкой. Выбирая сообщения в правом столбце, обратите внимание на информационное поле в нижней части диалогового окна, содержащее описание сообщения текущего идентификатора. В данном случае (рис. 1.21) о сообщении с идентификатором WM_LBUTTONDBLCLK говорится (внизу окна), что оно “Означает двойной щелчок левой кнопки мыши.” Хотя информация о сообщении не всегда настолько очевидна, как в данном случае, она окажется весьма полезной при поиске идентификатора, точное имя которого неизвестно.
6. Щелчок на кнопке со стрелкой развернет список действий, которые можно выполнить с помощью обработчика этого события. Поскольку на настоящий момент никаких обработчиков для события не существует, единственным пунктом списка будет <Add> OnLButtonDblClk (<Добавить> OnLButtonDblClk).
7. Выберите в списке этот пункт, и среда разработки Visual Studio самостоятельно создаст как функцию-член по имени CHelloDialogDlg::OnLButtonDblClk, так и элемент карты сообщений, переадресующий сообщение WM_LBUTTONDBLCLK именно этой функции. Вот и все.

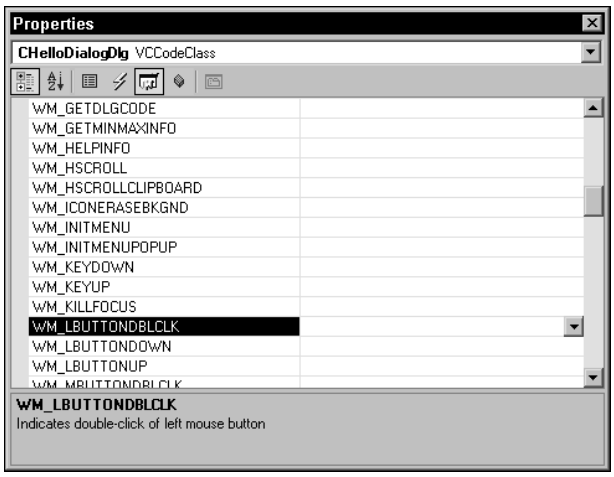


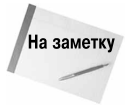
Рис. 1.21. Диалоговое окно Properties *позволяет легко добавлять новые обработчики сообщений*

8. Для проверки работоспособности обработчика измените функцию-член `OnLButtonDblClk` так, чтобы она выглядела следующим образом:

```
void CHelloDialogDlg::OnLButtonDblClk(UINT nFlags,
                                      CPoint point)
{
  AfxMessageBox("Ouch! Don't touch me there!");
  CDialog::OnLButtonDblClk(nFlags, point);
}
```

9. Откомпилировав и запустив это приложение, можно убедиться, что в ответ на двойной щелчок в клиентской области главного диалогового окна появляется соответствующее сообщение.

Безусловно, обработчик сообщения можно отредактировать в любой момент. Достаточно открыть содержащий его файл и внести изменения в функцию. Но как удалить обработчик? Один из способов заключается в удалении *прототипа функции* (function prototype) из файла заголовка, удалении *определения функции* (function definition) из файла реализации `.CPP` и удалении записи из карты сообщений. Однако такой подход сложен и труден. Проще всего удалить обработчик сообщения с помощью диалогового окна Properties (Свойства). Перейдя к полю сообщения `WM_LBUTTONDBLCLK` и открыв список возможных действий, можно заметить, что теперь он содержит два пункта — один для редактирования функции, а второй для ее удаления. Выбор пункта соответствующего удалению, приведет к полному автоматическому удалению обработчика события из всех вышеперечисленных мест.



Удаляя обработчики с помощью диалогового окна Properties, будьте очень осторожны. В предыдущих версиях Visual Studio для удаления обработчиков применялся мастер ClassWizard, который не только запрашивал подтверждение на удаление прототипа и записи в карте сообщений, но и оставлял текст кода, чтобы его можно было впоследствии восстановить в случае ошибочного удаления. Диалоговое окно Properties подтверждения при удалении не запрашивает и удаляет обработчик события полностью.

Обработка дочерних событий

Итак, обработка сообщений классов, представленных на вкладке Class View (Классы), успешно изучена. Но как обработать сообщения, передаваемые элементами управления, расположенными в диалоговом окне (отличные от простого сообщения `BN_CLICKED`)? Диалоговое окно Properties позволяет сделать и это, но на сей раз с помощью другой вкладки. Чтобы продемонстрировать всю последовательность действий на конкретном примере, предположим, что необходимо проверять содержимое элемента управления всякий раз, когда пользователь покидает его.

1. Откройте панель Class View (Классы).
2. Найдите класс, отвечающий за то диалоговое окно, обработчик сообщений которого необходимо создать. В данном случае это класс `CHelloDialogDlg`.
3. Щелкните правой кнопкой мыши на имени класса и в появившемся контекстном меню выберите пункт Properties (Свойства).
4. В диалоговом окне Properties щелкните на пиктограмме Events (События) (расположенной в верхней части диалогового окна). Как можно заметить, окно содержит древовидное представление с двумя столбцами, в котором события сгруппированы по типам. Поскольку в распоряжении диалогового окна есть только события элемента управления, данная категория (Controls) будет единственной доступной на этой вкладке.
5. Внутри категории Controls (Элементы управления) находятся несколько элементов управления, которые были размещены в данном диалоговом окне (отсортированные по идентификаторам ресурса). Раскрыв ресурс `IDOK` (кнопка OK), можно увидеть не только обработчик сообщения `OnBnClickedOk`, созданный ранее в этой главе, но и ряд других сообщений, для которых возможно создание обработчиков (рис. 1.22).
6. Раскройте узел `IDC_EDIT1`. Теперь не составит большого труда написать код, осуществляющий проверку данных, введенных пользователем в поле ввода после того, как фокус ввода покинет его, для чего достаточно создать обработчик сообщения `EN_KILLFOCUS`.

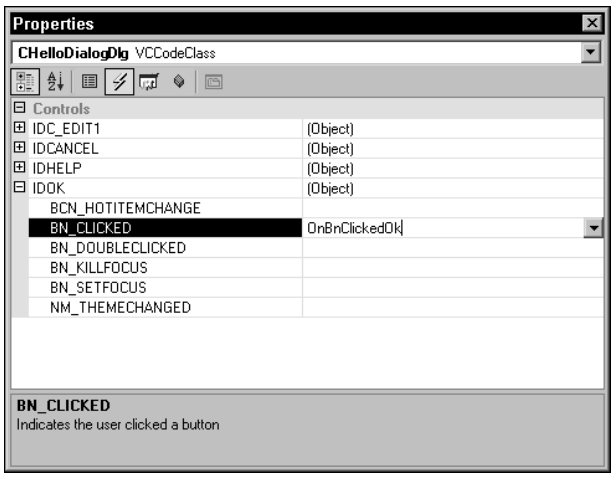


Рис. 1.22. Диалоговое окно Properties позволяет добавить обработчики и для дочерних событий диалогового окна

Переопределение функций базовых классов

Последняя тема, которую осталось рассмотреть в данной главе, посвящена *переопределению функций* (override function). Если один класс C++ происходит от другого, то он автоматически наследует *открытые* (public) и *защищенные* (protected) члены базового класса (включая как переменные, так и функции). В Visual Studio для переопределения функций базового класса также используется диалоговое окно Properties. Например, класс `CDialog` обладает виртуальной функцией `DestroyWindow`, вызов которой осуществляется в том случае, когда диалоговое окно получает сообщение `WM_DESTROY`. Следовательно, эту функцию придется переопределять достаточно часто, всякий раз, когда необходимо реализовать свой собственный код завершения работы, закрывающий, например, соединение с базой данных или освобождающий память, выделенную на протяжении периода существования диалогового окна. Итак, переопределить функции базового класса можно следующим образом.

1. Откройте панель Class View (Классы).
2. Найдите класс, отвечающий за то диалоговое окно, обработчик сообщений которого необходимо создать. В данном случае это класс `CHelloDialogDlg`.
3. Щелкните правой кнопкой мыши на имени класса и в появившемся контекстном меню выберите пункт Properties (Свойства).
4. В диалоговом окне Properties щелкните на пиктограмме Overrides (Переопределения) (расположенной в верхней части диалогового окна). Как можно заметить, окно содержит древовидное представление с двумя столбцами. Здесь переопределения событий сгруппированы в зависимости от вероятности возникновения необходимости реализации переопределения (“common” (обычно) или “uncommon” (иногда)).
5. В данном случае достаточно найти строку `DestroyWindow`, щелкнуть на втором столбце, а затем выбрать из списка пункт Add DestroyWindow.

Резюме

Настоящая глава предназначена для начинающих разработчиков, недостаточно хорошо знакомых с интерфейсом среды разработки Visual Studio. В ней обсуждается ряд наиболее часто используемых операций, неоднократно выполняемых на протяжении всей книги. Среди них: применение начальной страницы, создание диалогового приложения, добавление обработчика события, добавление элемента управления в диалоговое окно с помощью редактора диалоговых окон, использование DDX для связи элемента управления с переменной-членом, исправление ошибок компиляции и отладка простых приложений.

Хочется надеяться, что предоставленная здесь информация поможет читателю как на практике, так и при изучении остальных глав этой книги. Дальнейшее изложение предполагает, что читатель знаком с применением мастеров создания различных типов приложений и классов, с созданием переменных и обработчиков событий. Однако, материал первой главы всегда к услугам читателя, если потребуется освежить его в памяти. Итак перейдем к изучению программирования на Visual C++ с применением классов MFC.