

# **Visual C++.NET**

## **Классика программирования**

*Под ред. О. Е. Степаненко*

Москва «Научная книга» • Киев «Букинист»  
2010

ББК 32.973-01  
УДК 681.3.06  
С79

**Под редакцией О. Е. Степаненко**

С79 Visual C++.NET. Классика программирования.: - М: Научная книга, К.; Букинист, 2010. — 768 с.

ISBN 978-5-9315-0003-5

«Visual C++.NET. Классика программирования» — книга, необходимая как начинающему программисту, так и профессионалу, желающему познакомиться с новыми возможностями последней версии наиболее популярной системы программирования. Изучение этой книги не требует глубоких знаний языка C++, однако предполагает знание основ языка C. Поэтапное изучение предмета позволит человеку с любым уровнем начальных знаний о языке программирования C++ легко изучить самую современную его версию. Данная книга — не просто учебник по языку программирования, это первый шаг в создании удобных, профессиональных приложений, ориентированных на Internet.

**ББК 32.973-01**

*Учебное пособие*

**Visual C++.NET**

**Классика программирования**

Общество с ограниченной ответственностью «Букинист»

Подписано в печать 25.01.2010. Формат 70х 00 1/16.

Печать офсетная. Тираж 500 экз. Усл. печ. л. 48. Заказ № 2-1458

Отпечатано в типографии фирмы «ВИПОЛ» г. Киев

ISBN 978-5-9315-0003-5 (русск.)

© Оформление ООО «Научная книга», 2010

# Оглавление

---

<b>ВВЕДЕНИЕ .....</b>	<b>10</b>
ОБЗОР VISUAL STUDIO.NET .....	10
Платформа .NET.....	10
Отличия Visual Studio.NET от предыдущих версий.....	11
Варианты поставки Visual Studio.NET.....	11
ПЕРЕХОДИМ ОТ VISUAL STUDIO 5/6 К VISUAL STUDIO.NET.....	11
СТРУКТУРА КНИГИ.....	12
БАЗОВЫЕ ЗНАНИЯ .....	12
 <b>ЧАСТЬ I. ВВЕДЕНИЕ В MICROSOFT VISUAL C++.NET.....</b>	<b>13</b>
<b>ГЛАВА 1. УСТАНОВКА VISUAL C++.NET .....</b>	<b>14</b>
УСТАНОВКА MICROSOFT VISUAL STUDIO.NET .....	14
Установка справочной системы Visual Studio.NET.....	15
СОСТАВ ПАКЕТА VISUAL STUDIO.NET .....	16
РЕЗЮМЕ .....	16
 <b>ГЛАВА 2. ПРОГРАММИРОВАНИЕ В СРЕДЕ VISUAL STUDIO.NET.....</b>	<b>17</b>
СОЗДАНИЕ ПРОЕКТА.....	17
ИСХОДНЫЙ ФАЙЛ ПРОГРАММЫ .....	19
Вкладки Solution Explorer и Class View.....	23
Справочная система.....	23
КОНФИГУРИРОВАНИЕ ПРОЕКТА .....	24
ПОСТРОЕНИЕ ПРОГРАММЫ .....	24
ОТЛАДКА ПРОГРАММЫ .....	24
РЕЗЮМЕ .....	27
 <b>ЧАСТЬ II. ВВЕДЕНИЕ В C++ .....</b>	<b>28</b>
<b>ГЛАВА 3. ОТ C К C++ .....</b>	<b>29</b>
ОТ C К C++ .....	29
НОВИНКИ C++ .....	33
Оформление комментариев .....	33
Объявление переменных.....	34
Расширение области видимости.....	36
Встроенные функции.....	37
Значения по умолчанию параметров функции.....	38
Ссылки .....	39
Переменные и константы.....	43
Перегруженные функции.....	48
Операторы new и delete.....	49
РЕЗЮМЕ.....	52

<b>ГЛАВА 4. КЛАССЫ C++ .....</b>	<b>54</b>
ОПРЕДЕЛЕНИЕ КЛАССА .....	54
ЭКЗЕМПЛЯР КЛАССА И ДОСТУП К НЕМУ .....	55
ИНКАПСУЛЯЦИЯ .....	57
КОНСТРУКТОР И ДЕКТРУКТОР КЛАССА .....	60
<i>Конструктор класса</i> .....	60
<i>Дектруктор</i> .....	66
<i>Вызов конструктора и дектруктора</i> .....	67
ВСТРОЕННЫЕ ФУНКЦИИ-ЧЛЕНЫ .....	67
РАЗМЕЩЕНИЕ ОПРЕДЕЛЕНИЯ КЛАССА В ПРОГРАММЕ .....	69
УКАЗАТЕЛЬ THIS .....	71
СТАТИЧЕСКИЕ ЧЛЕНЫ КЛАССА .....	72
РЕЗЮМЕ .....	74
 <b>ГЛАВА 5. КЛАССЫ-НАСЛЕДНИКИ C++ .....</b>	<b>76</b>
КЛАССЫ-НАСЛЕДНИКИ (ПРОИЗВОДНЫЕ КЛАССЫ) .....	76
<i>Конструктор</i> .....	78
<i>Доступ к наследуемым переменным</i> .....	79
ИЕРАРХИЯ КЛАССОВ .....	81
<i>Иерархия классов на вкладке Class View</i> .....	82
ВИРТУАЛЬНЫЕ ФУНКЦИИ .....	83
<i>Управление объектами классов с помощью виртуальных функций</i> .....	86
<i>Модификации базовых классов с помощью виртуальных функций</i> .....	87
РЕЗЮМЕ .....	89
 <b>ГЛАВА 6. ПЕРЕГРУЗКА, КОПИРОВАНИЕ И ПРЕОБРАЗОВАНИЕ .....</b>	<b>90</b>
ПЕРЕГРУЗКА ОПЕРАТОРОВ .....	90
<i>Определение функций-операторов</i> .....	92
<i>Общие принципы перегрузки операторов</i> .....	95
<i>Перегрузка оператора присваивания</i> .....	96
КОНСТРУКТОРЫ КОПИРОВАНИЯ И ПРЕОБРАЗОВАНИЯ .....	99
<i>Конструктор копирования</i> .....	100
<i>Конструктор преобразования</i> .....	102
<i>Инициализация массивов</i> .....	107
РЕЗЮМЕ .....	108
 <b>ГЛАВА 7. ШАБЛОНЫ В C++ .....</b>	<b>109</b>
ШАБЛОНЫ ФУНКЦИЙ .....	109
<i>Переопределение шаблона</i> .....	112
ШАБЛОНЫ КЛАССОВ .....	112
<i>Порождение объектов по шаблонам</i> .....	114
<i>Конструктор в шаблоне функции</i> .....	116
РЕЗЮМЕ .....	118
 <b>ГЛАВА 8. ИСКЛЮЧИТЕЛЬНЫЕ СИТУАЦИИ .....</b>	<b>119</b>
ПРОГРАММНЫЕ ИСКЛЮЧЕНИЯ И ИХ ОБРАБОТКА .....	119
<i>Catch-блоки</i> .....	123



Универсальный или специальный обработчик? .....	124
Вложенные исключения .....	126
WIN32-ИСКЛЮЧЕНИЯ И ИХ ОБРАБОТКА .....	128
РЕЗЮМЕ .....	132
<b>ЧАСТЬ III. ПРОГРАММИРОВАНИЕ ГРАФИЧЕСКОГО ИНТЕРФЕЙСА....</b>	<b>134</b>
<b>ГЛАВА 9. ПРОГРАММА С ГРАФИЧЕСКИМ ИНТЕРФЕЙСОМ .....</b>	<b>135</b>
КАК СПРОЕКТИРОВАТЬ ГРАФИЧЕСКИЙ ИНТЕРФЕЙС .....	135
ПРОЕКТИРОВАНИЕ ПРОГРАММЫ .....	136
Как сгенерировать исходный код .....	136
Внесение изменений в сгенерированный код .....	138
Как скомпоновать и запустить программу .....	141
СОСТАВ ПРОЕКТА .....	142
КАК РАБОТАЕТ ПРОГРАММА .....	155
Последовательность выполнения программы .....	155
Как работает функция <i>InitInstance</i> .....	157
РЕЗЮМЕ .....	160
<b>ГЛАВА 10. КАК ОФОРМИТЬ ПРЕДСТАВЛЕНИЕ .....</b>	<b>162</b>
ПРОСТОЙ ГРАФИЧЕСКИЙ РЕДАКТОР .....	162
Исходные файлы редактора .....	163
Переменные класса представления .....	164
Обработчики сообщений .....	164
Ресурсы программы <i>ScratchBook</i> .....	172
Конфигурирование окна <i>ScratchBook</i> .....	175
Тексты программы <i>ScratchBook</i> .....	176
ПРОСТОЙ ТЕКСТОВЫЙ РЕДАКТОР .....	187
<i>MyScribe</i> – проектирование программы .....	188
Тексты программы <i>MyScribe</i> .....	191
РЕЗЮМЕ .....	201
<b>ГЛАВА 11. ДАННЫЕ В ДОКУМЕНТЕ .....</b>	<b>203</b>
СОХРАНЕНИЕ ДАННЫХ И ПЕРЕРИСОВКА ОКНА .....	203
МОДИФИКАЦИЯ МЕНЮ .....	207
Реализация команды <i>Remove All</i> .....	208
Реализация команды <i>Undo</i> .....	209
УДАЛЕНИЕ ДАННЫХ .....	210
ТЕКСТ ПРОГРАММЫ <i>SCRATCHBOOK</i> .....	211
РЕЗЮМЕ .....	224
<b>ГЛАВА 12. ВВОД-ВЫВОД .....</b>	<b>226</b>
ВВОД-ВЫВОД В ПРОГРАММЕ <i>SCRATCHBOOK</i> .....	226
Модификация меню <i>File</i> .....	226
Реализация команд .....	227
Текст программы <i>ScratchBook</i> .....	232
ВВОД-ВЫВОД В ПРОГРАММЕ <i>MYSCRIBE</i> .....	245

Модификация меню.....	246
Реализация команд.....	246
Текст программы MyScribe.....	247
Альтернативные способы ввода-вывода.....	258
РЕЗЮМЕ.....	259
<b>ГЛАВА 13. УПРАВЛЕНИЕ ОКНАМИ ПРЕДСТАВЛЕНИЯ .....</b>	<b>260</b>
ПРОКРУТКА ОКНА .....	260
Логические и фактические координаты.....	260
Границы рисунка в окне представления.....	262
РАЗДЕЛЕНИЕ ОКНА .....	266
ПЕРЕРИСОВКА ОКНА .....	267
ТЕКСТ ПРОГРАММЫ SCRATCHBOOK .....	272
РЕЗЮМЕ.....	288
<b>ГЛАВА 14. ПАНЕЛЬ ИНСТРУМЕНТОВ И СТРОКА СОСТОЯНИЯ .....</b>	<b>290</b>
ПЕРЕМЕЩАЕМАЯ ПАНЕЛЬ ИНСТРУМЕНТОВ В ПРОГРАММЕ SCRATCHBOOK.....	290
Модификация ресурсов .....	291
Модификация меню .....	292
Модификация текста программы.....	293
Диалоговые и переключаемые панели.....	300
СТРОКА СОСТОЯНИЯ В ПРОГРАММЕ SCRATCHBOOK .....	301
ТЕКСТ ПРОГРАММЫ SCRATCHBOOK .....	302
РЕЗЮМЕ.....	321
<b>ГЛАВА 15. ДИАЛОГОВЫЕ ОКНА .....</b>	<b>322</b>
МОДАЛЬНЫЕ ДИАЛОГОВЫЕ ОКНА.....	322
Программа FontView.....	323
Диалоговое окно Text Properties .....	323
Управление диалоговым окном.....	329
Обработчики сообщений.....	331
MFC-классы и функции для элементов управления и диалоговых окон .....	331
Управление диалоговым окном Text Properties .....	334
Отображение диалогового окна.....	339
Текст программы FontView.....	344
НЕМОДАЛЬНОЕ ДИАЛОГОВОЕ ОКНО.....	360
ДИАЛОГОВОЕ ОКНО С ВКЛАДКАМИ.....	361
Текст программы TabView.....	365
ДИАЛОГОВОЕ ОКНО ОБЩЕГО НАЗНАЧЕНИЯ .....	380
РЕЗЮМЕ.....	381
<b>ГЛАВА 16. ДИАЛОГОВЫЕ ПРОГРАММЫ.....</b>	<b>382</b>
ПРОСТАЯ ПРОГРАММА С ОКНОМ ДИАЛОГА DIALOGVIEW .....	382
Создание программы DialogView.....	382
Формирование диалогового окна .....	383
Текст программы DialogView .....	387
ПРОГРАММА ПРОСМОТРА ФОРМЫ FORMVIEW.....	393
Генерация и настройка программы FormView.....	394

Текст программы <i>FormView</i> .....	399
РЕЗЮМЕ .....	410
<b>ГЛАВА 17. МУЛЬТИДОКУМЕНТНЫЕ ПРОГРАММЫ.....</b>	<b>411</b>
МУЛЬТИДОКУМЕНТНЫЙ ИНТЕРФЕЙС.....	411
МУЛЬТИДОКУМЕНТНАЯ ВЕРСИЯ ПРОГРАММЫ <i>MyScribe</i> .....	412
Классы и программный код .....	412
РЕСУРСЫ.....	415
ТЕКСТ ПРОГРАММЫ <i>MyScribe</i> .....	416
РЕЗЮМЕ .....	428
<b>ГЛАВА 18. ВВОД/ВЫВОД СИМВОЛОВ.....</b>	<b>430</b>
ОТОБРАЖЕНИЕ ТЕКСТА В ОКНЕ ПРЕДСТАВЛЕНИЯ.....	430
Отображение строк .....	431
Сохранение текста и объект <i>Font</i> .....	435
Средства прокрутки.....	445
Модификация функции <i>InitInstance</i> .....	447
ВВОД СИМВОЛОВ С КЛАВИАТУРЫ.....	447
Сообщение <i>WM_KEYDOWN</i> .....	447
Сообщение <i>WM_CHAR</i> .....	452
ТЕКСТОВЫЙ КУРСОР.....	453
ТЕКСТ ПРОГРАММЫ <i>FontInfo</i> .....	455
РЕЗЮМЕ .....	473
<b>ГЛАВА 19. СРЕДСТВА РИСОВАНИЯ .....</b>	<b>475</b>
ОБЪЕКТ КОНТЕКСТА УСТРОЙСТВА .....	475
ИНСТРУМЕНТЫ РИСОВАНИЯ .....	476
ГРАФИЧЕСКИЕ АТТРИБУТЫ .....	482
РИСОВАНИЕ.....	485
Точка.....	485
Текст программы <i>FractalView</i> .....	490
Отрезки линий.....	501
ПРОГРАММА <i>ScratchBook</i> .....	508
Классы фигур.....	512
Текст программы <i>ScratchBook</i> .....	526
РЕЗЮМЕ .....	556
<b>ГЛАВА 20. РАСТРОВЫЕ ИЗОБРАЖЕНИЯ И БИТОВЫЕ ОПЕРАЦИИ.....</b>	<b>557</b>
РАСТРОВЫЕ ИЗОБРАЖЕНИЯ.....	557
Растровое изображение в ресурсах .....	558
Рисование растрового изображения.....	559
Отображение растрового изображения .....	561
БИТОВЫЕ ОПЕРАЦИИ.....	563
Функция <i>PatBlt</i> .....	564
Функция <i>BitBlt</i> .....	565
Функция <i>StretchBlt</i> .....	568
ЗНАЧКИ.....	569

ПРОГРАММА CHESSBOARD .....	571
<i>Текст программы ChessBoard</i> .....	573
РЕЗЮМЕ.....	583
<b>ГЛАВА 21. ПЕЧАТЬ И ПРЕДВАРИТЕЛЬНЫЙ ПРОСМОТР.....</b>	<b>585</b>
ПРОСТАЯ ПЕЧАТЬ И ПРЕДВАРИТЕЛЬНЫЙ ПРОСМОТР .....	585
УСОВЕРШЕНСТВОВАННАЯ ПЕЧАТЬ .....	588
<i>Размер рисунка</i> .....	589
<i>Переопределение виртуальных функций печати</i> .....	590
<i>Модификация функции OnDraw</i> .....	594
ТЕКСТ ПРОГРАММЫ SCRATCHBOOK .....	596
РЕЗЮМЕ.....	626
<b>ГЛАВА 22. ПОТОКИ.....</b>	<b>628</b>
ВТОРИЧНЫЕ ПОТОКИ.....	628
<i>Завершение потока</i> .....	630
<i>Функции управления потоком</i> .....	630
<i>Ограничения на использование MFC-классов</i> .....	632
СПОСОБЫ СИНХРОНИЗАЦИИ ПОТОКОВ .....	632
<i>Мьютекс и другие объекты синхронизации</i> .....	634
МНОГОПОТОКОВАЯ ПРОГРАММА РИСОВАНИЯ IMPFRACTAL.....	637
<i>Текст программы ImpFractal</i> .....	641
РЕЗЮМЕ .....	652
<b>ГЛАВА 23. ПРОЦЕССЫ.....</b>	<b>654</b>
ЗАПУСК ПРОЦЕССА.....	654
ДЕСКРИПТОРЫ ОБЩИХ ОБЪЕКТОВ .....	657
КАНАЛЫ И ПОЧТОВЫЕ ЯЩИКИ.....	659
ФАЙЛЫ ПАМЯТИ И СОВМЕСТНЫЙ ДОСТУП.....	660
БУФЕР ОБМЕНА.....	661
<i>Команды управления</i> .....	662
<i>Обмен текстом через буфер</i> .....	663
<i>Обмен графикой через буфер обмена</i> .....	671
<i>Данные зарегистрированных форматов в буфере обмена</i> .....	675
РЕЗЮМЕ.....	676
<b>ГЛАВА 24. OLE-МЕХАНИЗМ В VISUAL C++.....</b>	<b>678</b>
ВНЕДРЕНИЕ, СВЯЗЫВАНИЕ И АВТОМАТИЗАЦИЯ .....	678
ПРОГРАММА-СЕРВЕР EXPSERVER .....	680
<i>Генерация программы-сервера</i> .....	681
<i>Модификация кода приложения-сервера</i> .....	685
<i>Текст программы ExpServer</i> .....	690
ПРОГРАММА-КОНТЕЙНЕР EXPCONTAINER.....	710
<i>Классы</i> .....	710
<i>Ресурсы</i> .....	713
<i>Отладка программы ExpContainer</i> .....	713
<i>Текст программы ExpContainer</i> .....	715
РЕЗЮМЕ.....	733

<b>ГЛАВА 25. ACTIVEX И VISUAL C++ .....</b>	<b>734</b>
<b>СОБСТВЕННЫЙ ACTIVEX-ЭЛЕМЕНТ .....</b>	<b>734</b>
<i>Генерация и модификация программных файлов .....</i>	<i>735</i>
<i>Свойства ActiveX .....</i>	<i>737</i>
<i>Методы ActiveX .....</i>	<i>740</i>
<i>События ActiveX .....</i>	<i>740</i>
<i>Построение ActiveX-элемента .....</i>	<i>740</i>
<i>Текст элемента ActiveXControl .....</i>	<i>741</i>
<b>ПРОГРАММА-КОНТЕЙНЕР ACTIVEX-ЭЛЕМЕНТА .....</b>	<b>749</b>
<i>Текст программы ActiveXContainer .....</i>	<i>750</i>
<b>РЕЗЮМЕ .....</b>	<b>755</b>
 <b>ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ .....</b>	 <b>756</b>

# Введение

---

Инструментальный комплекс проектирования, отладки и сопровождения программ Visual C++ – один из наиболее полных и совершенных продуктов, предназначенных для разработки программного обеспечения. Это высокоскоростная и удобная для программирования система, предлагающая широкий набор разнообразных инструментов проектирования для любого стиля программирования. В Visual C++ .NET, входящей в состав Visual Studio.NET, функции системы существенно расширены по сравнению с предыдущими версиями. Новые компоненты содержат средства для программирования приложений (их построения и отладки), улучшенную реализацию технологий работы с ActiveX и Internet, дополнительные возможности разработки баз данных, а также новые архитектуры приложений и элементы взаимодействия между пользователями в сложных сетевых архитектурах.

Эта книга – единственное исчерпывающее пошаговое руководство по Visual C++ – построена таким образом, чтобы ее могли читать даже новички. Она дает общее представление о программном продукте, необходимое для решения конкретных задач. Данная книга намного больше, чем обзор системы программирования или введение в нее. Подробный анализ и наглядные примеры ознакомят вас с основными инструментами и приемами программирования. Автор ставил себе целью не столько полностью описать обширный инструментальный набор, сколько правильно сориентировать читателя в среде Visual C++, дать ему набор базовых навыков, основываясь на которых он сможет в дальнейшем самостоятельно разобраться в инструментарии, технологиях проектирования программ, средствах их сопровождения и эффективно применить приобретенные знания для решения своих задач.

## Обзор Visual Studio.NET

---

### Платформа .NET

Microsoft.NET – это универсальная программная платформа, предназначенная, в первую очередь, для программ, ориентированных на Internet, а именно для разработки Web-приложений и Web-служб (Web-сервисов). Web-приложения реализуют доступ к ресурсам Web-серверов для пользователей (через Internet-браузер), а Web-служб – для программ (в формате XML с использованием протокола SOAP).

Исходные тексты .NET-приложений компилируются в специальный промежуточный код на языке MSIL (Microsoft Intermediate Language) и выполняются в среде .NET Framework, состоящей из виртуальной машины CLR (Common Language Runtime) и библиотеки классов .NET Framework Class Library. CLR реализует проверку и компиляцию кода “на лету” из MSIL в команды процессора, управляет памятью, процессами и потоками, решает вопросы безопасности. Библиотека классов .NET Framework не зависит от языка программирования, и ее могут использовать все .NET-приложения. В ней содержатся классы Windows Form (предназначен для разработки графического пользовательского интерфейса), Web Forms (предназначен для создания Web-приложений и Web-служб на основе технологии ASP.NET), классы для работы с XML и Internet-протоколами (FTP, HTTP, SMTP, SOAP), библиотека ADO.NET (для работы с базами данных) и многое другое. В .NET Framework используется собственная компонентная модель, элементами которой являются сборки (assembly), а для прямой и обратной совместимости с моделью COM/COM+ в CLR введено группу механизмов под названием COM Interop, обеспечивающих доступ к COM-объектам по правилам .NET и к .NET-сборкам по правилам COM. При этом модель .NET не требует регистрации компонентов в системном реестре Windows, а для работы .NET-приложения достаточно поместить все относящиеся к нему сборки в один с ним каталог. Если какую-либо сборку необходимо использовать в нескольких приложениях, то с помощью специальной утилиты ее можно зарегистрировать в общем кэше Global Assembly Cache (GAC).

.NET Framework вместе с полной документацией по всей платформе с примерами распространяется бесплатно, а загрузить ее последнюю версию можно прямо с сайта Microsoft ([www.microsoft.com/net/downloads.asp](http://www.microsoft.com/net/downloads.asp)).

## **Отличия Visual Studio.NET от предыдущих версий**

Собственно, среда разработки Visual Studio.NET похожа на предыдущую версию, но в ней имеется ряд существенных нововведений и усовершенствований.

Безусловно, самым важным нововведением является наличие новых типов проектов, среди которых на первом месте по значимости стоят Web-службы, создаваемые в Visual C++.NET на базе новой библиотеки ATL Server Library или с помощью Managed Extension. С точки зрения программирования, Web-службы должны удовлетворять весьма сложным требованиям. Вопреки сказанному, Visual Studio.NET полностью скрывает все указанные нюансы.

Наконец-то вместо отдельных оболочек для различных языков программирования, которые существовали до Visual Studio 6 включительно, Microsoft создала единую интегрированную среду разработки, не зависящую от используемого языка. Благодаря этому сегодня возможности интеграции в Visual Studio .NET реализованы уже более чем для 20 языков, причем одновременно можно работать с проектами, реализованными на различных языках.

Полезной функцией является динамическая подсказка, которая в процессе работы с исходными текстами автоматически предлагает наиболее подходящие разделы справочника, сортируя их по убыванию значимости.

При редактировании текста программы может выполняться фоновая проверка синтаксиса языка. Фрагмент, в котором допущена ошибка, подчеркивается волнистой линией.

Если компьютер подключен к Internet, Visual Studio.NET автоматически выполняет поиск справочной информации на сайте Microsoft, а также обнаруживаются, загружаются и устанавливаются все обновления продукта.

## **Варианты поставки Visual Studio.NET**

Visual Studio.NET распространяется в трех редакциях – Professional, Enterprise Developer и Enterprise Architect. В Enterprise-версиях входят средства для моделирования, проектирования баз данных и интеграции с продуктами семейства .NET Enterprise Servers (Developer – частично, а Architect – полностью). Комплект Visual Studio.NET Enterprise Architect – наиболее полная версия – состоит из 5 CD.

## **Переходим от Visual Studio 5/6 к Visual Studio.NET**

Несмотря на то, что Visual Studio.NET предназначена в первую очередь для разработки программного обеспечения нового типа, т.е. Internet-приложений, большинство ее пользователей, вероятнее всего, будут составлять программы, ранее работавшие с Visual Studio предыдущих версий – пятой или шестой. Поэтому рассмотрим сначала, как переход на Visual Studio.NET согласуется с сохранением приобретенных до этого опыта и наработок. В этом смысле интересны, по крайней мере, два вопроса – как перенести в Visual Studio.NET проекты из Visual Studio 5/6 и можно ли создавать новое программное обеспечение, пользуясь навыками программирования, полученными в предыдущих версиях, или все же придется переучиваться. Кардинально перестроив базовую архитектуру самой Visual Studio.NET и изменив целевую платформу разрабатываемых в ней приложений, Microsoft сделала так, чтобы пользователи предыдущих версий Visual Studio смогли применить свои навыки и опыт, приобретенные ранее. Во-первых, сохранился все тот же удобный визуальный подход к созданию приложений (перетащить & опустить & выполнить двойной щелчок мышью). Так, чтобы расширить функциональность программы (добавить элемент управления, запрос к базе данных и др.), достаточно, как и ранее, выбрать соответствующий компонент в инструментальной панели, перетащить его на форму и дважды щелкнуть на нем мышью, чтобы оформить обработчик события. При этом поддерживаются как новые компоненты .NET, так и элементы управления ActiveX. Получается так, что программист, знакомый с предыдущими версиями Visual Studio, может создавать Windows-приложения в Visual Studio.NET аналогично тому, как он это делал в предыдущих версиях: открыть новый проект, нарисовать формы пользовательского интерфейса, написать обработчики, скомпилировать, отладить и создать исполняемый файл. Ни программист, ни пользователь даже не заметят, что они используют платформу .NET. Затем впоследствии, когда возникнет необходимость обратиться к Web-службе или взаимодействовать с другими .NET-приложениями, доработать программы будет совсем просто. Кроме того, указанный подход можно теперь применить и для создания Web-приложений. Эта одна из ключевых возможностей Visual Studio.NET. Благодаря ней разработка Web-приложений стала значительно

проще и нагляднее. Все сложности, связанные с обменом данными между клиентским браузером и Web-сервером, на котором выполняется обработка запросов, берет на себя сама среда разработки. Программист же может сосредоточиться на дизайне страниц и функциональности приложения.

Что касается непосредственно Visual Studio.NET, то в целом для него остается в силе старая технология разработки. Правда, появилась возможность создавать Web-службы на базе новой библиотеки ATL Server Library или с помощью Managed Extensions.

Теперь вернемся к первому вопросу. Перенести существующие проекты в Visual Studio.NET далеко не так легко, причем для каждого языка программирования приходится решать свои проблемы. Переход же с Visual C++ 6 на Visual C++.NET – наиболее безболезненный, поскольку они отличаются лишь поддержкой специальных расширений Managed Extensions. Если их не использовать, то компилятор Visual C++.NET будет создавать обычные “старые” исполняемые модули, для запуска которых никакой необходимости в .NET Framework нет. Поэтому практически все проекты переносятся в Visual C++.NET очень просто и без каких-либо затруднений. Microsoft намеренно сделала Visual C++.NET таким “гибридным” языком именно для поддержки ранее созданного программного обеспечения. Его перевод на платформу .NET может выполняться постепенно. Применение Managed Extensions позволяет внутри одного приложения смешивать код, компилируемый в команды процессора, с .NET-кодом, транслируемым в MSIL. Таким образом, можно легко добавить к старой программе классы, которые позволят обращаться к ней из .NET-приложений, а уже затем, при необходимости, переделать все остальное.

## Структура книги

---

- *Первая часть* (гл. 1 и 2) посвящена установке Visual C++. В ней также дано общее описание основных компонентов Visual C++. Здесь вы ознакомитесь с базовыми приемами написания и построения программ с использованием встроенной среды разработки Developer Studio. Работа в Developer Studio рассмотрена на уровне, достаточном для создания и отладки программ, приведенных во второй и третьей частях книги.
- *Вторая часть* (гл. 3 — 8) предлагает читателю введение в язык программирования C++, которое поможет перейти от программирования на языке C к программированию на C++. Внимание уделяется использованию основных приемов языка и его компонентов при работе с библиотекой MFC (Microsoft Foundation Classes).
- *Третья часть* (гл. 9 — 25) – основная часть книги. В ней рассмотрено программирование графического интерфейса в среде Microsoft Windows 2000, Windows NT 4.0 и более поздних версиях этой операционной системы. В этой части книги показано, как использовать Developer Studio и различные средства разработки, такие, как мастер Application Wizard, редакторы ресурсов. Использование перечисленных средств делает процесс программирования сравнительно простым. В части III рассмотрено не только программирование Windows, но и несколько относительно сложных тем, таких как реализация представлений разделяемых окон, отображение полос состояния и перемещаемых панелей инструментов, написание приложений многодокументного интерфейса (multiple document interface – MDI), применение функций рисования и битовых карт, печать предварительных просмотров, использование параллельных потоков выполнения и обмен данными с OLE (связывание и встраивание объектов). Отдельная глава посвящена созданию компонентов ActiveX – технологии, рекомендуемой фирмой Microsoft в качестве замены для OLE.

## Базовые знания

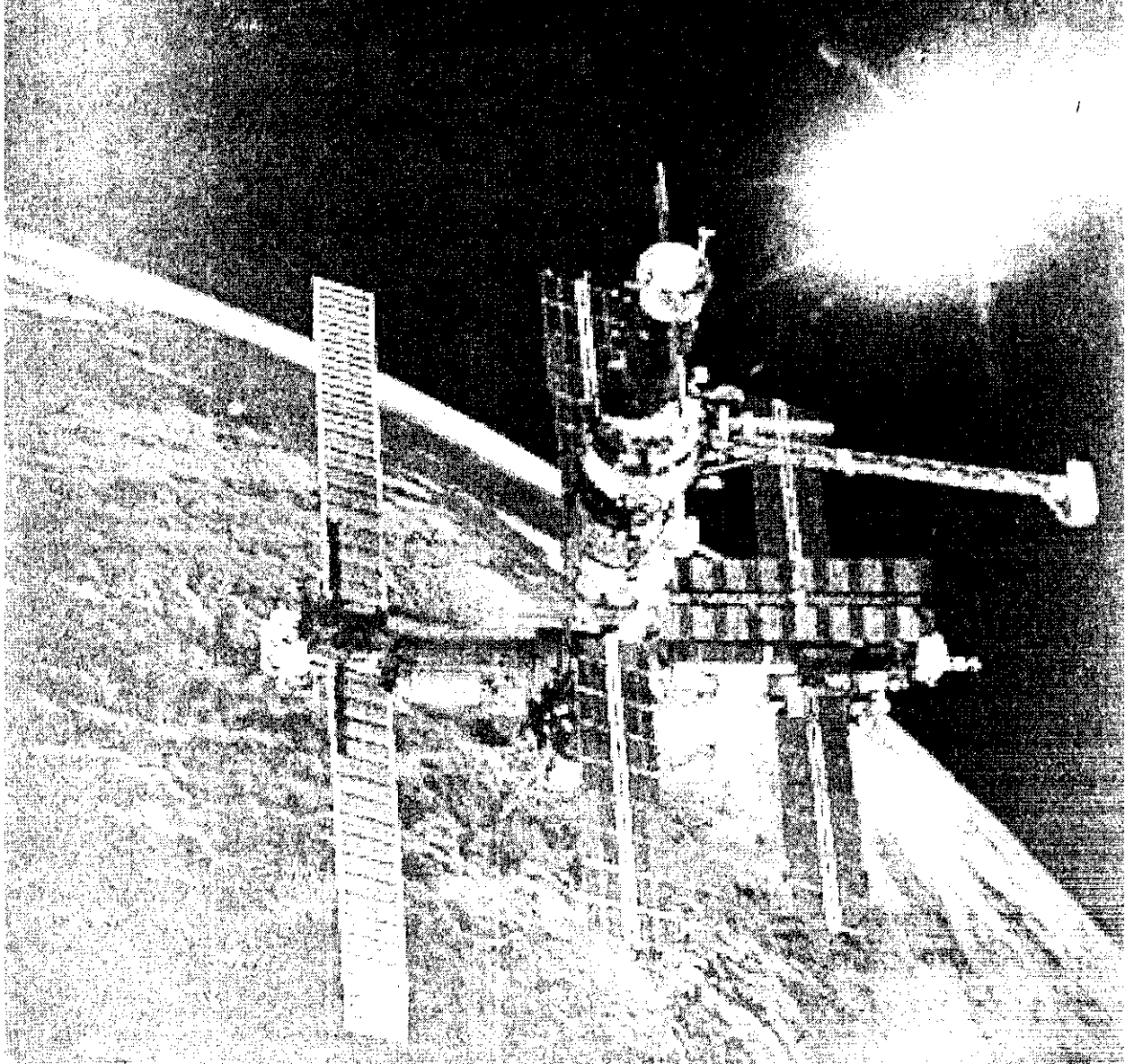
---

Чтобы начать работу с этой книгой читателю не обязательно обладать глубокими знаниями языка C++ и приемов программирования графического интерфейса Windows. Однако предполагается, что основные знания языка C у вас все-таки имеются. Многие концепции языка C++ объясняются с помощью концепций языка C. Если же вам понадобится более детальная информация о языке C, воспользуйтесь одной из книг, предлагаемых автором: “Язык программирования C” (“The C Programming Language”) Кернигана (Kernighan) и Ритчи (Ritchie) или “Справочное руководство по C” (“C: A Reference Manual”) Харбисона (Harbison) и Стила (Steele).



# Часть I

## Введение в Microsoft Visual C++.NET



# Глава 1

## Установка Visual C++.NET

---

- Установка Microsoft Visual Studio.NET и Visual C++.NET
- Состав пакета Visual C++.NET

В этой главе описана установка системы программирования Microsoft Visual C++.NET в составе Microsoft Visual Studio.NET и приведен обзор компонентов Visual C++, позволяющий ознакомиться с данным программным продуктом и выбрать подходящий вариант его установки. Установка Visual C++.NET без установки Visual Studio.NET и платформы .NET Frameworks невозможна, поскольку они обеспечивают выполнение базовых функций пакета.

### Установка Microsoft Visual Studio.NET

---

Для работы с системой Visual C++.NET необходимо, чтобы на компьютере была установлена операционная система Windows 2000, Windows NT 4.0, Windows XP или более поздняя версия одной из этих операционных систем (дополнительные требования приведены в документации по Visual Studio.NET).

Требования к аппаратному обеспечению из документации по установке:

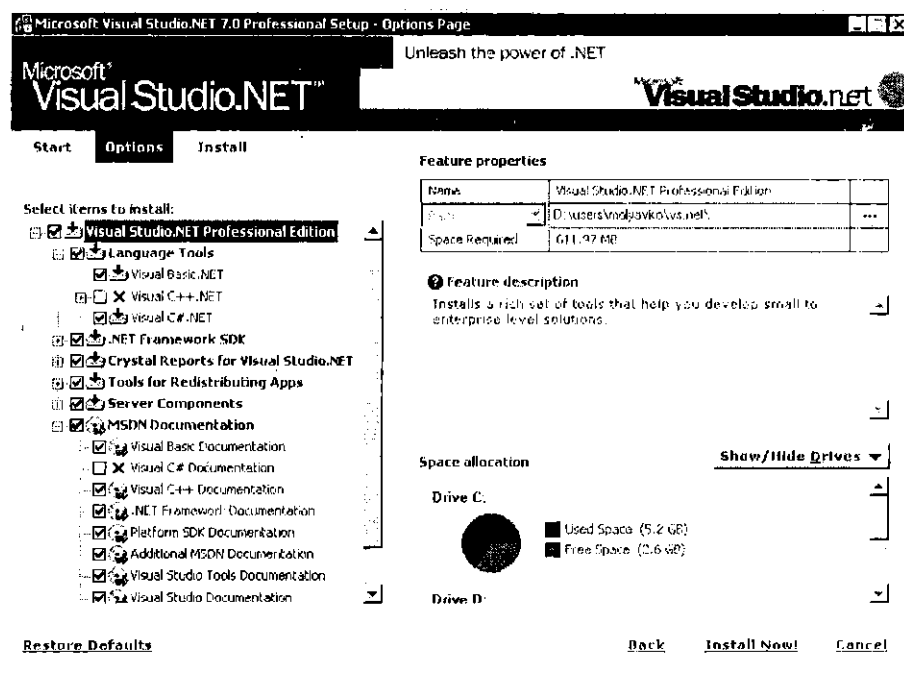
- объем оперативной памяти – 96 МБ для Windows 2000 Professional, 192 МБ для Windows 2000 Server (рекомендуется 128 МБ для Professional, 256 МБ для Server);
- процессор – Pentium II 400 (рекомендуется Pentium III 650);
- место на жестком диске (полная инсталляция) – 500 МБ на системном диске, 2.5 ГБ на диске, куда ведется собственно инсталляция.

Visual Studio.NET поставляется на 4-х CD или 1 DVD.

1. Для начала инсталляция следует запустить с первого инсталляционного диска файл setup.exe – программу установки.
2. Программа установки пытается выполнить необходимые обновления операционной системы вашего компьютера. Через некоторое время будет показан экран с единственной доступной опцией – Windows Component Update. Обновление Windows включает установку компонентов операционной системы, необходимых для работы Visual Studio.NET. Их список меняется от компьютера к компьютеру, поскольку при запуске программа установки анализирует установленное на компьютере программное обеспечение и исходя из результатов анализа формирует список устанавливаемых компонентов.
3. Перед началом обновления будет показано окно, требующее принять лицензионное соглашение для продолжения.
4. В ходе обновления компьютер может (при необходимости) несколько раз перезагрузиться, после каждой перезагрузки установка будет автоматически продолжена. Поскольку установка может проводиться на компьютер, где перезагрузка может потребовать ввода пароля, будет показан экран, где есть возможность ввести пароль для его автоматического ввода после каждой перезагрузки (если это нежелательно, не устанавливайте в этом окне опцию Automatically Log On).
5. После завершения установки компонентов обновления на начальном экране установки станет доступна вторая опция – Visual Studio.NET. Ее выбор приведет к появлению окна, в котором

необходимо будет ввести имя пользователя, регистрационный номер и принять лицензионное соглашение. После выполнения этих действий щелкните на надписи (ссылке) Continue.

6. На следующем экране производится выбор компонентов, подлежащих установке на компьютере и диска/папки, куда будет устанавливаться Visual Studio.NET. В дальнейшем все устанавливаемые компоненты и подкомпоненты будем называть просто *компонентами*. Далее в этой главе перечисляются компоненты Visual Studio.NET. Эту информацию целесообразно прочитать до выбора и установки компонентов.



7. После щелчка на надписи Install Now! начнется процесс инсталляции, который в зависимости от выбранных опций и характеристик оборудования может потребовать от 30 минут до 2-х часов.
8. В случае, если инсталляция окажется неудачной, откройте Control Panel и запустите утилиту Add/Remove Programs. В открывшемся списке выберите пункт Visual Studio.NET и щелкните на кнопке Change/Remove. Будет запущена программа установки, позволяющая:
  - доустановить компоненты;
  - восстановить поврежденную копию Visual Studio.NET на винчестере;
  - провести деинсталляцию.

#### Совет

Чтобы удалить установленные или добавить новые компоненты, можно также повторно запустить программу Setup с инсталляционного диска.

## Установка справочной системы Visual Studio.NET

Для эффективного программирования на вашем компьютере должна быть установлена справочная система (MSDN). Далее в книге есть ссылки на справочную систему, а в параграфе “Справочная система” гл. 2 описаны методы ее использования. В ходе установки Visual Studio.NET есть возможность

выбрать конфигурацию устанавливаемой копии MSDN и установить ее одновременно с Visual Studio.NET. После установки доступ к MSDN можно получить из меню Help или запустив MSDN как отдельную программу из меню Start (Пуск) операционной системы.

## **Состав пакета Visual Studio.NET**

---

В данном параграфе описаны компоненты Visual C++, а также инструменты и вспомогательные средства, устанавливаемые с помощью программы Setup.

Visual C++ содержит следующие компоненты:

- *Visual C++ Class & Template Libraries* – различные библиотеки классов и шаблонов (библиотеки Microsoft Foundation Classes).
- *Visual C++ Tools* – различные инструменты, реализующие дополнительные возможности
  - MFC Trace Utility – утилита для трассировки в библиотеке MFC и выбора сообщений, посылаемых в окно вывода при выполнении
  - Spy++ – утилита для просмотра сообщений, получаемых выбранными окнами
  - OLE/COM Object Viewer – утилита для просмотра COM-объектов
  - ActiveX Control Test Container – программа для встраивания и тестирования создаваемых ActiveX-компонентов
  - Visual C++ Error Lookup – утилита для обработки сообщений об ошибках компиляции, связывания и других
  - ISAPI Web Debug Tool – для отладки Web-приложений
  - Platform SDK Tools – набор инструментов для отладки в среде Windows.
- *Visual C++ Runtime Libraries* – библиотеки, используемые при выполнении модулями, создаваемыми Visual C++.
- *Другие компоненты.* Кроме Visual C++, программа установки позволяет устанавливать Visual C#, Visual Basic, MSDN и некоторые дополнительные инструменты: (Crystal Reports, инструменты для подготовки созданных приложений к распространению, библиотеки стандартных значков, средства удаленной отладки и т.д.)

## **Резюме**

---

Рассмотрен процесс установки программного продукта Microsoft Visual Studio.NET и кратко описаны компоненты.

## Глава 2

# Программирование в среде Visual Studio.NET

---

- Создание проекта
- Конфигурирование проекта
- Построение программы
- Отладка программы

Пакет Visual Studio – это интегрированное приложение, служащее основным интерфейсом доступа к инструментальным средствам системы программирования Visual C++. Его можно использовать для решения таких задач, как:

- управление программными проектами;
- создание и редактирование исходных файлов;
- конструирование таких ресурсов программы, как меню, диалоговые окна и значки;
- генерирование некоторых базовых фрагментов программы с помощью мастеров.

Пакет Visual Studio.NET позволяет провести разработку программы от начала и до подготовки готовой к распространению версии. Используя этот пакет, можно манипулировать программными константами и классами языка C++, а также обращаться к справочной системе Visual Studio и MSDN. В данной главе описаны основные возможности и инструментальные средства Visual Studio, кроме редакторов ресурсов и мастеров для создания графических приложений (часть III).

В качестве примера в этой главе создается простая *консольная* программа, работающая в текстовом режиме. Приобретенные знания позволят вам вводить и запускать примеры, а также создавать простые тестовые программы, основываясь на кратком введении в C++, приведенном в части II. Для получения дальнейших инструкций по созданию консольных программ пользуйтесь справочной системой.

## Создание проекта

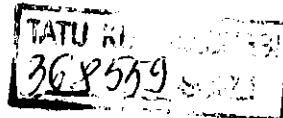
---

Для запуска Visual Studio, выберите команду Visual Studio.NET в меню Start операционной системы Windows (подменю Programs).

Как правило, первое действие в работе с пакетом Visual Studio – создание *проекта* программы, в котором будет храниться вся информация, необходимая для построения отдельной программы:

- имена и взаимосвязи исходных файлов;
- списки требуемых библиотечных файлов;
- параметры компилятора, компоновщика, а также других инструментов, использующихся при построении.

Напишем программу с названием Hello. Чтобы создать для нее проект, выполните следующие действия:

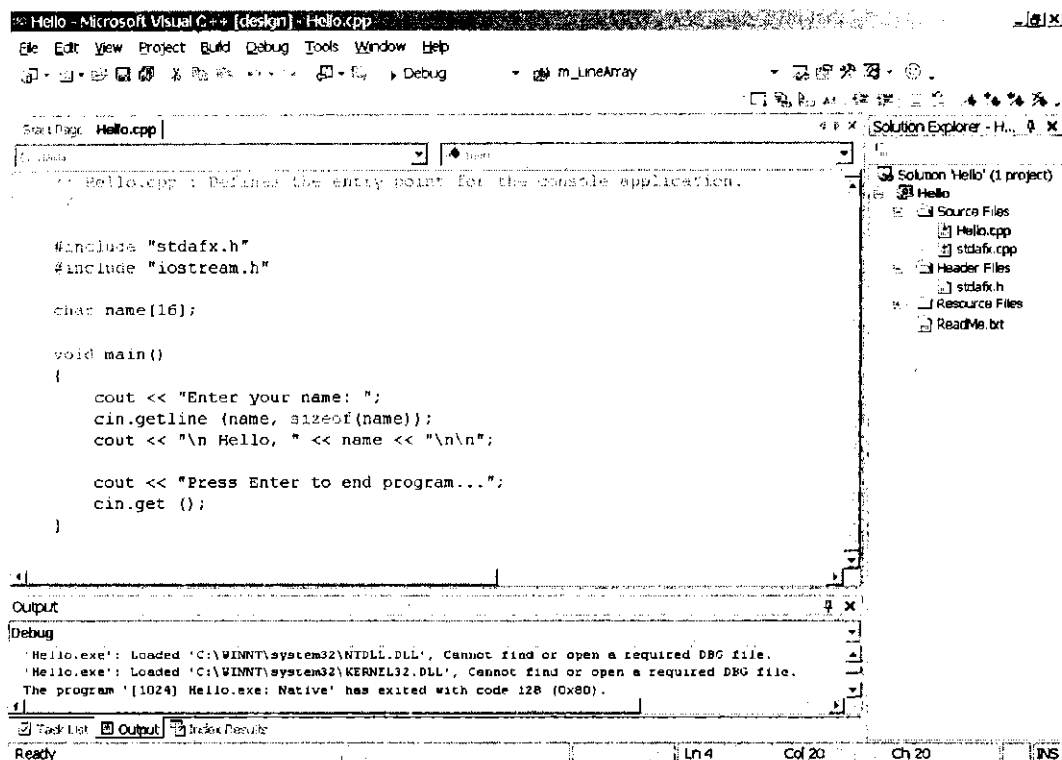


1. На панели инструментов Standard следует нажать кнопку New Project (если панель не отображается, ее отображение можно включить командой View/Toolbars/Standard).
2. В меню File необходимо выбрать команду New и в открывшемся подменю из двух пунктов выбрать Project... (данной команде соответствует сочетание клавиш Ctrl+Shift+N). При создании нового проекта пакет Visual Studio автоматически создает *рабочую область проекта* и добавляет в нее новый проект. В рабочей области могут храниться один или несколько отдельных проектов. Во всех примерах программ данной книги используется рабочая область с одним проектом. В более сложных разработках в рабочую область разрешается добавлять дополнительные проекты, работать с которыми можно одновременно.
3. Вышеописанные действия приведут к открытию диалогового окна New Project. В поле Project Types показана иерархия типов проектов Visual Studio. Выберите папку Visual C++ Projects. В результате в поле Templates отобразятся различные шаблоны проектов, доступные в VC7. В рассматриваемом примере необходимо выбрать шаблон типа Win32.

Шаблоны Win32 позволяют, в частности, создать 32-разрядное приложение, работающее в текстовом режиме. Подобно 16-разрядной программе MS-DOS, оно может выполняться либо в обычном окне Windows, либо во весь экран. Консольная программа пишется с использованием методов, используемых для написания текстовых программ в MS-DOS или UNIX. В частности, в таких программах можно выполнять вывод или ввод информации, применяя потоковые библиотечные функции, например, `printf` и `gets`, или (в программах языка C++) функции библиотеки классов `iostream`. В примерах программ во второй части книги, написанных как консольные приложения, используются функции `iostream`. Это сделано для того, чтобы вы могли сконцентрироваться на изучении языка C++, не отвлекаясь на сложности программирования графических приложений Windows. В части III с теми же целями создаются проекты типа "MFC Application" с использованием библиотеки MFC (Microsoft Foundation Classes) и мастеров Visual C++.

В операционных системах MS-DOS и Windows версий 3.x используются *16-разрядные программы*, т.е. они выполняются в таком режиме процессора, когда задействованы только 16-разрядные регистры. Следствие работы в этом режиме – 16-битная разрядность целых чисел, адресация ячеек памяти с использованием 16-разрядного адреса сегмента и 16-разрядного смещения. В целях совместимости в среде Windows 95 или Windows NT разрешается запускать 16-разрядные программы MS-DOS и Windows 3.1, хотя они не используют в полной мере преимущества системы. Программы Win32, разрабатываемые с помощью Visual C++.NET, являются *32-разрядными*, т.е. при их выполнении задействованы 32-разрядные регистры и используются 32-битная разрядность целых чисел, а также адресация ячеек памяти с *единым* 32-разрядным адресом. (Этот метод называется моделью *простой (flat)* адресации памяти. Теоретически он позволяет программе адресовать до 4 Гбайт виртуальной памяти.). В Visual C++ можно разрабатывать только 32-разрядные приложения. Для разработки 16-разрядных приложений (для Windows 3.1 или MS-DOS) необходима 16-разрядная версия Visual C++ или 16-разрядная среда разработки независимого поставщика.

4. В поле Name укажите имя проекта (в данном случае – Hello), в поле Location – рабочую папку создаваемого проекта. При необходимости рабочую папку можно выбрать кнопкой Browse...
5. Щелкните на кнопке OK. Visual Studio отобразит окно мастера Win32 Console Application. В окне мастера Win32 Console Application выберите опцию An empty project, что приведет к созданию проекта без добавления исходных файлов. Щелкните на кнопке Finish внизу диалогового окна Win32 Console Application.



Visual Studio создаст и откроет рабочую область проекта с именем Hello, содержащую единственный проект с таким же именем. Информация о нем отобразится в окнах Solution Explorer и Class View. Если вкладки Solution Explorer и Class View невидимы, то выберите соответствующие опции в меню View. При создании графических приложений (гл. 9) открывается дополнительная вкладка Resource View. Размещение окон и панелей инструментов в среде Visual Studio можно настраивать, перемещая окна просмотра информации, включая/выключая отображение панелей инструментов и т. д.

Для закрытия проекта Hello необходимо выбрать команду Close меню File. Открыть проект можно, выбрав файл "решения" проекта Hello (Hello.sln) в диалоговом окне Open. Если вы работали с проектом Hello недавно, то выберите соответствующий пункт в подменю Recent меню File для открытия данного проекта.

## Исходный файл программы

Следующим этапом разработки приложения в Visual Studio является редактирование исходного файла Hello.cpp, включенного в проект Hello. Введите в окне редактора исходный фрагмент программы (листинг 2.1).

### Листинг 2.1

```
// Hello.cpp: исходный файл программы Hello

#include "stdafx.h"
#include <iostream.h>

char Name [16];
```

```

void main ()
{
    cout << "Enter your name: ";
    cin.getline (Name, sizeof (Name));
    cout << "\nHello, " << Name << "\n\n";

    cout << "\nPress Enter to end program...";
    cin.get ();
}

```

Многие команды редактирования в Visual Studio вам, наверное, уже знакомы (особенно, если вы использовали другие текстовые редакторы или среды разработки приложений). Вообще сочетания клавиш, применяемые в редакторе Visual Studio для вызова команд, можно менять или выбирать в соответствии с одной из предложенных схем (схема по умолчанию, схема Visual C++ 6 и другие). При первом запуске Visual Studio это можно сделать на стартовой странице. В табл. 2.1 приведены наиболее важные сочетания клавиш, применяемые при редактировании. Эти сочетания стандартны, но их можно изменять. Полное описание команд редактирования, операций, выполняемых с помощью мыши, и способов настройки редактора представлено в справочной системе (использовалась схема сочетаний Visual C++ 6).

Табл. 2.1. Стандартные сочетания клавиш пакета Visual Studio

Перемещение курсора	Клавиша или сочетание клавиш
На один символ назад	←
На один символ вперед	→
На одну строку вверх	↑
На одну строку вниз	↓
На одно слово вперед	Ctrl+→
На одно слово назад	Ctrl+←
На первый символ строки	Home
В первый столбец	Home, затем снова Home
На один экран вверх	PgUp
На один экран вниз	PgDn
На один экран вправо	Ctrl+PgDn
На один экран влево	Ctrl+PgUp
В начало файла	Ctrl+Home
В конец файла	Ctrl+End
На дополняющую скобку	Поместите указатель перед скобкой и нажмите Ctrl+]
На нужную строку	Ctrl+G
Вверх на одну строку	Ctrl+↑
Вниз на одну строку	Ctrl+↓
Выделение	Сочетание клавиш
Текст	Shift+ одна из клавиш перемещения (или комбинация клавиш)
Столбцы текста	Ctrl+Shift+F8, чтобы включить режим выборки столбцов, затем одна из клавиш (или комбинация клавиш) перемещения



Табл. 2.1. (Продолжение)

<b>Выделение</b>	<b>Сочетание клавиш</b>
Отмена режима выделения столбцов	Снова Ctrl+Shift+F8 или Esc
Выборка всего файла	Ctrl+A
<b>Вырезание, копирование, вставка и удаление текста</b>	<b>Клавиша или сочетание клавиш</b>
Копировать выделенный текст в буфер	Ctrl+C
Копировать выделенный текст в буфер и удалить его из окна	Ctrl+X
Копировать текущую строку в буфер и удалить ее из окна	Ctrl+L
Вставить текст из буфера	Ctrl+V
Удалить выделенный текст	Del
Удалить символ справа от указателя	Del
Удалить символ слева от указателя	Backspace
Удалить слово слева от указателя	Ctrl+Backspace
Удалить текущую строку	Ctrl+Shift+L
<b>Табуляторы</b>	<b>Клавиша или сочетание клавиш</b>
Вставить символ табуляции (или пробелы)	Tab
Перейти к предыдущей позиции табуляции	Shift+Tab
Передвинуть несколько строк на одну позицию табуляции вправо	Выделить строки и затем нажать Tab
Передвинуть несколько строк на одну позицию табуляции влево	Выделить строки и затем нажать Shift+Tab
Отобразить символы табуляции (в виде >>) и пробелы (в виде точек)	Ctrl+ Shift+8
<b>Операции редактирования</b>	<b>Клавиша или сочетание клавиш</b>
Перевести выделенный текст в верхний регистр	Ctrl+Shift+U
Перевести выделенный текст в нижний регистр	Ctrl+U
Переключение режимов вставки и замены	Ins
Перестановка текущей и предыдущей строки	Alt+Shift+T
<b>Отмена/Восстановление</b>	<b>Сочетание клавиш</b>
Отменить предыдущую операцию редактирования	Ctrl+Z
Повторить предыдущую операцию редактирования	Ctrl+Y
<b>Поиск</b>	<b>Клавиша или сочетание клавиш</b>
Найти текст, используя окно Find или стандартную панель инструментов	Ctrl+D, введите текст, затем нажмите Enter или F3
Открыть диалоговое окно Find	Ctrl+F
Найти следующее вхождение	F3
Найти предыдущее вхождение	Shift+F3

Табл. 2.1. (Окончание)

Поиск	Клавиша или сочетание клавиш
Найти следующее вхождение выделенного текста	Ctrl+F3
Найти предыдущее вхождение выделенного текста	Ctrl+Shift+F3
Выполнить поиск в направлении вперед	Нажмите Ctrl+I с последующим вводом искомого текста
Выполнить поиск в направлении назад	Нажмите Ctrl+Shift+I с последующим вводом искомого текста
Найти следующую ошибку построения программы	F4
Найти предыдущую ошибку построения программы	Shift+F4
На дополняющую скобку	Поместите указатель перед скобкой и нажмите Ctrl+]
Закладки	Клавиша или сочетание клавиш
Включить/выключить закладку	Ctrl+F2
Перейти к следующей закладке	F2
Перейти к предыдущей закладке	Shift+F2
Снять все закладки	Ctrl+Shift+F2
Открыть диалоговое окно Open Bookmark, чтобы создать, удалить или открыть закладку	Alt+F2
Окна	Сочетания клавиш
Перейти в следующее окно	Ctrl+F6
Перейти в предыдущее окно	Ctrl+Shift+F6
Закрыть активное окно	Ctrl+F4

Завершив ввод текста программы, сохраните файл, выбрав команду Save в меню File или щелкнув на кнопке Save в панели инструментов Standard.

Приведенная выше программа Hello использует библиотеку `iostream`, которая содержит исчерпывающий набор предопределенных классов, применяемых в консольных программах для выполнения основных операций ввода и вывода. Классы языка C++ рассматриваются в гл. 4, а библиотека `iostream` описана в справочной системе. Программа содержит файл заголовков `iostream.h` с объявлениями, необходимыми для использования библиотеки `iostream`. Для отображения строки "Enter your name: " в окне консоли используется оператор

```
cout << "Enter your name: ";
```

Это выражение генерирует выводимую информацию, используя объект `cout`; (объекты рассматриваются в гл. 4). Оператор `<<` здесь *перегружен*. В данном контексте это означает, что оператор имеет некоторые особые функции. Перегруженные операторы описаны в гл. 6.

Вводимые пользователем символы читает и сохраняет в массиве `Name` оператор

```
cin.getline (Name, sizeof (Name));
```

Первый параметр, передаваемый в функцию `getline`, задает имя принимающего буфера, а второй – размер этого буфера; `getline` называют *функцией-членом* объекта `cin` (функции-члены рассмотрены в гл. 4).

Текст Hello отображается в начале новой строки окна оператором

```
cout << "\nHello, " << Name << "\n\n";
```

За этим текстом отображаются символы, введенные пользователем и сохраненные в массиве Name. Данный оператор использует объект cout.

Завершающие программу операторы генерируют паузу, длящуюся до тех пор, пока пользователь не нажмет клавишу Enter. Функция get читает один символ. Функции get и getline ожидают нажатия клавиши Enter для завершения ввода.

```
cout << "\nPress Enter to end program...";  
cin.get ();
```

Запрограммированная перед нажатием клавиши Enter в конце программы пауза позволяет пользователю просмотреть выведенную программой информацию до закрытия окна программы. Если программа выполняется в среде Visual Studio, то в этом нет необходимости, так как после завершения программы окно остается открытым до нажатия любой клавиши. Однако пауза необходима, если программа выполняется вне среды Visual Studio или под управлением отладчика Visual Studio, потому что в этих случаях при выходе из программы ее окно автоматически закрывается.

## ***Вкладки Solution Explorer и Class View***

Щелчок на ярлычке вкладки Solution Explorer отображает в виде дерева:

- имя рабочей области (в рассматриваемом примере Hello);
- имя проекта (в рассматриваемом примере Hello);
- имена исходных файлов, принадлежащих этому проекту (в рассматриваемом примере Hello.cpp, Stdafx.cpp), файлов заголовков (Stdafx.h) и некоторых других (ресурсных и вспомогательных).

Чтобы увидеть имя исходного файла, необходимо развернуть соответствующую ветвь. Можно развернуть (или свернуть) ветви дерева, щелкнув на символе "+" (или "-") слева от узла ветви. Можно открыть исходный файл (или отобразить его, если он уже открыт), выполнив двойной щелчок на его имени в дереве. Вкладка Solution Explorer наиболее полезна для сложных проектов, в которые входит несколько исходных файлов (см. примеры таких проектов в следующих главах книги).

Щелчок на ярлычке вкладки Class View отобразит дерево, в котором перечислены все глобальные символические обозначения, определенные в программе Hello. Если выполнить двойной щелчок на символе, представленном на вкладке ClassView, то Visual Studio отобразит исходный файл, в котором определено это обозначение, и поместит указатель на его определении. Для программ, содержащих классы, вкладка ClassView отображает все определенные в программе классы и их члены.

## ***Справочная система***

Установка справочной системы MSDN описывалась в гл. 1. В Visual Studio справочная система вызывается в отдельном окне программы. Чтобы найти один из разделов в Visual Studio, выберите в меню Help команду Contents. При этом откроется окно справочной системы и отобразится вкладка Contents. Во вкладке Contents разделы справочной системы представлены в виде дерева, подобно вкладкам Solution Explorer и Class View. Чтобы развернуть или свернуть ветвь дерева, щелкните на символе "+" или "-" слева от узла ветви. Узлы самого нижнего уровня представлены разделами (они не имеют символов "+" или "-"). Если выполнить двойной щелчок на одном из этих узлов, то внутри панели в правой части окна справочной системы появится соответствующий текст справки. Другой способ найти нужную информацию – выбрать пункт Index в меню Help и найти информацию по ключевому слову. В общем, техника работы со справкой в Visual Studio не отличается от работы со справочными системами других приложений Microsoft.

## Конфигурирование проекта

---

Для создаваемого проекта можно выбрать одну из двух создаваемых изначально конфигураций или создать собственную. Создаваемые по умолчанию конфигурации:

- Win32 Debug, генерирующую *отладочную* версию программы. Отладочная конфигурация содержит установки, позволяющие отлаживать программу при помощи встроенного отладчика, и обычно используется при разработке и тестировании программы.
- Win32 Release, генерирующую *выходную* версию программы. Финальная конфигурация содержит установки для оптимизированной версии программы.

Обратите внимание: можно добавить или удалить конфигурацию, выбирая команду Configurations Manager... в меню Build. Можно также изменить установки отдельной конфигурации, выбрав ее название в окне Configurations Manager.

Приятным следствием генерирования 32-разрядных программ является то, что не нужно беспокоиться о модели памяти. Все указатели являются 32-разрядными значениями. Нет необходимости в различных моделях адресации памяти, а также не нужно разделять указатели на типы *near* или *far*.

## Построение программы

---

Следующий этап должен быть посвящен *построению отладочной версии программы* Hello. Если проект Hello еще не открыт, откройте его с помощью команды Open... из меню File. При получении команды построения Visual Studio всегда строит программу, используя *активную* конфигурацию проекта. Чтобы построить программу, выберите команду Build в меню Build или нажмите клавиши Ctrl+Shift+B. Если программа состоит из нескольких исходных файлов, то Visual Studio обрабатывает только файлы, измененные с момента последнего построения программы. Кроме того, можно заставить Visual Studio перестроить *все* файлы проекта, выбрав команду Rebuild All в меню Build.

В ходе выполнения процедуры построения программы Visual Studio отображает результаты каждого шага построения в окне Output. Если оно невидимо, то выберите команду Output в меню View. Чтобы увидеть результат построения, необходимо открыть вкладку Build в окне Output (другие вкладки позволят увидеть результаты работы других инструментов разработки). В окне Output будут появляться все сообщения об ошибках или предупреждения, а затем, когда построение завершится (успешно или неуспешно), Visual Studio выведет общее количество ошибок и предупреждений. Если в окне Output отображаются сообщения об ошибках или предупреждения, то вы можете переместиться на строку исходного текста, вызвавшую проблему, выполнив двойной щелчок на строке, содержащей сообщение. Нажмите клавишу F4 для просмотра *следующей* ошибки, встретившейся при построении программы, или Shift+F4 для просмотра *предыдущей* ошибки. Чтобы поэкспериментировать с этими средствами, можете специально внести ошибку в файл Hello.cpp и перестроить программу.

Для отладочной конфигурации при построении программы создается подкаталог \Hello\Debug для сохранения выходных файлов проекта (Hello.obj, Hello.exe и других). При построении выходной конфигурации создается отдельный подкаталог \Hello\Release для сохранения выходных файлов данной версии программы. Следовательно, файлы различных конфигураций программы не перезаписывают друг друга.

## Отладка программы

---

К числу наиболее полезных утилит пакета Visual Studio относится встроенный отладчик, позволяющий тестировать программы в процессе разработки. В этом параграфе рассматривается простое упражнение для первоначального ознакомления со средствами отладчика. В табл. 2.2 перечислены

наиболее важные сочетания клавиш отладчика. Полное описание всех команд и функций приведено в справочной системе.

Табл. 2.2. Сочетания клавиш отладчика Visual Studio

Действие	Сочетание клавиш
Добавить или удалить точку останова в строке с курсором	F9
Удалить все точки останова	Ctrl+Shift+F9
Начать выполнение программы или возобновить его с текущего оператора	F5
Повторить выполнение программы с самого начала	Ctrl+Shift+F5
Выполнить следующий оператор, в том числе операторы внутри функции (режим step into)	F11
Выполнение до точки останова (следующего оператора, включая вызываемые операторы) (режим step over)	F10
Выполнить программу до момента достижения первого оператора за пределами текущей функции (режим step out)	Shift+F11
Выполнить программу до позиции курсора (перейти к курсору)	Ctrl+F10
Перейти к позиции курсора без выполнения промежуточных операторов	Ctrl+Shift+F10
Открыть диалоговое окно QuickWatch для быстрого просмотра или изменения переменной или выражения	Shift+F9
Открыть диалоговое окно Breakpoints для размещения точек останова	Ctrl+B
Конец отладки	Shift+F5

Рассмотрим технику работы с отладчиком на простом примере.

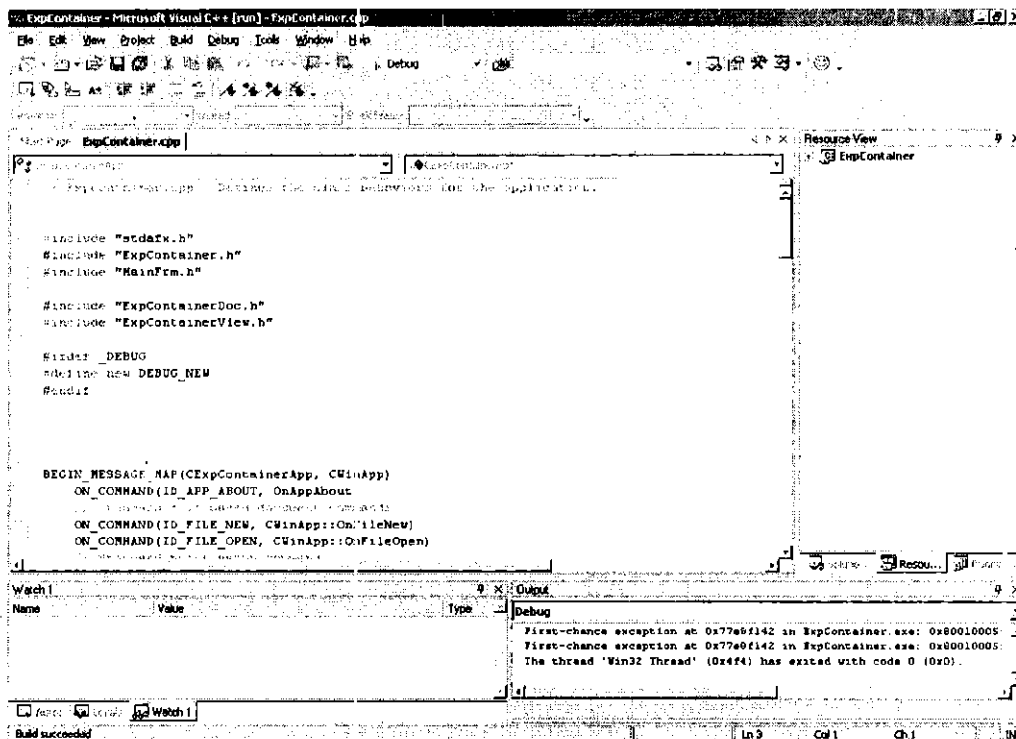
1. Откройте проект Hello, если он еще не открыт.
2. В качестве активной конфигурации проекта выберите конфигурацию Win32 Debug.
3. Выполните открытие исходного файла Hello.cpp, дважды щелкнув на имени файла Hello.cpp во вкладке Solution Explorer. Заметьте: вкладка Solution Explorer облегчает доступ к исходным файлам программы. Исходный файл можно открыть и традиционным способом, выбрав команду Open... в меню File или щелкнув в панели инструментов Standard на кнопке Open.
4. Щелчком мыши поместите курсор на первую строку функции main в файле Hello.cpp

```
cout << "enter your name: ";
```

и нажмите клавишу F9. В строке создается *точка останова* и при запуске в отладчике программа остановится непосредственно перед выполнением этой строки. Чтобы обозначить точку останова, Visual C++ отображает точку слева от строки. По умолчанию цвет этой точки красный, хотя его можно изменить, выбрав команду Options... в меню Tools. Если курсор остается в строке, то точка останова будет удалена при повторном нажатии клавиши F9.

5. Нажмите клавишу F5, чтобы начать выполнение программы в отладчике. На экране появится окно программы Hello. Однако ее выполнение остановится непосредственно перед строкой, содержащей точку останова. Управление будет возвращено в отладчик, а окно Visual Studio станет активным (в зависимости от упорядочения окон на экране окно Visual Studio может частично или полностью перекрыть окно программы Hello). Теперь точка останова отмечена стрелкой, означающей, что это *следующая* строка, которая будет выполнена. Следующую выполняемую строку называют *текущей* (стандартный цвет стрелки желтый).
6. Откройте окно наблюдения за переменными Watch. Обычно оно открывается автоматически, но если оно отсутствует на экране, выберите команду Watch в меню View.

7. Следите за содержимым переменной Name (глобальным символьным массивом, определенным в файле Hello.cpp). Для этого щелкните кнопкой мыши в прямоугольной области вверху окна Watch, введите слово Name и нажмите клавишу Enter. Окно Watch отображает адрес переменной, но не значения элементов массива. Для их просмотра щелкните на символе "+" слева от слова Name в окне Watch (или поместите курсор в верхней строке и нажмите Enter) – отобразятся элементы массива. Обратите внимание: поскольку Name является глобальной переменной, она инициализируется значением 0.



8. Для выполнения текущей строки нажмите клавишу F10. Текст, отображаемый в этой строке ("Enter your name: "), не появится в окне Hello, потому что он помещен в буфер и не будет отображаться до тех пор, пока не выполнится оператор ввода во второй строке. Заметьте: теперь становится текущей вторая строка.
9. Чтобы выполнить вторую строку программы, нажмите клавишу F10 или щелкните на кнопке Step Over. При выполнении этой строки (cin.get (Name, sizeof (Name));) необходимо ввести имя. Возобновить выполнение программы Hello из отладчика нельзя, пока не завершится выполнение этой строки. Поэтому необходимо переключиться на окно программы Hello, ввести какой-нибудь текст и нажать клавишу Enter. Таким образом управление возвращается отладчику, а текущей строкой становится третья строка примера.
10. Теперь переменная Name в окне Watch содержит только что введенную строку. Окно Watch позволяет изменить значение переменной. Для этого выполните двойной щелчок на одном из элементов массива, введите новое число и нажмите клавишу Enter.
11. Нажмите клавишу F5, чтобы возобновить выполнение программы. Поскольку другие точки останова отсутствуют, программа нормально завершит выполнение и отобразит в окне Hello строку Name с внесенными изменениями.

12. После нажатия клавиши Enter управление передается отладчику, который выводит сообщение о завершении программы. Это сообщение, как и предыдущие, появляется во вкладке Debug окна Output. Если окно Output невидимо, выберите команду Output в меню View.

Завершив разработку, отладку и проверку программы, обычно генерируют ее *выходную* версию, оптимизированную и, как правило, имеющую меньший размер. Выберите команду Set Active Configuration из меню Build, а затем отметьте пункт *Проект – Win32 Release* (*Проект* – это имя проекта) и щелкните на кнопке OK. Затем постройте программу с использованием этой конфигурации, выбрав команду Build в меню Build (*Программа* – это имя исполняемого файла программы) или нажав клавиши Ctrl+Shift+B.

Готовая программа запускается вне среды Visual Studio (исполняемый модуль запускается как любое приложение Windows).

## Резюме

---

Эта глава посвящена использованию Visual Studio для решения следующих задач:

- получение доступа к *справочной системе*;
- *создание проекта*, который хранит всю информацию, необходимую для построения программы;
- *добавление исходного файла в проект и редактирование* исходного файла программы;
- *просмотр файлов* и символических констант с использованием вкладок Solution Explorer и Class View;
- *выбор конфигурации*, влияющей на способ построения программы;
- *построение* программы;
- *отладка* программы.

# Часть II

## Введение в C++





## Глава 3

### От С к С++

---

- От С к С++
- Новинки С++

В этой главе рассмотрены методы преобразования программ на языке С в программы на С++. Она предназначена для пользователей, переходящих от программирования на языке С к программированию на С++. Приводятся сведения, которые помогут изменить навыки программирования на языке С и приобрести новые, более полезные для работы на С++. Показано, как можно воспользоваться преимуществами некоторых новых удобных средств С++. В этой главе мы подходим к изучению классов языка С++ и средств, требующих их использования. (Подробности см. в гл. 4, которая поможет эффективно использовать классы и изменять структуру программ). Изложенное в данной главе позволит вам быстро начать работу с языком С++, используя его как усовершенствованную версию языка С. Материал данной книги предполагает наличие у читателя базовых знаний о языке С. (Если вам нужно изучить язык С, воспользуйтесь списком литературы, приведенным во введении.) В книге рассмотрены особенности языка С++, не свойственные С, а концепции языка С++ часто объясняются с позиций соответствующих концепций С. Хотя описанные методы применимы как для программ с использованием графических средств пользовательского интерфейса (GUI) Windows, так и для программ, исполняемых из командной строки (консольные программы) Windows, ради простоты примеры написаны в виде консольных программ. Если вы хотите провести эксперимент со специальными компонентами языка С++, то можете запустить приложение Visual C++ Developer Studio и открыть проект Hello (гл. 2). Затем можно ввести код примера или собственный код в исходный файл Hello.cpp и построить программу (гл. 2). Если у вас имеется собственная программа на языке С, возможно она же будет вашей первой программой на С++!

### От С к С++

---

Кроме некоторых исключений, язык С является своеобразным подмножеством С++, т.е. С++ поддерживает почти все компоненты С, *дополняя их* многими собственными компонентами. Следовательно, начиная работать с С++, можно компилировать программы на С, как и на С++ с помощью компилятора С++ вместо С. Затем можно постепенно добавлять в программы различные компоненты, характерные для С++. Обычно файлы программ на языке С++ имеют окончания .cpp или .cxx (а не .c). Однако существуют приемы программирования, допустимые во многих версиях С, которые не будут компилироваться с помощью С++ или будут компилироваться, однако результирующий код будет выполнять другие операции. В языке С есть ряд конструкций, которые не поддерживаются в С++.

- Число зарезервированных ключевых слов в языке С++ намного больше, чем в С. В программах их *нельзя* использовать как идентификаторы. Ниже приведен список зарезервированных ключевых слов, используемых в системе Visual C++.

<code>_abstract (2)</code>	<code>_alignof</code>	<code>_asm</code>
<code>_assume</code>	<code>_based</code>	<code>_box (2)</code>
<code>_cdecl</code>	<code>_declspec</code>	<code>_delegate (2)</code>
<code>_event</code>	<code>_except</code>	<code>_fastcall</code>
<code>_finally</code>	<code>_forceinline</code>	<code>_gc (2)</code>
<code>_hook (2)</code>	<code>_identifier</code>	<code>_if exists</code>

<u>_if not exists</u>	<u>_inline</u>	<u>_int8</u>
<u>_int16</u>	<u>_int32</u>	<u>_int64</u>
<u>_interface</u>	<u>_leave</u>	<u>_m64</u>
<u>_m128</u>	<u>_m128d</u>	<u>_m128i</u>
<u>_multiple_inheritance</u>	<u>_nogc (2)</u>	<u>_noop</u>
<u>_pin (2)</u>	<u>_property (2)</u>	<u>_raise</u>
<u>_sealed (2)</u>	<u>_single_inheritance</u>	<u>_stdcall</u>
<u>_super</u>	<u>_try_cast (2)</u>	<u>_try/ _except</u>
<u>_try/ _finally</u>	<u>_unhook (2)</u>	<u>_uuidof</u>
<u>_value (2)</u>	<u>_virtual_inheritance</u>	<u>_w64</u>
bool	break	case
catch	char	class
const	const_cast	continue
default	delete	deprecated (1)
dllexport (1)	dllimport (1)	do
double	dynamic_cast	else
enum	explicit	extern
false	float	for
friend	goto	if
inline	int	long
mutable	naked (1)	namespace
new	noinline (1)	noreturn (1)
nothrow (1)	novtable (1)	operator
private	property (1)	protected
public	register	reinterpret_cast
return	selectany (1)	short
signed	sizeof	static
static_cast	struct	switch
template	this	thread (1)
throw	true	try
typedef	typeid	typename
union	unsigned	using
uuid (1)	virtual	void
volatile	wchar_t	while

Ключевые слова с пометкой (1) представляют собой дополнительные атрибуты ключевого слова `__declspec`, с пометкой (2) – применяются только в расширениях к C++. Ключевые слова в начале приведенного списка, начинающиеся со знака подчеркивания ( ) являются зарезервированными в VC7.

- В C++ определение или объявление функции *должно* предшествовать любому вызову функции внутри исходного файла (в противном случае компилятор сгенерирует ошибку *необъявленный идентификатор*). В языке C разрешен вызов функции перед ее объявлением или определением (в этом случае генерируется предупреждение). В этой книге термин *объявление функции* относится к объявлению, описывающему только имя функции, типы возвращаемого значения и параметров, а *определение функции* – к объявлению, содержащему полный код функции. Заметьте, что в описании этих терминов может быть использован общий термин *объявление*, связывающий имя функции с типом возвращаемого ею значения.
- Объявление функции, использующей один или более параметров, в языке C можно выполнить, предварительно их не перечисляя, например:

```
int FuncX (); /* эта функция языка C на самом деле будет
использовать несколько параметров */
```

В языке C++ такое объявление допустимо только для функций, которые *не* имеют параметров.

- В языке C++ *недопустимо* использование “старого стиля” синтаксиса определения функции, в котором типы параметров *следуют* за списком параметров, например:

```
int FuncX (A, B); /* правильно в C, ошибка в C++ */
int A;
int B;
{
    /* код FuncX */
    return 0;
}
```

- В языке C++ для определения переменной можно объявить глобальный объект данных без спецификатора `extern` только один раз. Чтобы сделать переменную доступной в определенной *области видимости* (т.е. в рамках фрагмента текста программы, в котором используется эта переменная), все другие объявления элемента должны включать ключевое слово `extern`.
- В соответствии со стандартом ANSI в языке C можно присвоить указатель типа `void` указателю *любого* типа. Например, следующий код задает присваивание указателю типа `int` указателя типа `void`:

```
int A;
int *PInt;
void *PVoid = &A;
```

```
PInt = PVoid; /* правильно для C; ошибка для C++ */
```

Компилятор C++ не будет автоматически преобразовывать указатель типа `void` в указатель другого типа в выражении присваивания. Для выполнения присваивания используется операция изменения типа.

```
PInt = (int *)PVoid; /* правильно в C и C++ */
```

- Названия тегов конструкций `enum`, `struct` или `union` в языке C могут совпадать с названиями типов данных, определенными оператором `typedef`, в пределах одной области видимости. Например, следующий код успешно компилируется в C:

```
/* этот код правильный в C, но ошибочный в C++: */
```

```
typedef int TypeA;
struct TypeA
{
    int I;
    char C;
    double D;
};
```

```
typedef int TypeB;
union TypeB
{
    int I;
    char C;
    double D;
};
```

```
typedef int TypeC;
enum TypeC (Red, Green, Blue);
```

Компилятор языка C может различать дублирующие имена, потому что при описании перечислений, структур и объединений используются префиксы `enum`, `struct` или `union`, например:

```
/* этот код правилен в C, но ошибочен в C++: */

typedef int TypeA;
struct TypeA
{
    int I;
    char C;
    double D;
};

TypeA X;          /* создает переменную типа int */
struct TypeA Y;   /* создает структуру TypeA */

sizeof (TypeA);   /* вычисляет размер переменной типа int */
sizeof (struct TypeA); /* вычисляет размер структуры TypeA */
```

Теги конструкций `enum`, `struct`, `union` или `class` в языке C++ (здесь их чаще называют *именами*, а не *тегами*) должны *отличаться* от любого имени, заданного в операторе `typedef` внутри той же области видимости. (Ключевое слово `class` уникально для языка C++. Классы рассмотрены в гл. 4.) Определение создаст новый тип, на который можно сослаться, используя одно имя этого типа, поэтому не рекомендуется использовать префикс `enum`, `struct`, `union` или `class` (хотя при желании это можно делать). Описанное свойство продемонстрировано следующим фрагментом программы:

```
/* этот код ошибочен в C, но правилен в C++: */
struct TypeA
{
    int I;
    char C;
    double D;
};

struct TypeA X;      /* создает структуру TypeA */
TypeA Y;             /* также создает структуру TypeA */

sizeof (struct TypeA); /* вычисляет размер структуры TypeA */
sizeof (TypeA);       /* вычисляет размер структуры TypeA */
```

Если задать оператор `typedef` с таким же именем, то компилятор C++ не всегда сможет различить эти два типа, например:

```
typedef int TypeA;
struct TypeA          /* ошибка в C++: повторное определение TypeA */
{
    int I;
    char C;
    double D;
};

TypeA X;              /* создаются при этом данные типа int
                       или структура TypeA? */
sizeof (TypeA);       /* вычисляется размер данных типа int
                       или размер структуры TypeA ?*/
```

- В языке C++ имя `typedef` должно быть скрыто. В языке C, если теги `enum`, `struct` или `union` такие же, как имя, определенное `typedef` во внешней области видимости, то вы все еще

можете оставить ссылку на `typedef` во внутренней области видимости (т. е. имя, определенное `typedef`, не скрыто). Следующий фрагмент программы иллюстрирует различие.

```
typedef int TypeA;

void main ()
{
    struct TypeA
    {
        int I;
        char C;
        double D;
    };

    TypeA X;          /* в языке C идентификатор X имеет тип int;
                       в C++ X - это структура TypeA */

    sizeof (TypeA);    /* в C - вычисляет размер данных типа int;
                       в C++ - вычисляет размер структуры TypeA */
}
```

- В языке C `sizeof('x')` равно `sizeof(int)`, а в C++ оно равно `sizeof(char)`. Если определено перечисление

```
enum E {X, Y, Z};
```

Тогда в C значение `sizeof (X)` будет равно `sizeof (int)`, а в C++ равно `sizeof (E)`, что не обязательно равно `sizeof (int)`.

#### Примечание

Из существующей программы на C можно удалить любую из несовместимых с C++ конструкций так, что код будет компилироваться и иметь такое же смысловое содержание, как и при использовании компилятора C++. Компилятор C++ заметит все перечисленные варианты несовместимости, кроме двух последних, которые (особенно различия в значениях, создаваемых оператором `sizeof`) не приводят к появлению сообщений компилятора об ошибке, но *изменяют результирующий код и результат выполнения программы*. Такая ситуация – исключение из общего правила, в соответствии с которым: если программа успешно компилируется компиляторами C и C++, то она имеет в обоих языках одинаковое смысловое содержание.

## Новинки C++

---

Этот раздел посвящен новым средствам языка C++ (классы рассмотрены отдельно в следующей главе). Некоторые из этих компонентов включены в последние версии языка C (в частности, новый стиль комментариев, встроенные (`inline`) функции и типы констант, хотя они могут синтаксически отличаться от представления в языке C++).

### Оформление комментариев

В программе C++ наряду со стандартными ограничителями комментария (`/*` и `*/`) можно задать комментарий-строку, начинающуюся двумя символами `//`. Все символы, следующие за символами `// до конца строки`, являются частью комментария и пропускаются компилятором.

```

void main () // это комментарий-строка
{
    /* по-прежнему можно пользоваться традиционными ограничителями
    комментариев, удобными для задания комментария, занимающего
    более одной строки ... */
    // операторы ...
}

```

## Объявление переменных

Локальная переменная (т.е. переменная, определенная внутри функции) в языке С должна объявляться в начале блока перед другими операторами программы. Однако в языке С++ объявление локальной переменной рассматривается как обычный оператор программы, следовательно объявление можно поместить в любом месте программы перед первой ссылкой на переменную. Фрагмент кода, ограниченный символами { и }, называется блоком. На переменную, определенную внутри блока, можно ссылаться только внутри этого или внутри вложенного блоков (если она не скрыта переменной с таким же именем внутри вложенного блока). Таким образом, блок определяет область видимости переменной.

Код в языке С++ можно сделать более легким для чтения и сопровождения, помещая определение переменной непосредственно перед кодом, который ее использует, например:

```

void main ()
{
    // другие операторы ...

    int Count = 0;
    while (++Count <= 100)
    {
        // операторы цикла ...
    }
    // другие операторы ...
}

```

Локальную переменную можно объявить даже *внутри* цикла for. В приведенном ниже коде считается, что объявление счетчика цикла i встречается за пределами блока, следующего непосредственно за оператором for. Следовательно, ссылаться на переменную счетчика можно и после блока for, а переменная Length, объявленная *внутри* блока for, доступна только внутри этого блока.

```

// другие операторы ...

for (int i = 0; i < MAXLINES; ++i)
{
    // другие операторы ...

    int Length = GetLineLength (i);
    if (Length == 0)
        break;

    // другие операторы ...
}

if (i < MAXLINES)
    // встретила строка длиной 0

```

Элемент данных можно инициализировать при выполнении оператора объявления, включив инициализацию в объявление. Так, в приведенном выше примере:

- переменная `i` создается и инициализируется *однократно*, когда программное управление достигает оператора `for`;
- переменная `Length` создается и инициализируется на каждой итерации цикла.

Локальная переменная, объявленная как статическая (ключевое слово `static`), создается и инициализируется только один раз, когда оператор объявления переменной отрабатывается *впервые* при выполнении кода.

Если объявление переменной содержит инициализацию, то необходимо убедиться, не пропущен ли оператор объявления, как в следующей конструкции `switch`, компиляция которой вызывает сообщение об ошибке. Компилятор генерирует ошибку по следующей причине: переменная `Count` инициализируется только тогда, когда управление достигает ветви `case 2` оператора `switch`. Если управление передается ветви `case 3`, содержащей обращение к переменной `Count`, то она не будет инициализирована. Ветвь `case 3` имеет доступ к переменной `Count`, так как объявление переменной предшествует данной ветви внутри того же блока. Компилятор сгенерирует ошибку, даже если код в этой ветви действительно не использует переменную `Count`. Важен факт, что использование возможно.

```
switch (Code)
{
    case 1:
        // ...
        break;

    case 2:
        int Count = 0; // ошибка
        // ...
        break;

    case 3:
        // ...
        break;
}
```

Для исправления ошибки нужно поместить код ветви `case 2` внутри ее собственного блока. В таком варианте оператора `switch` сослаться на переменную `Count` можно только внутри ветви `case 2`.

```
case 2:
{
    int Count = 0;
    // ...
    break;
}
```

А можно просто убрать инициализацию из объявления и присвоить переменной `Count` начальное значение.

```
case 2:
    int Count;
    Count = 0;
    // ...
    break;
```

## Расширение области видимости

Если объявить внутри некоторой области видимости переменную с таким же именем, как у переменной во внешней области, переменная во внутренней области *скрывает* переменную, находящуюся во внешней области видимости. Таким образом, внутри области видимости имя переменной будет содержать ссылку на другой объект.

```
double A;

void main ()
{
    int A;
    A = 5;          // присваивает значение 5 переменной A типа int
}
```

В этом примере переменная A типа `int`, объявленная внутри функции `main`, скрывает переменную A типа `double`, объявленную снаружи, и присваивание внутри функции изменяет значение переменной A типа `int`, но не типа `double`. Однако в языке C++ можно организовать доступ к глобальной переменной (т.е., переменной, объявленной снаружи любой функции), даже если она скрыта локальной с аналогичным именем. Для этого перед именем переменной указывается *оператор расширения области видимости* `::`. например:

```
double A;          // глобальная переменная A

void main ()
{
    int A;          // локальная переменная A

    A = 5;           // присваивает значение 5 локальной переменной A
                     // типа int

    ::A = 2.5;       // присваивает значение 2,5 глобальной
                     // переменной A типа double
}
```

Операция расширения области видимости используется только для организации доступа к глобальной переменной. Данную операцию *нельзя* использовать для доступа к локальной переменной:

```
void FuncX (int Code)
{
    double A;

    if (Code == 1)
    {
        int A;

        // попытка доступа к переменной A типа double во внешнем блоке:
        ::A = 2.5; // ОШИБКА: ::A описывает глобальную
                  // переменную A, что неправильно,
                  // если A не объявлена на глобальном уровне
        //...
    }
    //...
}
```



Расширение области видимости можно также использовать для доступа к классам (см. две следующие главы), а также к глобальной переменной или перечислению, скрытым локальной переменной или перечислением с аналогичными именами. Например:

```
#include <iostream.h>

typedef int A;
enum (Red, White, Blue);

void main ()
{
    typedef double A;
    enum (Blue, White, Red);
    cout << sizeof (A) << '\n';           // печатает размер переменной
                                           // A типа double
    cout << sizeof (::A) << '\n';         // печатает размер переменной
                                           // A типа int
    cout << (int)Blue << '\n';           // печатает 0
    cout << (int)::Blue << '\n';         // печатает 2
}
```

## Встроенные функции

Замена вызовов функции копией кода функции называется *макрорасширением (inline expansion)* или просто *встраиванием (inlining)*. Если объявить функцию с использованием ключевого слова `inline`, как это сделано в приведенном ниже примере, компилятор попытается заменить все ее вызовы фактическим кодом функции. Замена типа функции на `inline` не изменяет ее смысла, но увеличивает скорость выполнения программы и размер результирующего кода. Поэтому желательно использовать спецификатор `inline` при определении *маленькой* функции, вызываемой из небольшого числа мест в коде, особенно в случае ее многократного вызова в цикле.

```
inline int FuncA (int X)
{
    // код функции ...
}
```

Использование директивы `inline`, которая эквивалентна специфической директиве Microsoft `__inline`, *не гарантирует*, что функция будет реализована как встроенная. В некоторых ситуациях компилятору может понадобиться сгенерировать обычный вызов функции, например, если она рекурсивная (т.е. вызывающая саму себя) или вызывается указателем функции. Можно с помощью компилятора выполнить встраивание функции (если это возможно), используя директиву Microsoft `__forceinline`. Дополнительную информацию по использованию этих директив можно найти в справочной системе *Visual C++*. Описание встроенных функций, являющихся членами классов, приведено в гл. 4.

Для выполнения встраивания компилятор должен иметь прямой доступ к коду функции. Определение встроенной функции (т.е. объявление, содержащее ее полный код) должно быть представлено в каждом исходном файле, в котором вызывается данная функция. Чтобы в каждом модуле, содержащем вызовы встроенной функции, были доступны идентичные копии определения функции, необходимо определить функцию в файле заголовков, вызываемом каждым исходным файлом. При изменении встроенной функции должны быть перекомпилированы все исходные файлы, содержащие ее вызовы.

Можно *настроить* способ обработки встроенных функций для проекта Microsoft Visual C++, выбрав в меню Project команду Settings... (см. гл. 2). В диалоговом окне Project Settings выберите вкладку C/C++ и выберите пункт Optimizations в списке Category. Затем задайте требуемый параметр

встроенной функции в списке *Inline Functions Expansions*. (Изменения в данном окне касаются только текущего проекта). Описание различных параметров смотрите в справочной системе.

Реализация встраивания напоминает работу макросов, определенных с использованием директивы препроцессора `#define`. Например, следующая функция, предназначенная для возвращения абсолютной величины целого числа

```
inline int Abs (int N) {return N < 0 ? -N : N;}
```

подобна макросу

```
#define ABS (N) ((N) < 0 ? -(N) : (N));
```

Однако, есть и отличие: вызовы встроенной функции обрабатываются компилятором, а макросов – препроцессором, выполняющим простую текстовую подстановку. Следовательно, встроенная функция имеет два важных преимущества перед макросом. В данном примере:

- Во-первых, при вызове функции *компилятор проверяет тип передаваемых ей параметров*, чтобы убедиться в том, что они целочисленные или их значения можно преобразовать в целочисленные.
- Во-вторых, если функции передается выражение, то оно *вычисляется всего один раз*. Если же выражение передается в макрос, то оно может вычисляться несколько раз (в приведенном примере дважды), что может привести к неожиданным результатам. Например, следующий вызов макроса дважды декрементирует переменную `I` (возможен неожиданный или нежелательный результат!):

```
Abs (--I);
```

У макроса есть еще одно преимущество перед встроенной функцией: ему можно передать данные любого типа и он возвратит результат такого же типа (например, `long`, `int` или `double`). Хотя во встроенную функцию также можно передать аргумент любого числового типа, тип значения будет преобразован в `int`, а функция возвратит значение типа `int`, в результате чего может быть потеряна точность. Однако, как будет показано далее (в параграфе “Перегруженные функции”), в языке C++ эти ограничения преодолеваются путем определения нескольких вариантов функции, по одному для каждого типа или для множества типов параметров, передаваемых функции.

## Значения по умолчанию параметров функции

При объявлении функции удобно определять стандартные значения ее параметров (значения по умолчанию). Следующее объявление задает стандартные значения второму и третьему параметрам. Значение по умолчанию параметра `Length`, равное `-1`, заставляет функцию вычислять длину текста, а значение параметра `Color`, равное `0`, задает отображение текста черными буквами.

```
void ShowMessage (char *Text, int Length = -1, int Color = 0);
```

- Задав для некоторого параметра стандартное значение, необходимо определить стандартные значения для *последующих параметров* (т. е. для всех параметров, записанных справа от него).

```
void ShowMessage (char *Text, int Length = -1, int Color);  
// ОШИБКА: пропущено стандартное значение для 3-го параметра
```

- Стандартный параметр при вызове функции работает следующим образом:
  1. При заданном значении фактического параметра компилятор передает это значение в функцию.
  2. При опущенном фактическом параметре компилятор передает стандартное значение параметра (значение по умолчанию).

При вызове функции `ShowMessage` можно указать один, два или три параметра.

```
ShowMessage ("Hello"); // то же, что ShowMessage ("Hello", -1, 0);
```

```
ShowMessage ("Hello", 5); // то же, что ShowMessage ("Hello", 5, 0);
ShowMessage ("Hello", 5, 8);
```

- Опустив параметр с заданным стандартным значением при вызове функции, необходимо опустить все параметры, находящиеся справа от него и имеющие стандартные значения.

```
ShowMessage ("Hello", , 8); // ОШИБКА: синтаксическая ошибка
```

- Определяя стандартные значения, можно использовать выражение, содержащее глобальные переменные или вызовы функций (выражения *не могут* содержать локальных переменных). Допустимо такое объявление (предполагается, что код помещен *вне* функции):

```
// на глобальном уровне
int Palette = 1;
int GetColor (int Pal);
void ShowMessage (char *Text, int Length = -1,
    int Color = GetColor (Palette));
```

- Значения по умолчанию параметров можно задать *в объявлении или определении функции*. Но после определения стандартного значения параметра его уже нельзя переопределять в последующих объявлениях или определениях функции в прежней области видимости (даже для присвоения ему такого же значения). Например, следующий фрагмент программы генерирует ошибку.

```
void ShowMessage (char *Text, int Length = -1, int Color = 0);

void main ()
{
    // код функции ...
}

void ShowMessage (char *Text, int Length = -1, int Color = 0)
// ОШИБКА: повторное назначение стандартных
// значений параметрам 2 и 3
{
    // код функции ...
}
```

- Но в последующие объявления или определения функции в прежней области видимости можно *добавить* одно или более стандартных значений параметров.

```
void ShowMessage (char *Text, int Length = -1, int Color = 0);

// далее в исходном файле:

void ShowMessage (char *Text = "", int Length, int Color);
// ОК: Добавляется значение параметра по умолчанию
```

## Ссылки

Ссылку объявляют, используя оператор ссылки &. Переменная, объявленная как *ссылка*, служит псевдонимом другой переменной. В приведенном ниже примере переменная RefCount объявлена как ссылка на данные типа int и инициализирована так, что ссылается на переменную Count типа int. Это определение заставляет RefCount стать псевдонимом переменной Count, т. е. и RefCount, и Count будут ссылаться на *одно и то же* место в памяти.

```
int Count = 0;
int &RefCount = Count;
```

Пробел непосредственно перед или после оператора & несущественен. Следовательно, можно объявить RefCount как `int& RefCount`, `int & RefCount`, `int &RefCount` или даже `int&RefCount` (хотя последняя форма трудно читаема). Две объявленные таким способом переменные всегда имеют одинаковые значения, а присваивание, сделанное для одной переменной, изменяет значение другой:

```
int Count = 0;
int &RefCount = Count;

// здесь и Count, и RefCount равны 0
Ref Count = 1;

// здесь и Count, и RefCount равны 1
++Count;

// здесь и Count, и RefCount равны 2
```

Для нессылочной переменной инициализация при ее определении заключается в добавлении оператора присваивания, назначающего переменной новое значение. Для ссылочной переменной оператор присваивания = имеет несколько другое значение. Инициализация ссылки задает переменную, для которой ссылка будет псевдонимом. Другими словами, инициализация указывает, какую ячейку памяти представляет ссылочная переменная. Следующий за ним оператор присваивания *изменяет значение* этой области памяти. Переменная, определяемая как ссылка, *должна* быть инициализирована переменной объявленного типа. Как только эта инициализация выполнена, ссылаться на другие переменные уже нельзя. Кроме того, нельзя инициализировать ссылочную переменную значениями констант (например, 5 или 'a'), если переменная не объявлена как ссылка на тип `const` (описан далее в параграфе “Переменные и константы”).

```
int &RInt = 5; // ОШИБКА
```

Переменная, объявленная как ссылка, существенно отличается от указателя. Указатель ссылается на область памяти, содержащую адрес целевой ячейки, а ссылочная переменная – непосредственно на целевую ячейку памяти (в точности совпадая с переменной, используемой для ее инициализации). Рассмотрим пример объявлений:

```
int Count = 0;
int &RefCount = Count;
int *PtrCount = &Count;
```

Указатель `PtrCount` в приведенном коде может быть объявлен и инициализирован. Переменные `Count` и `RefCount` ссылаются на одну и ту же ячейку памяти, поэтому после выполнения операции взятия адреса & они будут иметь одинаковые значения. Помните, что ссылочная переменная всегда ведет себя точно так же, как и переменная, используемая для ее инициализации.

```
int *PtrCount = &RefCount; // что эквивалентно:
                          // int *PtrCount = &Count;
```

Тип *возвращаемого функцией значения* и параметры функции можно объявить с помощью ссылок. Например, в следующем примере функция имеет параметр, который ссылается на переменную типа `int`. Когда эта функция вызывается, ей передается переменная типа `int`, которая используется для инициализации ссылки на параметр `Parm`. Внутри кода функции параметр `Parm` является псевдонимом переменной, переданной при вызове, и любые изменения `Parm` также влияют и на саму переменную.

```
void FuncA (int &Parm);
```

Таким образом, ссылочный параметр предоставляет способ передачи параметра в функцию по ссылке, а не по значению (а в языках C и C++ нессылочные параметры передаются *значением*, т.е. функция получает *копию* оригинала переменной). Следующий код показывает различие между

ссылочным и нессылочным параметром. В этом примере функции FuncA и FuncB увеличивают значение параметра Parm. Однако только функция FuncA, которая получает параметр как ссылку, изменяет значение переменной N, передаваемой из вызывающей функции.

```
void FuncA (int &Parm) //ссылочный параметр
{
    ++Parm;
}

void FuncB (int Parm) //нессылочный параметр
{
    ++Parm;
}

void main ()
{
    int N = 0;

    FuncA (N);      // N передается по ссылке здесь N равна 1
    FuncB (N);      // передается через значение здесь N все еще равна 1
}
```

Нельзя передать константу (например, 5) как параметр, если последний не объявлен как ссылка на тип const (см. параграф “Функции и константы”).

```
void FuncA (int &RInt);
void main ()
{
    FuncA (5);      // ОШИБКА: ссылка не может инициализироваться
                    // значением константы
    // ...
}
```

При необходимости передавать в функцию переменную большого размера (например, большую структуру) использование ссылочного параметра сделает вызов функции более эффективным. При нессылочном параметре все содержимое переменной копируется в параметр. При ссылочном параметре он только инициализируется, ссылаясь непосредственно на оригинал переменной. Отдельная копия переменной при этом не создается. Чтобы использовать преимущества, предоставляемые ссылочным параметром (в частности, возможность изменения переменной, передаваемой вызывающей программой, и устранение операции копирования), в языке C++, как и в языке C, можно использовать в качестве параметра указатель. Однако ссылочными параметрами можно манипулировать с помощью более простого синтаксиса, подобного применяемому при работе с обычными переменными. Оператором &, можно объявить функцию, *возвращающую* ссылку. В приведенном ниже примере объявление означает, что при вызове функции GetIndex будет создан временный псевдоним для некоторой целочисленной переменной (определяемой оператором return внутри функции):

```
int & GetIndex ();
```

Следовательно, вызов функции GetIndex может находиться в любом месте выражения, содержащего данную целочисленную переменную:

```
int N;

N = GetIndex ();    // копирует значение переменной типа int
GetIndex () = 5;    // присваивает 5 переменной типа int
++GetIndex ();      // увеличивает значение переменной типа int
```

Ниже приведен вариант реализации функции `GetIndex`, и примеры ее вызова:

```
int Index = 0;
int & GetIndex ()
{
    return Index;
}

void main ()
{
    int N;
    // здесь Index равно 0

    N = GetIndex ();
    // здесь Index равно 0 и N равно 0

    GetIndex () = 5;
    // здесь Index равно 5

    ++GetIndex ();
    // здесь Index равно 6
}
```

Если функция объявлена как возвращающая значение по ссылке, то она должна возвращать переменную соответствующего типа, и может возвращать константу, например 5, только в случае, если она объявлена как возвращающая ссылку на тип `const` (см. параграф “Функции и константы”). Возвращаемая переменная используется для инициализации временной ссылочной переменной, создаваемой при вызове функции. Другими словами, полагая, что `Index` и `GetIndex` определены так, как показано выше, конструкция

```
++GetIndex ();           // вызов функции приводит к созданию временной
                        // ссылочной переменной, являющейся псевдонимом
                        // переменной Index
```

эквивалентна следующему фрагменту

```
int &Temp = Index; // объявление и инициализация действительной
                  // ссылки на переменную

++Temp;
```

Каждый из этих примеров увеличивает на 1 значение переменной `Index`.

Ссылка, создаваемая при вызове функции, используется *после* возврата из функции, поэтому последняя *не может* вернуть ссылку на переменную, которая уничтожается после выхода из функции. Например, нельзя вернуть ссылку на автоматическую переменную или параметр, как показано в приведенном ниже “опасном” коде. Функция, возвращающая ссылку, может “безопасно” вернуть только глобальную переменную или переменную типа `static`.

```
int &BadIdea1 ()
{
    int i;

    return i;
}

int &BadIdea2 (int Parm)
{
    return Parm;
}
```

Функция, возвращающая нессылочный тип, может выполнить “безопасный” возврат автоматической переменной или параметра, потому что при вызове функции создается *отдельная копия* содержимого переменной, а не просто генерируется ссылка на нее. Именно по этой причине функция, возвращающая значение, не являющееся ссылкой, менее эффективна, чем функция, возвращающая ссылку, особенно если она возвращает большой объект данных. Примеры передачи и возврата ссылок из функций приведены в гл. 6.

## Переменные и константы

Переменную для хранения значения константы можно определить, используя ключевое слово `const`. Объект типа `const` иногда называют *именованной константой*, а не *переменной типа константа*, так как употребление термина *переменная* в данном случае неправомерно. Для определения константной переменной (типа `const`) ее необходимо инициализировать.

```
const int MaxLines = 100;
```

Инициализация, выполняемая один раз, является единственно возможным способом задания значения константной переменной, которое *нельзя* изменить, используя операцию присваивания. Следующий код иллюстрирует допустимые и недопустимые операции с константами.

```
const double CD = 2.5;
double D;

D = CD;    // для чтения константы

CD = 5.0;  // ОШИБКА: нельзя присвоить новое значение константе

++CD;     // ОШИБКА: нельзя изменить значение типа const
```

Инициализировать константную переменную можно:

- используя *константное выражение* (например, 5);
- используя другую *переменную*.

Например, все приведенные ниже определения константных переменных являются корректными:

```
void Func (int Parm)
{
    int I = 3;

    const int CI1 = 5;
    const int CI2 = 2 * sizeof (float);
    const int CI3 = I;
    const int CI4 = Parm;
}
```

Инициализированную выражением, содержащим другие переменные, константная переменная, не может быть использована для определения размерности массива:

```
void Func (int Parm)
{
    const int CI1 = 100;
    const int CI2 = Parm;

    char Buf1 [CI1];    // правильно
    char Buf2 [CI2];    // ОШИБКА: требуется константное выражение
}
```

Этот метод может оказаться некорректным для динамически создаваемого массива, размер которого определяется во время выполнения. В последнем параграфе главы проанализированы корректные методы выделения памяти динамическим массивам. Константные переменные используются так же, как символические константы, определяемые директивой препроцессора `#define` и традиционно применяемые в программах на языке C. Там, где требуется константное выражение, нужно инициализировать константную переменную константным выражением. Подобно константе, определяемой оператором `#define`, константная переменная может быть определена в файле заголовков, включенном в один или более исходных файлов, составляющих программу. В отличие от неконстантной переменной, константная по умолчанию является *локальной* для файла, в котором она определена. Иногда она может быть определена более чем в одном исходном файле, что не приводит к появлению ошибки при компоновке программы. Константная переменная допускает обращение с помощью символического отладчика, что дает ей преимущества перед константой, определенной оператором `#define`.

## Указатели и константы

При объявлении указателя можно использовать ключевое слово `const` несколькими способами.

- Можно определить *указатель на целочисленную константу*:

```
const int *PCInt;
```

Значение такого указателя можно свободно изменять, но значение переменной, на которую он указывает, изменять нельзя:

```
const int A = 1;
int B = 2;

const int *PCInt; // не нужно инициализировать;
                  // PCInt не имеет тип const
PCInt = &A;

*PCInt = 5;        // ОШИБКА: нельзя изменять переменную

PCInt = &B;        // можно изменять PCInt
```

Указателю `PCInt` может быть присвоен адрес целочисленной переменной, которая является константой или таковой не является. Однако, даже если этому указателю присвоен адрес переменной, не являющейся константой, его нельзя использовать для изменения переменной. В этом случае к переменной, допускающей чтение и запись, указатель позволяет выполнить доступ “только для чтения”.

- Можно определить *константный указатель на переменную, не являющуюся константой*:

```
int N;
int *const CPInt = &N;
```

Этот указатель сам является константой, и поэтому он должен быть инициализирован при определении. Впоследствии ему нельзя присваивать другой адрес, но его можно использовать для изменения значения переменной, на которую он указывает:

```
int A = 1;
int B = 2;

int *const CPInt = &A; // нужно инициализировать указатель CPInt

*CPInt = 5;            // можно изменить переменную указатель
```



```
CPInt = &B;           // ОШИБКА: нельзя изменить указатель
                       // типа const
```

- Можно определить *константный указатель на константный объект*. Однако инициализировать его нужно при определении. Нельзя изменить ни значение самого указателя, ни значение переменной, на которую он указывает. Отметим, что *нельзя присвоить адрес константной переменной указателю на переменную-неконстанту*. В этом случае выбирается обходной путь изменения переменной:

```
const int N = 1;
int *PInt;

PInt = &N; // ОШИБКА: нельзя присвоить адрес целочисленной
           // константы указателю на целочисленную неконстанту

*PInt = 2; // Если бы присваивание было допустимо, это выражение
           // привело бы к изменению переменной-константы!
```

## Ссылки и константы

Допускается определение ссылок на константные объекты. Ссылки этого типа можно инициализировать:

- используя *константную переменную*

```
const int A = 1;
const int &RCIntA = A;
```

- используя *переменную-неконстанту*

```
int B;
const int &RCIntB = B;
```

Но в любом из этих случаев ссылку нельзя использовать для изменения значения переменной, на которую ссылается данная ссылка. Если константная ссылка (т.е. ссылка на переменную типа `const`) инициализирована с использованием переменной-неконстанты, то она служит для этой переменной псевдонимом, предназначенным “только для чтения”. Это и есть тот редкий случай, когда ссылка ведет себя не так, как переменная, на которую она ссылается.

```
const int A = 1;
const int &RCIntA = A;

int B;
const int &RCIntB = B;

RCIntA = 5; //ОШИБКА: нельзя изменять значение ссылки на константу
RCIntB = 10; //ОШИБКА: нельзя изменять значение ссылки на константу
```

- Можно инициализировать константную ссылку, используя *константное выражение* (вспомните, что нельзя инициализировать ссылку на переменную-неконстанту константным выражением). В приведенном ниже определении компилятор создает временную переменную типа `const int`, содержащую значение 5, а затем инициализирует `RCInt` как псевдоним этой временной переменной.

```
const int &RCInt = 5; // правильная инициализация
```

- Допустимо объявлять ссылочную переменную константой (хотя это и лишено смысла), поскольку все ссылки автоматически являются константами (вспомните, что после инициализации ссылки ее нельзя сделать ссылкой на другую переменную).

```
int N;

int &const RCInt = N; // разрешено, но бессмысленно
```

Нельзя использовать константную переменную для инициализации ссылки на не константный тип данных.

```
const int CInt = 1;

int &RInt = CInt; // ошибка
RInt = 5;         // Если бы инициализация была допустима, это
                  // выражение привело бы к изменению
                  // константной переменной!
```

## Функции и константы

Используя ключевое слово `const`, можно объявить *параметры* функции и тип *возвращаемого* ею значения.

- Если объявить *параметр* как *константу*, то функция не сможет изменить значение этого параметра (это касается лишь деталей реализации функции). На данную особенность можно не обращать внимания при вызове.

```
void FuncA (const int N); // FuncA не может изменить N; ну и что?
```

- Но если объявить *параметр указателем* или *ссылкой*, то функция может, как обычно, изменять значение передаваемой переменной.

```
FuncA (int *PInt);
FuncB (int &RInt);

void main ()
{
    int N = 1;

    FuncA (&N); // FuncA может изменять значение N
    FuncB (N);  // FuncB может изменять значение N

    // ...
}
```

- А если параметр является *указателем* или *ссылкой на тип const*, то функция не может изменить значение передаваемой переменной. При вызове функции побочный эффект, вызванный изменением значения переменной, отсутствует:

```
FuncA (const int *PInt);
FuncB (const int &RInt);

void main ()
{
    int N = 1;

    FuncA (&N); // FuncA НЕ МОЖЕТ изменить значение N
    FuncB (N);  // FuncB НЕ МОЖЕТ изменить значение N
```

```
// ...
}
```

- Если параметр является *указателем или ссылкой на тип данных const*, то функции *можно передать константную переменную*. Если же параметр является указателем или ссылкой на неконстантный тип данных, то передача константы запрещается, так как это приводит к возникновению некорректной ситуации (изменению константы). Например, для приведенного ниже объявления N можно законно передать адрес N в FuncA или передать N в FuncB, как в приведенном выше примере.

```
const int N = 1;
```

- Если параметр объявлен как *ссылка на константу*, то можно *передать функции константное выражение* (не допустимо для параметра, который является ссылкой на неконстанту).

```
void FuncA (const int &RInt);
```

```
void main ()
{
    FuncA (5);    // разрешено
    // ...
}
```

- Для функции, *возвращающей значение базового типа* (например, int или double), добавление ключевого слова const к спецификации возвращаемого значения в определении функции не имеет особого значения, потому что *изменить* такое возвращаемое значение нельзя никаким способом (т. е. это не *адресуемое выражение*). Например:

```
const int Func (); // Возвращается не адресуемое выражение,
                  // поэтому его никак нельзя изменить
```

- Для функции, *возвращающей указатель или ссылку*, добавление ключевого слова const означает, что вызывающая функция не может использовать возвращаемое значение для изменения значения переменной, на которую указывает или ссылается функция. Заметьте, что в приведенном ниже примере сами функции FuncA и FuncB изменяют значение внутреннего элемента данных. Так как в их объявлении имеется ключевое слово const, аналогичные изменения вызывающей функции производить запрещено.

```
const int *FuncA ()
{
    static int Protected = 1;
    ++Protected;

    return &Protected;
}
```

```
const int &FuncB ()
{
    static int Safe = 100;
    --Safe;

    return Safe;
}
```

```
void main (int Parm)
```

```

{
int N;
N = *FuncA (); //разрешено: N принимает копию Protected
N = FuncB (); //разрешено: N принимает копию переменной Safe

*FuncA () = 5; //ОШИБКА: попытка изменения значения типа const
++FuncB (); //ОШИБКА: попытка изменения значения типа const
}

```

## Перегруженные функции

Язык C++ допускает определение в одной программе нескольких функций с *одним и тем же* именем, если функции с идентичными именами отличаются по числу и типам параметров. Например, объявление двух различных вариантов функции Abs, один для вычисления абсолютной величины типа int, а другой – для значения типа double, может выглядеть так:

```

int Abs (int N)
{
return N < 0 ? -N : N;
}

double Abs (double N)
{
return N < 0.0 ? -N : N;
}

```

*Перегруженными функциями* называются такие функции, которые при идентичности названий объявлены внутри одной области видимости. Компилятор автоматически вызывает соответствующий вариант перегруженной функции на основании типа параметра (параметров) в фактическом вызове функции:

```

int Abs (int N); // оба варианта функции Abs объявлены в
double Abs (double N); // области видимости файла

void main ()
{
int I;
double D;
I = Abs (5); // вызывает 'int Abs (int N)'
D = Abs (-2.5); // вызывает 'double Abs (double N)'
// ...
}

```

Перегруженные функции должны отличаться хотя бы по одному из двух следующих признаков:

- различие в числе параметров;
- различие в типах одного или более параметров.

Перегруженные функции могут возвращать результаты различных типов. Однако, как показано в следующем ошибочном коде, две перегруженные функции не могут отличаться *только* типом возвращаемого значения.

```

int Abs (int N)
{
return N < 0 ? -N : N;
}

```

```
double Abs (int N) // ОШИБКА: перегруженная функция, которая
                  // отличается типом возвращаемого значения
{
    return (double N) (N < 0.0 ? -N : N);
}
```

Обратите внимание: если два параметра отличаются тем, что один из них имеет тип `const` или является ссылкой, то при определении перегруженных функций данные параметры будут считаться однотипными. В приведенном ниже примере параметры типа `int` и `const int` будут инициализированы с использованием одинакового множества типов данных. При передаче любого из этих типов данных после вызова функции компилятор не сможет определить, какая именно перегруженная функция вызывается.

```
int Abs (int N);
int Abs (const int N); //ОШИБКА: список параметров слишком похож
```

Нельзя определить перегруженные функции следующим образом (так как в обе функции будет передана переменная типа `int`):

```
int Abs (int N);
int Abs (int &N); // ОШИБКА: список параметров слишком похож
```

При передаче в момент вызова в перегруженную функцию аргумента, который не совпадает с типом аргумента, определенного для одного из вариантов функции, компилятор попытается преобразовать аргумент к одному из определенных типов. Будет выполнено либо стандартное преобразование (например, `int` в `long`), либо преобразование, определенное пользователем (гл. 6). Если преобразовать типы невозможно, то компилятор сгенерирует ошибку. В противном случае, т. е. когда преобразование возможно, будет вызвана функция, спецификация аргументов которой наиболее близка полученной. Когда функции подобны в равной степени, генерируется ошибка (вызов функции неоднозначен). Более подробно критерии сравнения типов параметров перегруженных функций рассмотрены в справочной системе *Visual C++*.

Причиной неоднозначного вызова перегруженных функций могут быть также параметры функции со стандартными значениями:

```
void Display (char *Buffer);
void Display (char *Buffer, int Lenght = 32);
```

Приведенный ниже вызов сгенерирует сообщение об ошибке, потому что список параметров соответствует *обеим* перегруженным функциям. Обратите внимание: компилятор сигнализирует об ошибке только при неоднозначном вызове. Он не генерирует ошибку при объявлении перегружаемых функций.

```
Display ("Hello"); //ОШИБКА: неоднозначный вызов!
```

Информация о перегрузке функций-членов класса (в частности, о конструкторах) приведена в гл. 4, а перегрузка стандартных операторов языка C рассмотрена в гл. 6. Определение *шаблонов* функции, как альтернативного и более эффективного способа обработки различных типов данных при вызовах функций дано в гл. 7.

## Операторы `new` и `delete`

Для принудительного выделения и освобождения блоков памяти в языке C++ используются операторы `new` и `delete`. Область памяти, в которой размещаются выделяемые блоки, известна как свободная память. Оператору `new` передается тип создаваемой переменной. Оператор выделяет блок памяти достаточной длины для размещения объекта описанного типа и адрес блока возвращается как

указатель на заданный тип данных. Блок памяти можно выделить для хранения объекта встроенного типа, например `char`, `int` или `double`:

```
char *PChar;           // объявление указателя
int *PInt;
double *PDouble;

PChar = new char;       // размещение объекта в памяти
PInt = new int;
PDouble = new double;

*PChar = 'a';           // присваивание значения
*PInt = 5;
*PDouble = 2.25;
```

Но гораздо чаще оператор `new` используется для размещения в памяти данных определенных пользователем типов, например, структур:

```
struct Node
{
    char *Name;
    int Value;
    Node *Next;
};
// ...

Node *PNode;           // объявить указатель

PNode = new Node;       // разместить в памяти

PNode->Name = "hello";   // присвоить значение
PNode->Value = 1;
PNode->Next = 0;
```

Если требуемый объем памяти выделить нельзя, то будет возвращено значение 0. Следовательно, указатель перед его использованием необходимо проверить.

```
PNode = new Node;
if (PNode == 0)
    // блок обработки ошибки ...
else
    // использование PNode ...
```

Дополнительные методы обработки таких ошибок вы узнаете в гл. 8.

Блок памяти, выделенный оператором `new`, по завершении использования можно освободить с помощью оператора `delete` с указателем, содержащим адрес блока. Например, блоки памяти, выделенные в предыдущем примере, освобождаются следующими операторами:

```
delete PChar;
delete PInt;
delete PDouble;
delete PNode;
```

Рекомендуется проверять, не вызывается ли оператор `delete` более одного раза с использованием одного адреса. Поэтому хорошим стилем программирования является установка указателя переменной в 0 сразу после использования оператора `delete` (удаление указателя на 0 всегда безопасно).

В отличие от глобального или локального объекта, можно точно управлять периодом жизни объекта, созданного оператором `new`. Глобально определенный объект остается в памяти на протяжении всего времени исполнения программы. Локально определенный объект существует, пока управление программы остается в блоке, в котором он определен. Объект же, созданный оператором `new`, может быть помещен в любую точку программы и освобожден в любой точке программы с использованием оператора `delete`. Операторы `new` и `delete` полезны для динамического создания объектов в памяти, особенно когда число или размер объектов неизвестны при компиляции. Эти операторы обычно удобней традиционных семейств функций распределения памяти `malloc`, предоставляемых исполняемой библиотекой. В отличие от `malloc`, операция `new` автоматически определяет правильный размер объекта и возвращает указатель корректного типа. Как вы увидите в гл. 4, при использовании оператора `new` для объектов типа класс, автоматически вызывается *конструктор* класса (т.е. функция, инициализирующая его). При использовании оператора `delete` автоматически вызывается *деструктор* класса (если он был определен). Операторы `new` и `delete` можно перегрузить, чтобы настроить управление памятью в программе. Общие вопросы перегрузки операторов рассмотрены в гл. 6. Информация о перегрузке операторов `new` и `delete` представлена справочной документацией.

Выделение памяти для размещения массивов с помощью оператора `new` требует задать:

- *базовый тип данных* (т.е. тип данных элементов массива);
- *число элементов* (указывается внутри квадратных скобок "[ ]"). Обратите вниманиис: здесь можно указать число элементов массива, используя переменную, что было недопустимо в объявлении переменной массива.

Например:

```
void Func (int Size)
{
    char *String = new char [25];    //массив из 25 символов
    int *ArrayInt = new int [Size];  //массив из 'Size' целых
    double *ArrayDouble;
    ArrayDouble = new double [32];   //массив из 32 переменных
                                    // двойной точности

    // ...
}
```

В результате выделения памяти для размещения массива, оператор `new` возвращает адрес первого элемента массива. Чтобы освободить массив, к оператору `delete` при вызове требуется добавить пару квадратных скобок "[ ]", обозначающих, что освобождается массив, а не единичный объект базового типа. Например, массивы, размещенные в предыдущем примере, освобождаются следующими операторами:

```
delete [] String;
delete [] ArrayInt;
delete [] ArrayDouble;
```

Выделенный с помощью оператора `new` блок памяти, *не* может быть инициализирован автоматически значениями 0. Однако при использовании оператора `new` для выделения памяти объекту встроенного типа (например, `char`) можно явно инициализировать объект константой соответствующего типа:

```
char *PChar = new char ('a'); // инициализирует char значением 'a'
int *PInt = new int (3);      // инициализирует int значением 3
```

Для определенного пользователем типа (например, структуры) также можно инициализировать объект существующим значением этого типа. В приведенном ниже примере содержимое `NodeA` будет скопировано поле за полем в новый объект, для которого оператором `new` выделена память.

```

struct Node
{
    char *String;
    int Value;
    Node *Next;
};

void Func ()
{
    Node NodeA = {"hello", 1, 0};

    Node *PNode = new Node (NodeA);
}

```

Изнутри оператора `new` нельзя инициализировать массив встроенного типа. Необходимо самому написать код, инициализирующий массив *после* выполнения оператора `new`. Однако (гл. 4) можно задать специальную функцию (называемую конструктором класса) для инициализации массива, тип которого определяется пользователем, если оператор `new` использован для выделения памяти массиву.

## Резюме

---

Мы рассмотрели методы преобразования программ на языке C в программы на C++. Уделено внимание и множеству новых средств, предоставляемых C++, которые не связаны с классами:

- *C++ как расширение языка C.* Язык C++ является своеобразным расширением языка C (кроме немногих исключений), следовательно, можно перейти к программированию на C++, компилируя программы, написанные на C, используя компилятор C++ вместо компилятора C. Затем можно постепенно добавлять в код уникальные средства языка C++. Некоторые конструкции, возможно, потребуются изменить при компиляции программы, написанной на C, компилятором языка C++. Практически все эти конструкции являются устаревшей практикой программирования на языке C и будут замечены компилятором C++.
- *Комментарий-строку* в языке C++ легко задать символами `“//”`.
- *Локальную переменную* на языке C++ можно объявить в любом месте кода, непосредственно перед ссылкой на данную переменную. *Не обязательно* размещать все локальные объявления в начале блока.
- *Область видимости.* Можно сослаться на глобальную переменную, скрытую локальной переменной с идентичным именем, поместив *операцию расширения области видимости* `“::”` перед именем глобальной переменной.
- *Inline-функции.* Если объявить функцию, используя ключевое слово `inline`, то компилятор заменит (если это возможно) все вызовы функции копией фактического кода функции.
- *Значения по умолчанию.* При объявлении или определении функции допустимо присвоить одному или более параметрам стандартные значения. Если при вызове функции опустить параметр со стандартным значением, то компилятор автоматически передаст стандартное значение для данного параметра.
- *Переменная-ссылка.* Символ `&` используется для определения переменной типа *ссылка*, служащей псевдонимом другой переменной. Кроме того, как ссылки могут быть определены параметры функции и тип возвращаемого функцией значения.
- *Переменные-константы.* Можно определить переменную для хранения значения константы, используя ключевое слово `const`. Указатель или ссылка на тип `const` используется как значение,



предназначенное “только для чтения”. Параметр функции, объявленный как указатель или ссылка на данные типа `const`, гарантирует, что функция значения переданной переменной не изменит.

- *Перегружаемые функции.* Под одним именем допускается объявлять несколько функций, если каждая из них отличается числом или типами параметров. Когда такие функции встречаются внутри одной области видимости, говорят, что они перегружаются. При вызове *перегружаемой* функции, компилятор вызывает, соответствующую числу и типу передаваемых параметров функцию.
- *Управление памятью.* Выделять и освобождать блоки памяти из свободной памяти можно, используя операторы C++ `new` и `delete`. Механизм работы этих операторов адаптирован к нуждам программ, написанных на C++. Особенно эффективно применение этих операторов в использующих классы программах.

# Глава 4

## Классы C++

---

- Определение класса
- Экземпляр класса и доступ к нему
- Инкапсуляция
- Конструктор и деструктор класса
- Встроенные функции-члены
- Размещение определений класса в программе
- Указатель `this`
- Статические члены класса

Глава посвящена классам языка C++. При определении класса создается новый тип данных, который можно использовать подобно встроенному типу данных C++. Однако, в отличие от встроенных типов, классы содержат как данные, так и функции (функции, содержащиеся в стандартных библиотеках, описаны отдельно от определений типов). Класс позволяет инкапсулировать все функции и данные, необходимые для управления частными компонентами программы (например, окном на экране; рисунком, построенным с помощью графической программы; устройством, подключенным к компьютеру; задачей, выполняемой операционной системой). Рассматриваются базовые средства создания и использования отдельных классов. Техника определения и использования иерархии связанных классов рассматривается в следующей главе.

### Определение класса

---

Используемое в языке C++ понятие класса очень близко к стандартной структуре языка C, хотя средства, предоставляемые классами C++, превосходят возможности структур языка C. Для понимания классов C++ полезно сначала обсудить использование структур в C. Структуры языка C позволяют сгруппировать набор связанных переменных-членов. Например, если создан прямоугольник, удобно сохранить его координаты в виде структуры, определенной так:

```
struct Rectangle
{
    int Left;
    int Top;
    int Right;
    int Bottom;
};
```

Затем можно определить функцию рисования прямоугольника. В этом примере `Line` – гипотетическая функция, которая позволяет рисовать линию от точки, заданной первыми двумя координатами, до точки, определенной вторыми двумя координатами. Такая функция может быть определена где-либо в программе или вызвана из библиотеки.

```
void DrawRectangle (struct Rectangle *Rect)
{
    Line (Rect->Left, Rect->Top, Rect->Right, Rect->Top);
}
```

```

    Line (Rect->Right, Rect->Top, Rect->Right, Rect->Bottom);
    Line (Rect->Right, Rect->Bottom, Rect->Left, Rect->Bottom);
    Line (Rect->Left, Rect->Bottom, Rect->Left, Rect->Top);
}

```

Чтобы нарисовать прямоугольник в определенном месте, нужно определить и инициализировать переменную типа `Rectangle`, а затем передать ее в функцию `DrawRectangle`.

```

struct Rectangle Rect = (25, 25, 100, 100);
DrawRectangle (&Rect);

```

В отличие от структуры в С, класс языка С++ определяет не только семейство компонентов данных, но и функции, работающие с этими данными. В С++ можно совместить координаты прямоугольника и функции рисования прямоугольника внутри единого определения класса:

```

class CRectangle
{
    int Left;
    int Top;
    int Right;
    int Bottom;

    void Draw (void)
    {
        Line (Left, Top, Right, Top);
        Line (Right, Top, Right, Bottom);
        Line (Right, Bottom, Left, Bottom);
        Line (Left, Bottom, Left, Top);
    }
};

```

Определенные внутри класса компоненты данных, называются *переменными-членами* класса (иногда их называют также *полями данных*). Функции, определенные внутри класса, называются *функциями-членами* или *методами* класса. В этом примере переменные-члены – `Left`, `Top`, `Right` и `Bottom`, а функция-член – `Draw`. Обратите внимание: функция-член может, не используя специальный синтаксис, содержать ссылку на любую переменную класса.

## Экземпляр класса и доступ к нему

Определение структуры в языке С передает компилятору сообщение о ее форме, но не резервирует место в памяти и не создает переменную, которую можно использовать для хранения данных.

```

struct Rectangle
{
    int Left;
    int Top;
    int Right;
    int Bottom;
};

```

Для того, чтобы зарезервировать память и создать переменную, нужно задать определение

```

struct Rectangle Rect;

```

Класс, например `CRectangle`, показанный в предыдущем параграфе, определяется аналогично. При этом компилятору предоставляется проект класса, но в действительности место в памяти не резервируется. Как и структура, такая переменная-член должна определяться выражением

```
CRectangle Rect;
```

Такое определение создает *экземпляр* класса CRectangle, который также называют *объектом* (иногда – *представителем* или *копией* класса). В этой книге термины *экземпляр класса* и *объект* используются как синонимы. Экземпляр Rect класса CRectangle занимает собственный блок памяти и может использоваться для хранения данных и выполнения операций над ними. Как и переменная встроенного типа, объект существует, пока поток управления не выходит за пределы области видимости его определения (например, если объект определен внутри функции, то уничтожается при выходе из нее). Точно так же, как и для структур языка C, определение класса должно предшествовать определению и использованию экземпляра класса в исходном файле.

Кроме того, экземпляр класса можно создать, используя имеющийся в языке C++ оператор new. Этот оператор выделяет блок памяти достаточного объема, чтобы разместить в нем экземпляр класса, и возвращает указатель на данный блок.

```
CRectangle *PRect = new CRectangle;
```

Объект будет оставаться в памяти, пока вы явно не освободите ее с помощью оператора delete (как это делается для встроенных типов данных показано в гл. 3).

```
delete *PRect;
```

При создании экземпляра класса перед именем класса можно не указывать слово class. В C++ определение класса создает новый тип данных, на который можно сослаться, используя одно лишь имя класса. Можно создать произвольное число экземпляров данного класса.

Для созданного экземпляра класса организуется доступ к переменным-членам и функциям-членам класса. При этом используется синтаксис, подобный применяемому для работы со структурами языка C. Однако при наличии приведенного выше определения класса Rectangle программа не сможет обратиться ни к одному из его членов, так как по умолчанию все переменные и функции, принадлежащие классу, определены как *закрытые* (*private*) (иногда говорят *внутренние*, *личные* или *частные*). Это означает, что они могут использоваться только внутри функций-членов самого класса. Так, для функции Draw разрешен доступ к переменным-членам Top, Left, Right и Bottom, потому что Draw – функция-член класса. Для других частей программы, таких как функция main, доступ к переменным-членам или вызов функции-члена Draw запрещен. К счастью, можно использовать *спецификатор доступа* public, чтобы создать открытый член класса (иногда называемый *общедоступным* или *публичным*), доступный для использования всеми функциями программы (как внутри класса, так и за его пределами). Например, в следующем варианте класса Rectangle открытыми являются все члены. Используемый спецификатор доступа применяется ко *всем* членам, расположенным после него в определении класса (пока не встретится другой спецификатор доступа, как будет показано ниже).

```
class CRectangle
{
public:
    int Left;
    int Top;
    int Right;
    int Bottom;

    void Draw (void)
    {
        Line (Left, Top, Right, Top);
        Line (Right, Top, Right, Bottom);
        Line (Right, Bottom, Left, Bottom);
        Line (Left, Bottom, Left, Top);
    }
};
```

В результате приведенного определения все члены класса `CRectangle` открыты, доступ к ним возможен с использованием оператора `"."`, как и доступ к полям структуры в языке C

```
CRectangle Rect;           // определение объекта CRectangle

Rect.Left = 5;             // присваивание значений переменным-членам
Rect.Top = 10;             // для указания координат прямоугольника
Rect.Right = 100;
Rect.Bottom = 150;

Rect.Draw ();              // создание прямоугольника
```

Кроме того, можно создать экземпляр класса оператором `new`, а затем использовать указатель на экземпляр для доступа к переменным-членам класса. Например:

```
CRectangle *PRect = new CRectangle;

PRect->Left = 5;
PRect->Top = 10;
PRect->Right = 100;
PRect->Bottom = 150;

PRect->Draw ();
```

## Инкапсуляция

---

В соответствии с принципами *инкапсуляции* внутренние структуры данных, используемые в реализации класса, не должны быть доступны пользователю класса непосредственно (преимущества инкапсуляции будут рассмотрены ниже). Однако текущая версия класса `Rectangle` явно нарушает этот принцип, так как пользователь может непосредственно читать или модифицировать любые переменные-члены. Для достижения большей инкапсуляции класс `CRectangle` должен быть определен так, чтобы иметь доступ к функциям-членам (в нашем случае к `Draw`), но не иметь доступа к внутренним переменным-членам (`Left`, `Top`, `Right` и `Bottom`), используемым этими функциями.

```
class CRectangle
{
private:
    int Left;
    int Top;
    int Right;
    int Bottom;

public:
    void Draw (void)
    {
        Line (Left, Top, Right, Top);
        Line (Right, Top, Right, Bottom);
        Line (Right, Bottom, Left, Bottom);
        Line (Left, Bottom, Left, Top);
    }
};
```

Переменные доступны только функциям-членам класса. Спецификатор доступа `private` делает переменные, определенные позже, закрытыми. Подобно спецификатору доступа `public`, рассмотренному ранее, спецификатор `private` воздействует на все объявления, стоящие после него,

пока не встретится другой спецификатор. Следовательно, такое определение делает переменные `Left`, `Top`, `Right` и `Bottom` закрытыми, а функцию `Draw` открытой. Заметим: в действительности не требуется помещать спецификатор `private` в начале определения класса, потому что члены класса по умолчанию являются закрытыми. Однако включение спецификатора `private` облегчает чтение программы. Язык C++ предоставляет еще и третий вид доступа – `protected`. Этот спецификатор требует понимания термина “наследование” и будет рассмотрен в гл. 5.

Ниже приведен пример, который иллюстрирует как корректное, так и некорректное обращение к членам очередного варианта класса `Rectangle`.

```
void main ()
{
    CRectangle Rect;    // определение объекта CRectangle

    Rect.Left = 5;      // ОШИБКА: нет доступа к закрытому члену
    Rect.Top = 10;      // ОШИБКА
    Rect.Right = 100;   // ОШИБКА
    Rect.Bottom = 150;  // ОШИБКА
    Rect.Draw ();       // допускается (но координаты не определены)
}
```

Пользователю класса запрещен прямой доступ к переменным-членам, поэтому класс должен предоставить альтернативное средство указания координат перед созданием прямоугольника. Хороший способ для этого – предоставление открытой функции-члена, принимающей требуемые значения координат и использующей эти значения для установки переменных-членов. Эта функция добавляется в раздел `public` определения класса `Rectangle`, поэтому ее можно вызвать из любой функции программы.

```
void SetCoord (int L, int T, int R, int B)
{
    L = __min ( __max (0,L), 80);
    T = __min ( __max (0,T), 25);
    R = __min ( __max (0,R), 80);
    B = __min ( __max (0,B), 25);
    R = __max (R,L);
    B = __max (B,T);
    Left = L; Top = T; Right = R; Bottom = B;
}
```

Обратите внимание: при необходимости, перед присваиванием параметров переменным класса, функция `SetCoord` организует проверку принадлежности значений параметров диапазону корректных значений и следит, чтобы правая координата была больше левой, а нижняя – больше верхней. Макросы `__max` и `__min` представляются динамической библиотекой языка C++. Для их применения в программу нужно включить файл заголовков `Stdlib.h`. Теперь класс `CRectangle` можно использовать для создания прямоугольника, как показано в следующем примере:

```
void main ()
{
    //...
    CRectangle Rect;

    Rect.SetCoord (25,25,100,100);    // установка координат
                                      // прямоугольника
    Rect.Draw ();                     // отображение прямоугольника

    // ...
};
```

Иногда необходимо добавить функцию-член, позволяющую другим частям программы *получать* текущие значения координат прямоугольника. Эта функция также должна быть добавлена в раздел `public` определения класса.

```
void GetCoord (int L, int T, int R, int B)
{
    *L = Left;
    *T = Top;
    *R = Right;
    *B = Bottom;
}
```

Ниже приведено законченное определение класса `Rectangle`, содержащее новые функции-члены `SetCoord` и `GetCoord`. Теперь, с помощью функций-членов `SetCoord` и `GetCoord` класс `Rectangle` предоставляет доступ к закрытым переменным (согласно принципам инкапсуляции) только посредством четко определенного интерфейса, контролирующего корректность новых присвоенных значений и корректирующего эти значения при необходимости.

```
#include <Stdlib.h>

class Rectangle
{
private:
    int Left;
    int Top;
    int Right;
    int Bottom;

public:
    void Draw (void)
    {
        Line (Left, Top, Right, Top);
        Line (Right, Top, Right, Bottom);
        Line (Right, Bottom, Left, Bottom);
        Line (Left, Bottom, Left, Top);
    }

    void GetCoord (int *L, int *T, int *R, int *B)
    {
        *L = Left;
        *T = Top;
        *R = Right;
        *B = Bottom;
    }

    void SetCoord (int L, int T, int R, int B)
    {
        L = __min ( __max (0,L), 80);
        T = __min ( __max (0,T), 25);
        R = __min ( __max (0,R), 80);
        B = __min ( __max (0,B), 25);
        R = __max (R,L);
        B = __max (B,T);
        Left = L; Top = T; Right = R; Bottom = B;
    }
};
```

Программирование с использованием инкапсуляции обладает рядом преимуществ:

- Инкапсуляция позволяет разработчику класса *проверить правильность любых значений*, присваиваемых переменным-членам, и тем самым предотвратить ошибки программирования.
- Инкапсуляция исключает зависимость пользователя класса от специфического внутреннего представления данных, задаваемого автором класса. Ограничение внешнего доступа к внутренним структурам данных позволяет автору класса свободно *изменять способ представления* этих данных, не изменяя другие части программы, использующие класс (до тех пор, пока сохраняется интерфейс вызовов общедоступных функций-членов). Простой пример: автор класса CRectangle решил хранить координаты левого верхнего угла прямоугольника вместе с его шириной и высотой вместо координат правого нижнего угла. В этом случае переменные-члены могут быть определены так:

```
private:
    int Left;
    int Top;
    int Width;
    int Height;
```

До тех пор, пока интерфейс вызова функций SetCoord и GetCoord остается прежним, внутренние изменения не влияют на другие части программы или любые другие программы, использующие класс Rectangle. (Конечно, эти две функции пришлось бы изменить так, чтобы выполнять преобразования между значениями координат и значениями ширины и высоты.)

## Конструктор и деструктор класса

---

Для класса можно определить функции-члены двух специальных типов:

- *конструкторы*;
- *деструкторы*.

### Конструктор класса

Последний из приведенных выше вариантов класса CRectangle позволяет инициализировать переменные-члены путем вызова функции-члена SetCoord. В качестве альтернативного способа инициализации переменных можно определить специальную функцию класса, называемую *конструктором*. Конструктор автоматически вызывается при создании экземпляра класса. Он может инициализировать переменные класса и выполнять любые другие задачи инициализации, необходимые для подготовки объекта к использованию. Конструктор имеет такое же имя, как и класс. При определении конструктора нельзя указывать тип возвращаемого значения, даже void (конструктор никогда не возвращает значение). Однако он может принять любое число аргументов (включая нулевое). Например, следующий вариант класса CRectangle содержит конструктор, принимающий четыре параметра для инициализации переменных-членов:

```
class CRectangle
{
private:
    int Left;
    int Top;
    int Right;
    int Bottom;

public:
    // конструктор:
```



```

    CRectangle (int L, int T, int R, int B)
    {
        SetCoord (L, T, R, B);
    }

    // определения других функций-членов ...
};

```

Конструктор для создания экземпляра класса должен быть функцией-членом типа *public*. (При использовании класса только для порождения других классов можно сделать конструктор *защищенным* членом, поместив его в раздел *protected*, как описано в гл. 5.) При определении объекта значения параметров передаются конструктору с использованием синтаксиса, подобного обычному вызову функции. В результате выполнения приведенного ниже кода создается экземпляр класса CRectangle путем вызова конструктора класса и передачи ему заданных значений параметров.

```
CRectangle Rect (25,25,100,100);
```

Используя конструктор, можно создать объект Rectangle и нарисовать прямоугольник с помощью двух операторов (а не трех, как в параграфе “Инкапсуляция”).

```

void main ()
{
    CRectangle Rect (25,25,100,100);    // Создание объекта и описание
                                        // размеров прямоугольника
    Rect.Draw ();                       // Рисование прямоугольника
}

```

Для создания экземпляра класса значения параметров можно передать конструктору, используя оператор new. Оператор new автоматически вызывает конструктор для созданного им объекта, что является важным преимуществом использования оператора new по сравнению с другими методами выделения памяти, например, функцией malloc.

```
CRectangle *PRect = new CRectangle (25,25,100,100);
```

## Конструктор по умолчанию

*Конструктором по умолчанию* называют конструктор без параметров. Такой конструктор обычно инициализирует переменные-члены, присваивая им стандартные, установленные по умолчанию значения. Например, приведенный ниже вариант класса CRectangle имеет конструктор по умолчанию, инициализирующий все данные значениями 0. Конструктор с одним или более параметрами, имеющими стандартные значения, также считается конструктором по умолчанию, потому что его *можно* вызвать без передачи параметров (см. в гл. 3 параграф “Значения по умолчанию параметров функции”).

```

class CRectangle
{
private:
    int Left;
    int Top;
    int Right;
    int Bottom;

public:
    CRectangle ()
    {
        Left = Top = Right = Bottom = 0;
    }

    // определения других функций-членов ...
};

```

Компилятор сам генерирует конструктор по умолчанию, если конструктор для какого-либо класса *не* определен. Такие конструкторы, сгенерированные компилятором, *не присваивают* начальные значения переменным-членам класса. Поэтому, если необходимо однозначно инициализировать переменные-члены или выполнить любые другие задачи инициализации, нужно определить собственный конструктор. Если класс имеет конструктор по умолчанию (явно определенный или сгенерированный компилятором), можно определить объект класса без передачи параметров, как это сделано в следующей конструкции.

```
CRectangle Rect;
```

Если параметры не передаются конструктору, в определение объекта *не нужно включать пустые круглые скобки*. Включать их нужно не при определении экземпляра класса, а при объявлении функции, возвращающей тип класса.

```
CRectangle Rect (); // определение функции, которая не принимает  
// параметры и возвращает объект CRectangle
```

Если все-таки включить круглые скобки в определение объекта, компилятор не сгенерирует сообщение об ошибке до тех пор, пока вы не попытаетесь использовать Rect как экземпляр класса. В гл. 6 приведено описание особенностей конструкторов, использующих единственный параметр (в параграфе “Конструкторы копирования и преобразования”).

## Перегруженный конструктор

Можно перегружать конструктор класса или любую другую функцию-член класса, за исключением деструктора (аналогично перегрузке функций, описанной в гл. 3). Как будет показано далее, деструкторы нельзя перегружать из-за отсутствия параметров. Перегруженные конструкторы достаточно распространены. Они предоставляют альтернативные способы инициализации вновь создаваемого объекта класса. Например, следующее определение класса CRectangle содержит перегруженные конструкторы, которые позволяют задавать начальные значения переменным-членам либо просто принимать в качестве начальных стандартные значения.

```
class CRectangle  
{  
private:  
    int Left;  
    int Top;  
    int Right;  
    int Bottom;  
  
public:  
    // конструктор по умолчанию:  
    CRectangle ()  
    {  
        Left = Top = Right = Bottom = 0;  
    }  
  
    // конструктор с параметрами:  
    CRectangle (int L, int T, int R, int B)  
    {  
        SetCoord (L, T, R, B);  
    }  
  
    // определения других функций-членов ...  
};
```

Использование перегруженного конструктора CRectangle продемонстрировано ниже.

```

void main ()
{
    // создание объекта с использованием конструктора по умолчанию:
    CRectangle Rect1;

    // создание объекта с указанием начальных значений:
    CRectangle Rect2 (25, 25, 100, 100);

    //...
}

```

Если определить конструктор класса, то компилятор *не создает* конструктор по умолчанию. Иными словами, если определен один конструктор или более, но среди них нет конструктора по умолчанию, класс *не будет иметь* такового. При использовании класса без конструктора по умолчанию, в определенных ситуациях могут возникать ошибки (см. ниже).

## Объекты-константы и функции-члены

Добавление спецификатора `const` к определению переменной означает невозможность изменения ее значения (см. гл. 3). Аналогично, добавление спецификатора `const` в определение объекта класса означает, что нельзя изменять значения переменных, принадлежащих этому классу. Например:

```

class CTest
{
public:
    int A;
    int B;
    CTest (int AVal, int BVal);
    {
        A = AVal;
        B = BVal;
    }
};

```

Создавая объект этого класса, можно инициализировать обе переменные-члены. Это первая и последняя возможность присвоить значения таким переменным.

```
const CTest Test (1, 2);
```

Из-за спецификатора `const` недопустимо присваивание

```

Test.A = 3;    // ОШИБКА: нельзя изменять переменные-члены
               // объекта типа const

```

Можно использовать и другие варианты объявления объекта с использованием спецификатора `const`. Чтобы продемонстрировать их, рассмотрим объявление объекта класса `CRectangle`.

```
const CRectangle Rect (5, 5, 25, 25);
```

Функция-член `GetCoord` класса `CRectangle` не изменяет никаких переменных-членов, но компилятор не позволит программе вызвать ее для объекта с атрибутом `const`. Поскольку обычная функция-член *может* изменить значение одной или нескольких переменных-членов, компилятор не позволяет вызвать функцию-член объекта типа `const`.

```

int L, T, R, B;
Rect.GetCoord (&L, &T, &R, &B); //ошибка

```

Компилятор не имеет возможности проверить, модифицирует ли в действительности метод класса переменные-члены, так как реализация функции может находиться в отдельном файле исходного кода. Чтобы иметь возможность вызова функции `GetCoord` для объекта типа `const`, следует

включить в определение функции спецификатор `const`. Спецификатор `const` в определении функции `GetCoord` означает, что функция *не может* изменять переменные-члены. Если она пытается это сделать, компилятор будет генерировать ошибку при компиляции исходного кода.

```
class CRectangle
{
    //...
    void GetCoord (int *L, int *T, int *R, int *B) const
    {
        *L = Left;
        *T = Top;
        *R = Right;
        *B = Bottom;
    }
    // ...
};
```

Функцию `GetCoord` сейчас можно вызвать для объекта типа `const` класса `CRectangle`.

```
const CRectangle Rect (5, 5, 25, 25);

int L, T, R, B;
Rect.GetCoord (&L, &T, &R, &B); // допустимо: функция GetCoord
                                // объявлена как const
```

Можно и функцию-член `Draw` объявить как `const`, так как она не изменяет значений переменных. Очевидно, имеет смысл добавлять спецификатор `const` ко всем функциям-членам, которые не модифицируют поля класса, чтобы пользователь мог свободно вызывать такие функции для объектов типа `const`. Конечно, нельзя объявить как `const` функцию, подобную `CRectangle::SetCoord`.

## Инициализация переменных-членов в конструкторе

Следующее определение класса содержит ошибки, поскольку при определении класса запрещается инициализировать переменные-члены. Инициализация переменных внутри определения класса бессмысленна, потому что определение класса задает всего лишь *тип* каждой из переменных-членов, но не резервирует для них реальную область памяти.

```
class C
{
private:
    int N = 0;                //ОШИБКА
    const int CInt = 5;       //ОШИБКА
    int &RInt = N;            //ОШИБКА
    // ...
};
```

Скорее всего, при написании программы требуется инициализировать переменные-члены каждый раз при описании экземпляра класса. Следовательно, целесообразно инициализировать переменные внутри конструктора класса. Конструктор класса `CRectangle` инициализирует переменные-члены, используя выражение *присваивания*. Однако определенным типам данных, в частности, константам и ссылкам, *не могут* быть присвоены значения. Чтобы эта проблема не возникала, в языке C++ для конструктора предусмотрен специальный механизм, называемый *списком инициализации*, который позволяет *инициализировать* одну или более переменных, а не *присваивать* им значения. Список инициализации в определении конструктора помещается непосредственно после списка параметров. Он содержит двоеточие с последующим одним или несколькими *инициализаторами полей*, отделенными друг от друга запятыми. Инициализатор поля содержит имя переменной с последующим начальным значением в круглых скобках. Например, в приведенном ниже классе конструктор содержит

список инициализации, который, в свою очередь, содержит для всех переменных класса инициализаторы полей.

```
class C
{
    private:
        int N;
        const int CInt;
        int &RInt;
        // ...

    public:
        C (int Parm) : N (Parm), CInt (5), RInt (N)
        {
            // код конструктора ...
        }

        // ...
};
```

Применение списка инициализации, где переменные N и CInt инициализированы значениями 0 и 5, а переменная-член RInt – как ссылка на переменную N выглядит так:

```
C CObject (0);
```

Можно определить переменную, являющуюся объектом другого класса, т. е. можно встроить объект одного класса в объект другого. Такие переменные называют *объектами-членами* или *встроенными объектами*. Его можно инициализировать, передавая требуемые параметры конструктору, помещенному в список инициализации конструктора класса, содержащего объект-член. Например, класс CContainer в следующем примере содержит объект-член класса CEmbedded, инициализируемый в конструкторе класса CContainer. Если объект-член *не* инициализирован в списке инициализации конструктора или если конструктор генерируется компилятором, то последний автоматически вызовет для объекта конструктор по умолчанию, если таковой имеется (вспомните: не каждый класс имеет конструктор по умолчанию). Если подобного рода конструктор отсутствует, то компилятор выдаст ошибку.

```
class CEmbedded
{
    //...

public:
    CEmbedded (int Parm1, int Parm2)
    {
        // ...
    }

    // ...
};

class CContainer
{
    private:
        CEmbedded Embedded;

    public:
        CContainer (int P1, int P2, int P3) : Embedded (P1, P2)
        {
            // код конструктора ...
        }

        // ...
};
```

Список инициализации можно также использовать внутри конструктора производного класса, чтобы передать значения в конструктор, принадлежащий базовому классу (см. гл. 5).

## Деструктор

В программе на языке C++ можно определить специальную функцию-член, называемую *деструктором* и автоматически вызываемую всякий раз при уничтожении объекта. Имя деструктора совпадает с именем класса и содержит дополнительный символ префикса “~”. По аналогии с конструктором деструктор определяется без задания типа возвращаемого значения (нельзя указать даже void). Однако в отличие от конструктора ему нельзя передавать параметры. Например, деструктор класса CMessage, можно задать так:

```
~CMessage ()
{
    // код деструктора ...
}
```

На деструктор можно возложить любые задачи, необходимые для удаления объекта. Например, если конструктор класса (CMessage) выделяет блок памяти для хранения строки сообщения, то деструктор непосредственно перед удалением экземпляра класса должен освободить память.

```
#include <string.h>

class CMessage
{
private:
    char *Buffer;    // хранит строку сообщения

public:
    CMessage ()
    {
        Buffer = new char ('\0'); // инициализирует буфер
                                   // пустой строкой
    }

    ~CMessage ()           // деструктор класса
    {
        delete [] Buffer;   // освобождает память
    }

    void Display ()
    {
        // код для отображения содержимого переменной Buffer ...
    }

    void Set (char *String) // запись новой строки сообщения
    {
        delete [] Buffer;
        Buffer = new char [strlen (String) + 1];
        strcpy (Buffer, String);
    }
};
```

## Вызов конструктора и деструктора

Конструктор вызывается при создании объекта, а деструктор – при его уничтожении. В следующем списке для различного вида объектов подробно перечислены способы вызова конструкторов и деструкторов:

- *Глобально определенный объект* (т. е. вне любой функции). Конструктор вызывается в самом начале программы до вызова функции `main` (или `WinMain` в программе, работающей в среде Windows), деструктор – по окончании программы.
- *Локально определенный объект* (т.е. внутри функции). Конструктор вызывается, когда поток управления достигает определения объекта, деструктор – при выходе за пределы блока, в котором определен объект (т.е. когда объект выходит из области видимости).
- *Локально определенный с использованием спецификатора `static` объект*. Конструктор вызывается, когда поток управления *впервые* достигает определения объекта, деструктор – в конце программы.
- *Динамически созданный с использованием оператора `new` объект*. Конструктор вызывается при создании объекта, а деструктор – когда объект явно уничтожается с использованием оператора `delete`. Если этого не происходит, деструктор не будет вызван никогда.

Можно определить массив объектов, как показано в следующем примере (`CRectangle` – это класс, рассмотренный выше).

```
CRectangle RectTable [10];
```

Массив объектов также можно создать динамически.

```
CRectangle *RectTable = new CRectangle [10];
```

При создании объекта в обоих случаях компилятор вызывает конструктор по умолчанию для каждого элемента массива. Когда же массив уничтожается, компилятор вызывает деструктор для каждого такого элемента (полагая, что деструктор определен для класса). Если класс не имеет конструктора по умолчанию, возникает ошибка. Конструкторы для элементов массива вызываются в порядке возрастания адресов, а деструкторы – в обратном порядке.

- *Именованный массив объектов*. Если вы определили именованный массив объектов (как в первом примере), то каждый элемент массива можно инициализировать, передавая в конструктор требуемые значения (см. в гл. 6 параграф “Инициализация массивов”).
- *Динамически созданный массив*. Если массив создан динамически с использованием оператора `new`, нельзя обеспечить инициализацию отдельных элементов, так как компилятор всегда вызывает конструктор по умолчанию для каждого элемента. Кроме того, при уничтожении динамически созданного массива нужно в оператор `delete` добавить символы массива “[ ]”. Если символы “[ ]” отсутствуют, то компилятор вызовет деструктор только для *первого* элемента массива.

```
delete [] RectTable;
```

## Встроенные функции-члены

Функции-члены в каждом из примеров классов, приведенных в этой главе, полностью определялись *внутри* тела определения класса. В качестве альтернативы можно *определить* функцию-член вне класса, а внутри класса только *объявить*.

```
class CRectangle
{
private:
```

```

    int Left;
    int Top;
    int Right;
    int Bottom;

public:
    CRectangle ();
    CRectangle (int L, int T, int R, int B);
    void Draw (void);
    void GetCoord (int *L, int *T, int *R, int *B);
    void SetCoord (int L, int T, int R, int B);
};

```

Перед именем функции-члена при внешнем определении нужно поставить имя класса с последующим оператором расширения области видимости "::", как показано ниже.

```

#include <stdlib.h>

CRectangle::CRectangle ()
{
    Left = Top = Right = Bottom = 0;
}

CRectangle::CRectangle (int L, int T, int R, int B)
{
    SetCoord (L,T,R,B);
}

void CRectangle::Draw (void)
{
    Line (Left, Top, Right, Top);
    Line (Right, Top, Right, Bottom);
    Line (Right, Bottom, Left, Bottom);
    Line (Left, Bottom, Left, Top);
}

void CRectangle::GetCoord (int *L, int *T, int *R, int *B)
{
    *L = Left;
    *T = Top;
    *R = Right;
    *B = Bottom;
}

void CRectangle::SetCoord (int L, int T, int R, int B)
{
    L = __min (__max (0,L), 80);
    T = __min (__max (0,T), 25);
    R = __min (__max (0,R), 80);
    B = __min (__max (0,B), 25);
    R = __max (R,L);
    B = __max (B,T);
    Left = L; Top = T; Right = R; Bottom = B;
}

```

Функция, определенная внутри тела класса, существенным образом отличается от функции, определенной вне класса. По умолчанию для последней не выполняется inline-подстановка (см. описание inline-функций в гл. 3). Следовательно, очень короткие функции можно определить внутри тела



класса, а более длинные – вне его. Например, конструкторы класса `CRectangle` и метод `GetCoord` (самые короткие) могут быть определены внутри класса, а функции `Draw` и `SetCoord` (самые длинные) – вне его. Полное определение класса `CRectangle` приведено в конце следующего параграфа в листингах 4.1 и 4.2. Можно заставить компилятор рассматривать функцию, заданную вне определения класса, как встроенную, используя спецификатор `inline`, как было описано в гл. 3. Например, объявив функцию `CRectangle::GetCoord` внутри определения класса `CRectangle`, можно сделать ее встроенной:

```
void inline GetCoord (int *L, int *T, int *R, int *B);
```

после чего вне описания класса *определить* ее:

```
void inline CRectangle::GetCoord (int *L, int *T, int *R, int *B)
{
    *L = Left;
    *T = Top;
    *R = Right;
    *B = Bottom;
}
```

## Размещение определения класса в программе

---

- *Встроенные функции-члены.* В программе состоящей из нескольких исходных файлов, написанных на языке C++, определение класса обычно помещается вместе с определениями всех встроенных функций-членов внутри единого заголовочного файла (файла с расширением `.h`). Затем данный файл включается в какой-либо один исходный файл, в котором используется этот класс. Такая организация файлов гарантирует, что определение класса вместе с кодом любой встроенной функции будет доступно при обращении к классу или при вызовах его функций-членов. Как указывалось в гл. 3, компилятор должен иметь доступ к встроенной функции при каждом ее вызове.
- *Невстроенные функции-члены.* Определения любых функций-членов, не являющихся встроенными, обычно также размещаются внутри отдельного файла, называемого файлом *реализации* класса. Скомпилированный вариант файла реализации необходимо скомпоновать с программой (например, путем включения файла реализации `.cpp` в список файлов проекта Visual C++). Если разместить определения невстроенных функций в файле заголовков вместо отдельного файла реализации и включить заголовочный файл более чем в один исходный файл, то компоновщик выдаст сообщение об ошибке “повторное символическое определение” (*symbol redefinition*).

Приведенные ниже листинги 4.1 и 4.2 содержат полный исходный код последнего варианта класса `CRectangle`. Определение класса `CRectangle` помещено в файл с именем `CRect.h` (файл заголовков класса), а определения функций-членов, не являющихся встроенными, – в `CRect.cpp` (файл реализации класса). При создании класса `CRectangle` предполагалось, что функция `Line` определена в другом модуле. Файл `CRect.h` должен быть включен в любой файл, содержащий ссылку на класс `CRectangle` (включая `CRect.cpp`!), а скомпилированный вариант `CRect.cpp` должен быть скомпонован с программой.

---

### Листинг 4.1

```
// CRect.h: файл заголовков CRectangle

class CRectangle
{
private:
    int Left;
```

```

    int Top;
    int Right;
    int Bottom;

public:
    CRectangle ()
    {
        Left = Top = Right = Bottom = 0;
    }
    CRectangle (int L, int T, int R, int B)
    {
        SetCoord (L, T, R, B);
    }
    void Draw (void);
    void GetCoord (int *L, int *T, int *R, int *B)
    {
        *L = Left;
        *T = Top;
        *R = Right;
        *B = Bottom;
    }
    void SetCoord (int L, int T, int R, int B);
};

```

---

#### ЛИСТИНГ 4.2

```

// CRect.cpp: файл реализации

#include "crect.h"
#include <stdlib.h>

void Line (int X1, int Y1, int X2, int Y2);

void CRectangle::Draw (void)
{
    Line (Left, Top, Right, Top);
    Line (Right, Top, Right, Bottom);
    Line (Right, Bottom, Left, Bottom);
    Line (Left, Bottom, Left, Top);
}

void CRectangle::SetCoord (int L, int T, int R, int B)
{
    L = __min (__max (0,L), 80);
    T = __min (__max (0,T), 25);
    R = __min (__max (0,R), 80);
    B = __min (__max (0,B), 25);
    R = __max (R,L);
    B = __max (B,T);
    Left = L; Top = T; Right = R; Bottom = B;
}

```

## Указатель `this`

---

Если на переменные-члены класса нужно сослаться из кода, находящегося *вне* класса, в выражении всегда следует указывать экземпляр класса. Благодаря этому компилятор может определить, какой экземпляр переменных-членов ему доступен. Предположим, например, что класс `CTest` содержит переменную `N`, тогда чтобы сначала напечатать копию переменной `N` объекта `Test1`, а затем – объекта `*PTest2`, можно использовать такой код:

```
CTest Test1;
CTest *PTest2 = new CTest;
//...
cout << Test1.N << '\n';
cout << Test2->N << '\n';
```

Если на переменную-член нужно сослаться *внутри* функции-члена, то частный экземпляр класса *не* описывается. Например:

```
class CTest
{
public:
    int N;

    int GetN ()
    {
        return N;    // N возвращается БЕЗ точного указания объекта
    }
}
```

В действительности, компилятор передает методу класса скрытый указатель на объект. Функция использует именно этот скрытый указатель для доступа к корректной копии переменных-членов. Например, в следующем вызове компилятор передает функции `GetN` скрытый указатель на объект `Test`.

```
Test.GetN ();
```

Функция `GetN` использует именно этот указатель для доступа к принадлежащей объекту `Test` копии переменной `N`.

Непосредственно обратиться к скрытому указателю можно с использованием спецификатора `this`. Другими словами, внутри функции-члена указатель `this` является предопределенным указателем, содержащим адрес объекта, на который ссылается функция (иногда называемого *текущим* объектом). Так функция `GetN` может быть описана так:

```
int GetN ()
{
    return this->N; // эквивалентно 'return N;'
}
```

Поскольку использование указателя `this` и так подразумевается в простой ссылке на переменную-член, то имя функции-члена с выражением `this->` перед ним является корректным, но не преследует никакой цели.

Чтобы получить доступ к глобальным переменным или к функции, имеющей такое же имя, как переменная-член или функция-член, перед именем необходимо поставить операцию расширения области видимости `::`:

```
int N = 0;    // глобальная переменная N

class CTest
```

```

{
public:
    int N;          //переменная-член N

    int Demo ()
    {
        cout << ::N << '\n'; // печать глобальной переменной N
        cout << N << '\n';   // доступ к переменной-члену N
                                // с использованием указателя 'this'
    }
}

```

## Статические члены класса

Экземпляр класса, обычно, имеет собственную копию переменных-членов, принадлежащих классу. Однако, если переменная-член объявлена с использованием спецификатора `static`, возможно существование *единственной копии* этой переменной-члена независимо от того, сколько экземпляров класса создается (даже если *не* создается ни одного экземпляра). Например, поскольку следующий класс определен как статическая переменная `Count`, то независимо от того, сколько создается экземпляров класса `CTest`, существует единственная копия переменной-члена `Count`.

```

class CTest
{
public:
    static int Count;
    // остаток определения класса ...
}

```

Кроме объявления статической переменной внутри класса, ее необходимо также определить и инициализировать *снаружи*, как глобальную переменную-член. Так как определение статических переменных помещено вне класса, необходимо описать класс, в котором они объявлены, используя операцию расширения области видимости (в этом примере `CTest::`). Например, можно следующим образом определить и инициализировать переменную `Count`.

```

// вне любого класса или функции
int CTest::Count = 0;

```

Статические переменные-члены класса существуют независимо от любого объекта класса, поэтому к ним можно получить доступ, используя имя класса и оператор расширения области видимости *без* ссылки на экземпляр класса:

```

void main ()
{
    CTest::Count = 1;

    // ...
}

```

Можно воспринимать статическую переменную как нечто среднее между глобальной переменной и данными обычного типа, принадлежащими классу:

- Подобно глобальной переменной, такая переменная *определяется и инициализируется вне функции*. Фактически, она представляет именованную область памяти, которая существует на протяжении жизни всей программы.

- Однако, как и обычные переменные класса, она *объявляется внутри класса*, а доступ к ней может быть управляемым (т.е. открытым, закрытым или защищенным).

Функцию-член класса также можно определить с использованием спецификатора `static`, например:

```
class CTest
{
    // ...

    static int GetCount ()
    {
        // код функции ...
    }
    // ...
}
```

Если функция-член класса объявлена как статическая, то ей присущи специфические особенности:

- Не принадлежащий классу программный код может *вызвать функцию* с использованием имени класса и оператора расширения области видимости *без ссылки на экземпляр класса*, который не нужен, даже если он существует, например:

```
void main ()
{
    int Count = CTest::GetCount ();
    // ...
}
```

- Если функция-член класса объявлена как статическая, то она *непосредственно может ссылаться только на статические переменные и статические функции*, принадлежащие ее классу. Так как эту функцию можно вызвать без ссылки на экземпляр класса, статическая функция-член не имеет указателя `this`, содержащего адрес объекта. Следовательно, компилятор не сможет определить, какому объекту принадлежат переменные-члены, если такая функция пытается получить непосредственный доступ к нестатическим переменным-членам.

Если переменные-члены и функции-члены объявлены статическими, то они могут использоваться для хранения переменной, применяемой всем классом, или переменной, общей для всех экземпляров класса. Использование статических элементов для подсчета текущего количества экземпляров класса иллюстрирует следующая программа:

```
include <iostream.h>

class CTest
{
private:
    static int Count;

public:
    CTest ()
    {
        ++Count;
    }
    ~CTest ()
    {
        --Count;
    }
}
```

```

static int GetCount ()
{
    return Count;
};

};

int CTest::Count = 0;

void main ()
{
    cout << CTest::GetCount () << " objects exist\n";
    CTest Test1;
    CTest *PTest2 = new CTest;
    cout << CTest::GetCount () << " objects exist\n";
    delete PTest2;
    cout << CTest::GetCount () << " objects exist\n ";
}

```

Результат выполнения этого кода на экране будет выглядеть так:

```

0 objects exist
2 objects exist
1 objects exist

```

## Резюме

---

Мы рассмотрели, как решается задача определения класса, создания экземпляра класса и получения доступа к его данным и функциям. Вы также узнали о некоторых членах класса специального типа: конструкторах, деструкторах, встроенных функциях-членах и статических членах.

- *Класс.* В языке C++ класс – это нечто похожее на структуру в языке C. Однако он может содержать и *переменные-члены* (или *поля*), и функции (известные как *функции-члены* или *методы* класса), которые оперируют переменными-членами. Определение класса создает новый тип данных.
- *Экземпляр.* Чтобы использовать класс, вы должны реально создать члены класса, принадлежащие к этому типу данных. Они называются *экземплярами* класса или *объектами* класса. Можно создать объект класса, определяя его, как переменную встроенного типа. В качестве альтернативного варианта используются операторы `new` и `delete` для динамического создания и разрушения объектов.
- *Доступ к объекту.* Доступ к членам класса организуется операторами “.” или “->”, подобно доступу к элементам структуры языка C. Доступом к членам класса можно управлять, используя *спецификаторы доступа* `public` (открытые) или `private` (закрытые) (переменные класса, закрытые по умолчанию). Согласно принципам *инкапсуляции*, спецификаторы доступа необходимо использовать для предотвращения непосредственного доступа пользователя к переменным-членам внутри класса. Можно создать функцию-член, изменяющую переменные-члены, но только после проверки правильности заданного пользователем значения.
- *Конструкторы и деструкторы.* *Конструктор* – это специальная функция-член, которая автоматически вызывается каждый раз при создании экземпляра класса. Обычно она используется для инициализации переменных-членов или выполнения любых других задач, подготавливающих класс к использованию. Конструктор может быть определен с любым числом параметров. *Конструктор по умолчанию* – это конструктор без параметров. *Деструктор* – это специальная функция-член, вызываемая автоматически при уничтожении объекта класса. Он может быть

использован для освобождения памяти, зарезервированной для хранения объекта класса, или для выполнения других задач очистки памяти.

- *Метод.* Метод (функция-член) класса может быть определен внутри тела класса либо объявлен внутри класса, но определен вне его. Функция-член, определенная внутри класса, автоматически рассматривается как встроенная функция. Функция-член, определенная вне класса, рассматривается как встроенная функция, только в случае, когда она объявлена с использованием спецификатора `inline`. Внутри метода спецификатор `this` содержит адрес объекта, на который ссылается вызов функции (т. е. того объекта, для которого была вызвана функция-член).
- *Статические переменные и методы.* Если переменная-член определена с использованием спецификатора `static`, будет существовать единственная копия такой переменной-члена, независимо от числа созданных экземпляров класса. К статической переменной можно обратиться с использованием имени класса и операции расширения области видимости без ссылки на частный экземпляр класса. Функция-член, определенная с использованием спецификатора `static`, может быть вызвана с использованием имени класса и операции расширения области видимости без ссылки на специфический объект. Такая функция применима только к принадлежащим к ее классу статическим переменным-членам и статическим функциям-членам.

## Глава 5

# Классы-наследники C++

---

- Классы-наследники (производные классы)
- Иерархия классов
- Виртуальные функции

В этой главе мы рассмотрим, как определить классы, производные от других классов (классы-наследники), позволяющие повторно использовать код и структуры данных, принадлежащие уже существующим классам. Производные классы используются для настройки и расширения существующих классов в специальных целях. Изучим также определение и использование *виртуальных* функций-членов классов, которые позволяют настраивать существующие классы, а также управлять разнообразными программными объектами посредством простых и компактных подпрограмм.

## Классы-наследники (производные классы)

---

Будем считать, что используемый для работы с прозрачными прямоугольниками класс `CRectangle` уже написан, отлажен и используется в программе. Предположим также, что кроме прозрачных прямоугольников вам потребовалось отображать и окрашенные блоки (прямоугольники, залитые сплошным цветом). Для этого необходимо определить новый класс. Назовем его, например, `CBlock`. Он будет содержать большую часть свойств класса `CRectangle` и некоторые дополнительные средства для заливки нарисованных прямоугольников. Если создавать `CBlock` как полностью новый класс, придется продублировать большую часть всего написанного ранее для `CRectangle`. К счастью, средства языка C++ позволяют избежать дублирования кода и данных благодаря возможности создавать новые классы как *производные* от существующих. Создаваемый производный класс *наследует* все данные и функции, принадлежащие существующему классу. Например, `CBlock` можно определить так:

```
class CBlock : public CRectangle
{
};
```

Использованное в этом примере выражение `: public CRectangle` обозначает класс `CBlock` как производный от `CRectangle`. Класс `CBlock` наследует все переменные-члены и функции `CRectangle`. Другими словами, несмотря на то, что определение класса `CBlock` является пустым, он уже содержит функции `GetCoord`, `SetCoord` и `Draw`, а также переменные `Left`, `Top`, `Right` и `Bottom`, определенные в классе `CRectangle`. При этом `CRectangle` называют *базовым*, а класс `CBlock` – *производным* классами. В общем случае спецификатор `public` следует включать в первую строку определения производного класса, как показано в приведенном примере. Использование данного спецификатора приводит к тому, что все открытые члены базового класса (объявленные как `public`), в производном классе тоже остаются открытыми.

Создав производный класс, требуется добавить в него новые детали, необходимые для удовлетворения специальных требований, предъявляемых к новому классу. Например, определение класса `CBlock` можно представить так, как показано в приведенном ниже коде. Кроме наследуемых членов, `CBlock` содержит закрытую переменную `FillColor` для хранения цветового кода заливки блока. Значение переменной `FillColor` устанавливается с помощью новой открытой функции-члена



SetColor. Класс CBlock также содержит *новую версию* функции Draw для рисования прозрачных прямоугольников с последующей заливкой сплошным цветом. Фактически класс CBlock включает *две* версии функции Draw – наследуемую и определенную явно.

```
class CBlock : public CRectangle
{
private:
    int FillColor; //хранит цвет, используемый для закрашивания блока
public:
    void Draw (void)
    {
        int L, T, R, B;
        CRectangle::Draw ();
        GetCoord (&L, &T, &R, &B);
        Fill ((L + R) / 2, (T + B) / 2, FillColor);
    }
    void SetColor (int Color)
    {
        FillColor = __max (0, Color);
    }
};
```

При вызове функции Draw для объекта типа CBlock версия функции, определённая внутри CBlock, переопределяет функцию, определённую внутри CRectangle, как показано в следующем фрагменте программы (более эффективный способ задания и переопределения функций класса рассмотрен далее в параграфе “Виртуальные функции”).

```
CBlock Block;
// ...
Block.Draw (); // вызов версии функции Draw, определенной внутри
               // класса CBlock, так как объект Block имеет тип CBlock
```

Определённая для CBlock версия функции Draw, начинается с вызова версии функции Draw класса CRectangle, рисующей прозрачный прямоугольник. Использование выражения с оператором расширения области видимости (CRectangle::) приводит к вызову версии функции Draw, принадлежащей классу CRectangle.

```
CRectangle::Draw();
```

Если же данное выражение будет опущено, то компилятор сгенерирует рекурсивный вызов функции Draw, определённой для текущего класса CBlock. Это пример того, как функция, определённая в рамках текущего класса, заменяет собой наследуемую функцию. На следующем этапе Draw вызывает наследуемую функцию GetCoord для вычисления текущих значений координат прямоугольника, а затем – функцию Fill для заливки внутренней области прямоугольника цветом, соответствующим текущему цветовому коду. Fill – это гипотетическая функция заливки замкнутой области. Первые два параметра задают координаты точки внутренней области, а третий – цветовой код, используемый при заливке этой области. Такая функция может быть определена в любом месте программы или вызвана из библиотеки функций.

```
GetCoord (&L, &T, &R, &B);
Fill ((L + R) / 2, (T + B) / 2, FillColor);
```

Ниже приведен фрагмент программы, в котором используется класс CBlock для рисования закрашенного прямоугольника.

```
CBlock Block; // создание экземпляра класса CBlock
Block.SetCoord (25,25,100,100); // установка координат блока
```

```

Block.SetColor (5);           // установка цвета блока
Block.Draw ();               // вызов функции Draw класса CBlock

```

## Конструктор

Приведенная выше программа с помощью класса CBlock сначала вызывает функции-члены SetCoord и SetColor для установки значений переменных класса, а затем вызывает функцию Draw для рисования блока. Чтобы упростить использование CBlock, необходимо добавить конструктор для установки при создании экземпляра класса исходных значений всех переменных.

```

CBlock (int L, int T, int R, int B, int Color)
: CRectangle (L, T, R, B)
{
    SetColor (Color);
}

```

В данном конструкторе список инициализации членов класса содержит вызов конструктора базового класса CRectangle, которому передаются значения, присваиваемые переменным-членам. Список инициализации может использоваться для инициализации как базового класса, так и переменных членов (он рассматривался в параграфе “Инициализация переменных-членов в конструкторе” гл. 4). Конструктор класса CBlock содержит вызов функции SetColor, которая устанавливает значение переменной FillColor. Теперь, используя конструктор класса, можно создать объект класса CBlock и всего двумя операторами задать блок.

```

CBlock Block (25, 25, 100, 100, 5);
Block.Draw ();

```

Для данного производного класса можно также добавить конструктор по умолчанию, присваивающий нулевые значения всем переменным-членам класса. Так как конструктор по умолчанию явно не инициализирует базовый класс (список инициализации пуст), компилятор *автоматически* вызывает конструктор по умолчанию базового класса (CRectangle::CRectangle()), присваивающий нулевые значения всем переменным, определенным внутри него. Если базовый класс не имеет конструктора по умолчанию, то будет получено сообщение об ошибке.

```

CBlock ()
{
    FillColor = 0;
}

```

Конструктор по умолчанию класса CBlock позволяет создавать объекты, в которых все переменные-члены равны нулю, без передачи значений или вызова методов класса.

```

CBlock Block; // создается объект класса CBlock, у которого всем
              // переменным-членам присвоены нулевые значения

```

Полный текст класса CBlock с двумя новыми конструкторами приведен в конце следующего параграфа (листинг 5.2).

Создавая экземпляр производного класса, компилятор сначала вызывает конструктор базового класса. Затем вызываются конструкторы всех объектов-членов (т.е. тех элементов класса, которые являются объектами классов). Эти конструкторы вызываются в порядке перечисления объектов в определении класса. В завершение вызывается собственный конструктор класса. Деструкторы, если они определены, вызываются в обратной последовательности. Таким образом, если код конструктора выполнен, можно быть уверенным в том, что базовый класс и его члены уже инициализированы и их использование допустимо. Аналогично, если выполнен код собственного деструктора производного класса, то можно быть уверенным в том, что базовый класс и все объекты-члены еще не уничтожены и их по-прежнему можно использовать.

## Доступ к наследуемым переменным

В рассмотренном выше примере класс CBlock наследует переменные-члены Left, Top, Right и Bottom из своего базового класса, но это *не* означает возможности прямого доступа, так как они определены в базовом классе как закрытые. Вместо этого пришлось бы использовать открытую функцию GetCoord, как и в оставшейся части программы. Такое ограничение приводит к тому, что код функций-членов становится в определенной степени неэффективным. В качестве альтернативы (листинг 5.1) можно указать вместо private спецификатор доступа protected.

---

### Листинг 5.1

```
// CRect1.h: файл заголовков класса CRectangle
class CRectangle
{
protected:
    int Left,
    int Top;
    int Right;
    int Bottom;
public:
    CRectangle ()
    {
        Left = Top = Right = Bottom = 0;
    }
    CRectangle (int L, int T, int R, int B)
    {
        SetCoord (L, T, R, B);
    }
    void Draw (void);
    void GetCoord (int *L, int *T, int *R, int *B)
    {
        *L = Left;
        *T = Top;
        *R = Right;
        *B = Bottom;
    }
    void SetCoord (int L, int T, int R, int B);
};
```

В данной версии класса CRectangle предполагается определение методов Draw и SetCoord вне класса. Подобно закрытым членам, к защищенным членам класса доступ извне невозможен:

```
void main ( )
{
    CRectangle Rect;
    Rect.Left = 10; // ОШИБКА: доступ к защищенному
                   // члену класса невозможен
}
```

Но в отличие от закрытых к защищенным членам класса можно непосредственно обращаться из класса, производного от того класса, в котором данная переменная определена. Следовательно (листинг 5.2), CBlock можно переписать, используя прямой доступ к переменным-членам, определенным в CRectangle.

---

## Листинг 5.2

```
// CBlock.h: файл заголовков класса CBlock
#include "rect1.h"
#include <stdlib.h>
void Fill (int X, int Y, int Color);
class CBlock: public CRectangle
{
protected:
    int FillColor;
public:
    CBlock ()
    {
        FillColor = 0;
    }
    CBlock (int L, int T, int R, int B, int Color)
        : CRectangle (L, T, R, B)
    {
        SetColor (Color);
    }
    void Draw (void)
    {
        CRectangle::Draw ( ) ;
        Fill ((Left + Right) / 2, (Top + Bottom) / 2, FillColor);
    }
    void SetColor (int Color)
    {
        FillColor = __max (0, Color);
    }
};
```

Здесь переменную `FillColor` класса `CBlock` предпочтительнее определить как защищенную, а не как закрытую. Это обеспечит доступ к ней из любых классов, производных от `CBlock`, как описано в следующем параграфе. Можно предоставить другому классу или глобальной функции доступ к внутренним или защищенным членам класса, объявив его или функцию *дружественными* (*friend*) для данного класса (см. в гл. 6 раздел “Перегрузка оператора присваивания”).

В языке C++ производные классы почти всегда порождаются открытыми. В частности, во всех примерах мы используем открытые производные классы.

- Если класс порожден с использованием спецификатора `public`, то открытые члены базового класса остаются открытыми, а защищенные – защищенными и в производном классе.

```
class CBlock : public CRectangle
{
    // определение класса ...
}
```

- Если для определения производного класса использован спецификатор `private`, то все закрытые и защищенные члены базового класса становятся закрытыми членами производного класса.

```
class CBlock : private CRectangle
{
    // определение класса ...
}
```

- Если при определении производного класса спецификатор не задан, то по умолчанию он определяется как закрытый.

Независимо от способа наследования, закрытые члены базового класса недоступны для производного (несмотря на то, что они наследуются производным классом и являются частью любого экземпляра производного класса, прямой доступ к ним из производного класса невозможен).

## Иерархия классов

---

Класс-наследник может служить базовым для других производных классов. Таким образом, можно создавать иерархии связанных между собой классов. Например, для создания окрашенных блоков с закругленными углами создадим производный от `CBlock` класс, называющийся `CRoundBlock`. Класс `CRoundBlock` называют прямым (*direct*) потомком класса `CBlock`, который по отношению к производному является прямым базовым (*direct base*) классом. Класс `CRoundBlock` называют косвенным (*indirect*) потомком `CRectangle`, являющегося для него косвенным базовым (*indirect base*) классом.

```
class CRoundBlock : public CBlock
{
protected:
    int Radius;
public:
    CRoundBlock ()
    {
        int Radius = 0;
    }
    CRoundBlock (int L, int T, int R, int B, int Color, int Rad)
        : CBlock (L, T, R, B, Color)
    {
        SetRadius (Rad);
    }
    void Draw (void)
    {
        // рисует прозрачный прямоугольник с закруглением
        // используя значение Radius) ...
        // и выполняет заливку прямоугольника заданным цветом:
        Fill ((Left + Right) / 2, (Top + Bottom) / 2, FillColor);
    }
    void SetRadius (int Rad)
    {
        Radius = __max (0, Rad);
    }
};
```

Описанный выше класс `CRoundBlock` наследует все члены `CBlock`, включая наследуемые классом `CBlock` из `CRectangle`. В классе `CRoundBlock` определена дополнительная переменная `Radius` для хранения радиуса закругления, а также новая версия функции `Draw` для создания залитых прямоугольников с закругленными углами. Класс `CRoundBlock` также предоставляет открытую функцию-член `SetRadius` для установки значения переменной `Radius` и конструкторы (конструктор по умолчанию, устанавливающий значения всех переменных класса в 0, и конструктор, инициализирующий все переменные указанными значениями). Переменная `Radius` класса `CRoundBlock` является защищенной, поэтому она доступна из любого класса, производного от `CRoundBlock`, но не из функций, не принадлежащих иерархии классов. В следующем фрагменте программы создается

экземпляр класса `RoundBlock`, инициализируются все переменные-члены и рисуется блок с закругленными углами.

```
CRoundBlock RoundBlock (10, 15, 50, 75, 5, 3);  
RoundBlock.Draw ();
```

Популярные библиотеки классов, такие, как MFC и стандартная библиотека `iostream`, содержат преимущественно древовидные иерархии родственных классов. В таких иерархиях один класс обычно служит базовым для нескольких производных (что и порождает древовидную структуру). Названные библиотеки классов включены в систему программирования Visual C++. Библиотека MFC рассматривается в третьей части книги. Как и в случае с библиотекой `iostream`, возможна ситуация, когда один класс является производным более чем от одного класса. Это называется *множественным наследованием* (*multiple inheritance*).

Наследование позволяет повторно использовать созданные ранее фрагменты программы и структуры данных. Это позволяет избежать ненужного дублирования. Преимущество наследования также состоит в упрощении отладки и сопровождения программ, поскольку код и данные, управляющие выполняемой задачей, не рассеяны, а содержатся в одном, легко определяемом месте программы. В языке C++ с помощью иерархии классов можно воссоздать модель взаимоотношений реального мира. Например, при создании класса `CBlock`, производного от `CRectangle`, отражается тот факт, что блок является прямоугольником определенного *типа* (прямоугольником, внутренняя область которого залита цветом), а создание класса `CRoundBlock`, производного от `CBlock`, соответствует тому, что закругленный блок является блоком определенного *типа*. В качестве другого примера приведем MFC, иерархия классов которой тщательно моделирует взаимосвязи между различными элементами Windows.

## Иерархия классов на вкладке *Class View*

Эта вкладка отображается в окне Visual Studio (см. гл. 2.).

- Во вкладке *Class View* отображены все *глобальные переменные*, определенные в программе, что позволяет быстро найти описание переменной, выполнив двойной щелчок на ее обозначении.
- В программе, написанной на языке C++ и содержащей классы, вкладка *Class View* также показывает каждый класс вместе с его переменными-членами и функциями.

При двойном щелчке на имени класса, члена класса или глобальной переменной отобразится исходный файл, содержащий определение выбранного элемента, и отмечается конкретное место его расположения в определении. Кроме того, можно выполнять различные операции, щелкая правой кнопкой мыши на имени класса или члена класса и выбирая опции из меню:

- Перейти на *определение или объявление* члена класса.
- Найти все *производные* и *базовые* классы.
- *Добавить* в описание класса новые переменные-члены или функции. Для определения обработчика сообщений в программе, работающей в среде Windows, следует использовать окно *Properties* (гл. 10).
- Найти *ссылки* на член класса.
- Найти функции, *вызывающие* функцию-член, а также *вызываемые* ею функции.
- Назначить *точки останова* функции-члена.
- Если щелкнуть правой кнопкой мыши на вершине списка классов, то можно выбрать команду *New Class ...* для *получения помощи в создании* нового класса.

Отладочная информация (например, информация о точках останова в коде программы) хранится в служебных файлах проекта, не редактируемых напрямую.

## Виртуальные функции

Виртуальные функции позволяют писать простые универсальные подпрограммы для манипулирования объектами различных типов. Они дают возможность изменять свойства существующих базовых классов *даже при отсутствии доступа* к исходному коду этих классов. Для понимания концепции виртуальных функций сравним еще раз приведенные выше классы CRectangle и CBlock. Вспомним: класс CBlock является производным от CRectangle. Следовательно, CRectangle – базовый по отношению к CBlock. Оба этих класса содержат функцию Draw. Если для каждого класса объявлены экземпляры

```
CRectangle Rect;  
CBlock Block;
```

то версию функции Draw, определенную в классе CRectangle, можно вызвать так

```
Rect.Draw();
```

а версию функции Draw, определенную в рамках CBlock, так

```
Block.Draw();
```

Компилятор в обоих случаях без проблем определит, какая версия функции вызывается, так как вызов функции содержит ссылку на экземпляр класса. Однако в языке C++ принято иногда использовать указатель на базовый класс, содержащий либо адрес экземпляра базового, либо адрес экземпляра производного классов. Например:

```
CRectangle *PRect;
```

Этому указателю в C++ разрешается присвоить адрес как экземпляра класса CRectangle, так и экземпляра класса, производного (прямо или косвенно) от CRectangle, без применения операции приведения типов. Например:

```
CRectangle *PRect;    // объявляет указатель на класс CRectangle  
  
CRectangle Rect;      // создает экземпляр класса CRectangle  
CBlock Block;         // создает экземпляр класса CBlock  
  
PRect = &Rect;        // допустимо: присваивает указателю адрес  
                      // экземпляра CRectangle  
PRect = &Block;       // также допустимо: присваивает указателю  
                      // адрес экземпляра CBlock
```

В программах на языке C++ допускается присваивать адрес производного класса указателю базового, так как его использование в таком контексте вполне корректно. Указатель на базовый класс используется для доступа к членам, определенным только в базовом классе. Все они также определены внутри производного класса путем наследования. Следовательно, если указатель содержит адрес объекта производного класса, то можно получить значение любого члена класса, на который задана ссылка с помощью указателя. Хотя компилятор может присвоить адрес базового класса указателю производного класса с помощью операции приведения типов, такой указатель небезопасен, так как может использоваться для доступа к членам, не определенным в данный момент в объекте, на который делается ссылка (базовый класс не всегда содержит все определенные в производном классе члены).

При использовании указателя PRect для вызова функции Draw может возникнуть проблема. Компилятор не может определить заранее, на какой тип объекта указывает PRect, пока программа не начнет выполняться. Поэтому компилятор всегда генерирует вызов версии Draw, определенной в классе CRectangle, так как указатель PRect объявлен как указатель на CRectangle. Допустим, указатель PRect содержит адрес являющегося экземпляром класса CRectangle объекта Rect.

```

CRectangle *PRect;
CRectangle Rect;

// ...

PRect = &Rect;

```

Применение указателя PRect при вызове функции-члена Draw в данном случае будет инициировать вызов определенной в классе CRectangle версии Draw.

```
PRect->Draw();
```

Но даже если указатель PRect содержит адрес экземпляра класса CBlock, то

```

CRectangle *PRect;
CBlock Block;

// ...

PRect = &Block;

```

использование этого указателя для вызова функции-члена Draw программа будет *по-прежнему* вызывать определенную в классе CRectangle версию Draw.

```
PRect->Draw();
```

В данной ситуации мы получаем вызов *неправильной версии* функции Draw, создающей прозрачный, а не сплошной прямоугольник. Решением такой проблемы может быть превращение Draw в *виртуальную* функцию. Определение Draw как виртуальной функции гарантирует, что при запуске программы будет вызвана корректная версия функции, даже если вызов будет осуществляться через указатель базового класса. Чтобы задать функцию Draw как виртуальную, нужно включить спецификатор `virtual` в ее объявление в классе CRectangle.

```

class CRectangle
{
    // другие объявления ...

public:
    virtual void Draw (void);

    // другие объявления ...
}

```

Спецификатор `virtual` нельзя включить в определение функции Draw, находящееся вне определения класса. Спецификатор `virtual` можно включить в объявление функции Draw в производном классе CBlock, хотя необходимости в этом нет. Если функция объявлена в базовом классе как виртуальная, то функция с таким же именем, типом возвращаемого значения и параметрами, объявляемая в производном классе, автоматически рассматривается как виртуальная. Следовательно, нет необходимости повторять спецификатор `virtual` в каждом производном классе, хотя иногда это облегчает чтение программы.

```

class Cblock : public CRectangle
{
    // другие объявления ...

public:
    virtual void Draw (void);

    // другие объявления ...
}

```



Когда в классе функция Draw определена как виртуальная и программа вызывает ее через указатель PRect, как показано ниже, компилятор *не* сгенерирует автоматически вызов версии Draw, определенной в классе CRectangle. Вместо этого компилятор создает специальный код, вызывающий *правильную* версию функции Draw в процессе выполнения программы.

```
CRectangle *PRect;  
  
// ...  
  
PRect->Draw();
```

Таким образом, следующие операторы приведут к вызову определенной в классе CRectangle функции Draw

```
CRectangle *PRect;  
CRectangle Rect;  
  
// ...  
  
PRect = &Rect;  
PRect->Draw ( );
```

А следующие операторы вызывает функция Draw класса CBlock.

```
CRectangle *PRect;  
CBlock Block;  
  
// ...  
  
PRect = &Block;  
PRect->Draw ( );
```

До запуска программы действительный адрес функции не известен, поэтому такой механизм вызова называют *поздним* (или *динамическим*) *связыванием*. Стандартный механизм вызова функций, при котором компилятору заранее известен точный адрес вызова, называется *ранним* (или *статическим*) *связыванием*. Программа должна содержать адрес правильной версии функции в каждом объекте. (Точнее, она хранит адрес таблицы адресов виртуальных функций.) Кроме дополнительного объема памяти, это требует косвенного вызова функций, что приводит к более медленной работе по сравнению с вызовом стандартных функций. Поэтому рекомендуется делать функцию-член виртуальной только тогда, когда требуется действительно позднее связывание.

В объектно-ориентированном программировании поддержка виртуальных функций является важной характеристикой и называется *полиморфизмом*. Полиморфизм – это способность использовать один и тот же оператор для выполнения различных действий. Действие, выполняемое в данный момент, определяется конкретным видом вызываемого объекта. Пример полиморфизма – вызов *одной и той же* функции

```
PRect->Draw ( );
```

для рисования прямоугольников, блоков и блоков с закругленными углами. Конкретно выполняемое действие определяется классом объекта, на который указатель PRect ссылается в данный момент.

## **Управление объектами классов с помощью виртуальных функций**

С помощью виртуальных функций можно создавать простые универсальные подпрограммы, автоматически управляющие множеством объектов различных типов. Предположим, имеется программа, позволяющая создавать прямоугольники, блоки или блоки с закругленными углами. Каждый раз, когда пользователь создает одну из этих фигур, программа вызывает новый оператор для

динамического создания объекта соответствующего класса (CRectangle, CBlock или CRoundBlock), управляющего новой фигурой. Так как CBlock и CRoundBlock являются производными от CRectangle, то указатели на все объекты удобно хранить в виде одномерного массива указателей на CRectangle:

```
const int MAXFIGS = 100;
CRectangle *PFigure [MAXFIGS];
int Count = 0;

// ...
// пользователь создает блок:
PFigure [Count++] = new CBlock (10, 15, 25, 30, 5);
// ...
// пользователь создает прямоугольник:
PFigure [Count++] = new CRectangle(5, 8, 19, 23);
// ...
// пользователь создает блок с закругленными углами:
PFigure [Count++] = new CRoundBlock(27, 33, 43, 56, 10, 5);
```

Пусть решается задача перерисовки всех объектов на экране. Если функция Draw *не определена* как виртуальная, то для каждого элемента массива необходимо каким-либо образом определить тип фигуры, а затем вызвать соответствующую версию функции Draw. Например, в класс CRectangle можно добавить переменную с именем Type, в которой будет храниться код, идентифицирующий класс объекта. В данном примере предполагается, что три символьные константы RECT, BLOCK и ROUNDBLOCK были определены в программе предварительно.

```
// НЕ РЕКОМЕНДУЕТСЯ:

class CRectangle
{
    // другие определения ...

public:
    int: Type; // наследуется всеми производными классами;
               // хранит код, идентифицирующий класс объекта:
               // RECT, BLOCK, либо ROUNDBLOCK
}
```

Переменную Type затем можно использовать для определения типа каждой из фигур массива, чтобы вызвать соответствующую версию функции Draw. Этот пример не только неуклюж, но и требует добавления новой ветки case в оператор switch при изменении программы для поддержки нового типа фигур (например, при внесении в иерархию класса для создания новой фигуры).

```
// дурной тон; НЕ РЕКОМЕНДУЕТСЯ:
for (int i = 0; i < Count; ++i)
    switch (PFigure [i]->Type)
    {
        case RECT:
            PFigure [i]->Draw();
            break;
        case BLOCK:
            ((CBlock *)PFigure [i])->Draw();
            break;
        case ROUNDBLOCK:
            ((CRoundBlock *)PFigure [i])->Draw();
            break;
    }
```

Однако, функцию Draw можно сделать виртуальной, добавив спецификатор `virtual` в ее объявление внутри класса `CRectangle`. В этом случае программа *автоматически* вызовет правильную версию функции для текущего типа объекта. Перерисовка фигуры может быть выполнена с помощью приведенного ниже фрагмента программы. Этот код проще и компактнее. К тому же он не требует изменения при добавлении в иерархию нового класса, поддерживающего фигуры другого типа.

```
for (int i = 0; i < Count; ++i)
    PFigure [i]->Draw();
```

Создаваемая для практических нужд программа рисования, конечно, будет поддерживать множество видов фигур. Тем не менее, для классов, управляющих отдельными фигурами, и производных от общего базового класса, можно использовать аналогичный подход. Программа *ScratchBook*, приведенная в части III этой книги, иллюстрирует описанную методику.

## Модификации базовых классов с помощью виртуальных функций

При необходимости виртуальную функцию можно использовать для модификации базового класса, не изменяя при этом его код. Предположим, задан класс, предназначенный для вывода окна сообщения.

```
class CMessageBox
{
protected:
    char *Message;
    virtual void DrawBackground (int L, int T, int R, int B);
    // закрашивает фон окна сообщения в белый цвет
public:
    CMessageBox()
    {
        Message = new char ('\0');
    }
    ~CMessageBox ( )
    {
        delete [] Message;
    }
    void Display ( )
    {
        DrawBackground (0, 0, 35, 25);
        // код для вывода строки сообщения ...
    }
    void Set (char *Msg);
};
```

Описанная в данном примере открытая функция-член `Set` позволяет передать строку сообщения, а `Display` выводит это сообщение на экран. Отметим: функция `Display` очищает фон, вызывая другую функцию-член `DrawBackground` и передавая ей размеры окна сообщения. Эта функция закрашивает фон, используя непрозрачный белый цвет. Функция `DrawBackground` предназначена для использования внутри класса. Она *не* предназначена для вызова извне класса и, следовательно, объявлена как защищенный член класса. Функция `DrawBackground` также объявлена виртуальной. Соответственно, при порождении от класса `CMessageBox` нового класса, содержащего собственную версию `DrawBackground`, новая функция перекроет старую, даже если будет вызываться из функции-члена класса `CMessageBox`. Например, от класса `CMessageBox` можно породить класс

CMyMessageBox. Обратите внимание: новая версия функции DrawBackground создает синий фон, а не белый.

```
class CMyMessageBox: public CMessageBox
{
protected:
    virtual void DrawBackground (int L, int T, int R, int B)
    {
        // закрашивает фон окна сообщения СИНИМ цветом ...
    }
};
```

В приведенном ниже фрагменте будет создано окно сообщения с синим фоном.

```
CMyMessageBox myMessageBox;

myMessageBox.Set ("hello");
myMessageBox.Display ();
```

Оформляя функцию DrawBackground как виртуальную, мы получаем возможность настраивать класс CMessageBox (а именно, цвет или узор, которым заполняется фон окна сообщения) без модификации исходного текста данного класса (при этом даже не нужно просматривать исходный программный код). Многие классы, определенные в библиотеке MFC, предусматривают использование виртуальных функций, которые можно переопределять в производных классах, что позволяет легко модифицировать MFC-классы в программных проектах.

## Механизм переопределения

При вызове виртуальной функции с помощью указателя на объект класса, вызов будет интерпретироваться в соответствии с текущим типом объекта, а не указателя. Как это осуществляется для виртуальных функций, вызываемых внутри функций-членов базового класса? Вспомним: ссылка на член класса из функции-члена неявно связана с применением указателя this. Таким образом, функцию Display класса CMessageBox можно запрограммировать так:

```
class CMessageBox
{
    // другие определения

public:
    void Display ( )
    {
        this->DrawBackground (0, 0, 35, 25);
        // ...
    }

    // другие определения ...
};
```

В случае, если бы функция DrawBackground *не* являлась виртуальной, то ее вызов из функции Display инициализировал бы вызов версии DrawBackground класса CMessageBox (так как в пределах этого класса указатель this является указателем на объект класса CMessageBox). Однако если функция DrawBackground *является* виртуальной, ее вызов инициализирует вызов версии DrawBackground класса текущего объекта. Таким образом, если функция Display вызвана для объекта класса CMyMessageBox, то будет вызвана и DrawBackground, определенная внутри класса CMyMessageBox.

```
CMyMessageBox MyMessageBox;  
// ...  
MyMessageBox.Display ();
```

## Резюме

---

Рассмотрена базовая методика создания новых классов, производных от уже существующих, а также способы построения иерархии родственных (связанных) классов. Введено понятие виртуальных функций и описаны некоторые способы их использования.

- *Производный и базовый классы.* Можно создать новый класс, порожденный от существующего, указав имя существующего класса в определении нового. Существующий класс в этом случае называют *базовым*, а новый класс – *производным*. Производный класс *наследует* все члены базового класса. В производный класс можно добавить новые члены для его настройки в соответствии с вашими требованиями. Производный класс (*класс-наследник*) может служить базовым для других классов, что позволяет создавать многоуровневую иерархию связанных классов.
- *Конструктор производного класса* может явно инициализировать свой базовый класс путем передачи параметров конструктору базового класса. Если конструктор производного класса *неявно* инициализирует базовый класс, компилятор автоматически вызывает конструктор по умолчанию базового класса.
- *Спецификаторы доступа.* Если члены базового класса объявлены со спецификатором доступа `protected`, то они доступны в производных классах, но не доступны для других функций программы.
- *Наследование* (способность порождать один класс из другого) позволяет повторно использовать коды и структуры данных, ранее созданные для других классов. Это делает программу более понятной и помогает в моделировании взаимосвязей между объектами, управляемыми программой.
- *Виртуальные функции и полиморфизм.* Каждый класс в иерархии производных классов может иметь собственные версии отдельных функций-членов. Если функция объявлена как `virtual`, ее вызов будет автоматически инициализировать вызов версии, определенной для класса текущего объекта, даже если функция вызывается с помощью указателя на базовый класс. Виртуальным функциям свойственен *полиморфизм* – способность использовать один оператор для выполнения любого из множества различных действий. При этом конкретное действие определяется типом вызываемого объекта. Используя виртуальные функции, можно создавать простые универсальные подпрограммы для управления множеством различных (но связанных) объектов. Виртуальные функции позволяют изменять настройки базового класса, принятые по умолчанию, без модификации исходного кода базового класса.

## Глава 6

# Перегрузка, копирование и преобразование

---

- Перегрузка операторов
- Конструкторы копирования и преобразования

Рассмотрим способы настройки созданных классов. Особое внимание уделим процедуре *перегрузки* (overloading) стандартных операторов языка C++, позволяющей описывать их поведение при работе с объектами классов. Рассмотрим также определение специальных конструкторов:

- *конструктор копирования* (copy constructor), инициализирующий объект класса другим объектом такого же типа;
- *конструктор преобразования* (conversion constructor), преобразовывающий данные различных типов в объект класса.

## Перегрузка операторов

---

В языке C++ операторы применяются к данным встроенных типов заранее определенным способом. Например, при применении операции сложения к двум переменным типа `int` выполняется целочисленное сложение, а при использовании оператора “+” для двух переменных типа `double` – операция сложения с плавающей запятой. В языке C++ допустимо применение стандартных операторов к объектам классов при условии точного описания действия этих операторов. Определение способа работы оператора с объектами конкретного класса называется *перегрузкой* оператора. Предположим, имеется класс `CCurrency`, предназначенный для хранения и обработки денежных сумм. В классе `CCurrency` денежные суммы хранятся в виде целочисленных величин, предназначенных для последующей обработки с использованием быстродействующих целочисленных операций.

```
class CCurrency
{
private:
    long Dollars;
    int Cents;
public:
    CCurrency ()
    {
        Dollars = Cents = 0;
    }
    CCurrency (long Dol, int Cen)
    {
        SetAmount (Dol, Cen);
    }
    void GetAmount (long *PDol, int *PCen)
    {
        *PDol = Dollars;
        *PCen = Cents;
    }
}
```

```

void PrintAmount ()
{
    cout.fill('0');
    cout.width (1);
    cout << '$' << Dollars << '.';
    cout.width (2);
    cout << Cents << '\n';
}
void SetAmount (long Dol, int Cen)
{
    // проверка суммы центов, которая превышает 100:
    Dollars = Dol + Cen / 100;
    Cents = Cen % 100;
}
};

```

Класс содержит конструктор по умолчанию для установки начальных значений переменных, содержащих количество долларов и центов, в 0 и конструктор для присвоения этим переменным конкретных значений. Предусмотрены отдельные функция-член (SetAmount) для задания величины денежной суммы и функции-члены GetAmount и PrintAmount для подсчета суммы и ее печати соответственно. Заметим: можно хранить значение суммы в одной переменной long integer в виде общей суммы в центах, что позволяет обрабатывать большие числа. Однако предпочтительнее для хранения величины, обозначающей количество долларов, использовать переменную типа long, а для центов – переменную типа int. В результате можно будет оперировать количеством долларов, достигающим максимальной величины типа long, что в системе программирования Visual C++ составляет 2147483647. Несмотря на то, что в Visual C++ 7 размер переменной типа int совпадает с размером переменной типа long (4 байта), переменная Dollars явно объявлена как переменная типа long для того, чтобы она могла содержать максимально возможные целочисленные значения при использовании любого компилятора.

В качестве альтернативы заданию функций-членов для арифметических операций с денежными суммами, можно *перегрузить* стандартные операторы C++. Это позволяет выполнять арифметические действия, применяя выражения, традиционно используемые для встроенных типов данных. Для перегрузки оператора определяется функция с именем operator, за которым следует обозначение операции. Например, добавив в определение класса CCurrency следующую функцию-член, можно перегрузить оператор “+”.

```

class CCurrency
{
    // другие объявления ...
public:
    CCurrency operator+ (CCurrency Curr)
    {
        CCurrency Temp (Dollars + Curr.Dollars,
                        Cents + Curr.Cents);
        return Temp;
    }
    // другие объявления ...
};

```

Приведенная в данном примере функция-оператор определяется как *public*, чтобы ее могли использовать другие функции-программы. Если такая функция задана, операцию сложения можно реализовать так:

```

CCurrency Amount1 (12, 95);
CCurrency Amount2 (4, 38);

```

```
CCurrency Total;
Total = Amount1 + Amount2;
```

Выражение `Amount1 + Amount2` компилятор языка C++ интерпретирует как

```
Amount1.operator+ (Amount2);
```

Используемая функция `operator+` создает временный объект `Temp` класса `CCurrency`, содержащий величину денежной суммы, полученной в результате сложения двух объектов. Затем она возвращает временный объект. Кроме того, приведенное выше выражение присваивает возвращаемый объект экземпляру `Total` класса `CCurrency`. Такой способ присваивания делает возможным поэлементное копирование компилятором переменных-членов одного класса в другой.

Выражение, содержащее более одного перегруженного оператора “+”, вычисляется, подобно стандартной операции сложения, слева направо. Например, следующая программа использует перегруженный оператор для сложения хранимых в трех объектах класса `CCurrency` величин.

```
void main ()
{
    CCurrency Advertising (235, 42);
    CCurrency Rent (823, 68);
    CCurrency Entertainment (1024, 32)
    CCurrency Overhead;

    Overhead = Advertising + Rent + Entertainment;
    Overhead.PrintAmount();
}
```

Используемую функцию `operator+` можно упростить, заменив неявным временным объектом локальный временный объект класса `CCurrency`. При вызове конструктора класса компилятор из выражения создает временный объект класса. Функция `operator+` непосредственно возвращает содержимое этого временного объекта (как объекты класса возвращаются функциями описано ниже в параграфе “Конструктор копирования”).

```
CCurrency operator+ (CCurrency Curr)
{
    return CCurrency (Dollars + Curr.Dollars, Cents + Curr.Cents);
}
```

Для более эффективной реализации функции `operator+` можно передавать ей *ссылку* на объект класса `CCurrency`, а не сам объект. Передача ссылок исключает необходимость копирования объекта в локальный параметр, что особенно важно для объектов больших размеров (см. гл. 3). Следующий фрагмент программы является окончательной версией функции `operator+`. Использование спецификатора `const` при объявлении параметра – гарантия того, что значение параметра не изменится функцией.

```
CCurrency operator+ (const CCurrency &Curr)
{
    return CCurrency (Dollars + Curr.Dollars, Cents + Curr.Cents);
}
```

## Определение функций-операторов

Функции-операторы могут быть перегружены подобно другим функциям языка C++. При этом есть несколько способов вызова операторов. Например, с помощью оператора “+” можно сложить объект класса `CCurrency` с константой типа `int` или `long`. (В определенной выше функции предполагается, что оба операнда являются объектами класса `CCurrency`.) Для этого в класс `CCurrency` нужно добавить следующую функцию.



```

CCurrency operator+ (long Dol)
{
    return CCurrency (Dollars + Dol, Cents);
}

```

Эта функция позволит использовать оператор “+” так:

```

CCurrency Advertising (235, 42);
// ...
Advertising = Advertising + 100;

```

Выражение Advertising + 100 будет интерпретироваться компилятор как

```
Advertising.operator+ (100)
```

вызывая в результате новую версию функции operator+. Тем не менее, целочисленную константу *нельзя* поставить первой, так как компилятор будет интерпретировать выражение 100 + Advertising следующим образом

```
100.operator+ (Advertising) // бессмысленно!
```

Это ограничение можно обойти, написав функцию-оператор, которая *не является членом класса*, но первый параметр которой имеет тип long.

```

// определяется глобально:
CCurrency operator+ (long Dol, const CCurrency &Curr)
{
    return CCurrency (Dol + Curr.Dollars, Curr.Cents);
}

```

Однако, при использовании этой функции также могут возникнуть проблемы. Так как она не является функцией-членом *класса* CCurrency, то не имеет доступа к закрытым переменным этого класса (а именно, к Dollars и Cents). Для получения такого доступа функцию нужно сделать *дружественной* классу CCurrency, объявив ее внутри определения CCurrency с использованием спецификатора friend.

```

class CCurrency
{
    // другие объявления ...
    friend CCurrency operator+ (long Dol, const CCurrency &Curr);
    // другие объявления ...
};

```

Дружественная функция – даже если она не является функцией-членом класса – имеет доступ как к закрытым, так и к защищенным членам класса, который объявляет ее дружественной. Когда такая функция-оператор определена, операцию сложения можно применить так

```

CCurrency Advertising (235, 42);
// ...
Advertising = 100 + Advertising;

```

Выражение 100 + Advertising компилятор теперь проинтерпретирует следующим образом

```
operator+ (100, Advertising)
```

в результате чего вызовет *дружественную* версию функции-оператора.

Можно сделать дружественными две первые версии функции operator+, не определяя их как функции-члены класса (хотя особенного преимущества это не дает).

```

friend CCurrency operator+ (const CCurrency &Curr1,
                             const CCurrency &Curr2);
friend CCurrency operator+ (const CCurrency &Curr, long Dol);

```

Перегружающая оператор функция, если она не является методом класса, должна иметь хотя бы один параметр – объект класса. Следовательно, функцию-оператор нельзя использовать для изменения стандартного смысла операторов в выражении, содержащем только данные встроенных типов.

Полное описание класса CCurrency вместе с функцией, не являющейся его членом, выглядит так.

```

#include <iostream.h>

class CCurrency
{
private:
    long Dollars;
    int Cents;
public:
    CCurrency ()
    {
        Dollars = Cents = 0;
    }
    CCurrency (long Dol, int Cen)
    {
        SetAmount (Dol, Cen);
    }
    void GetAmount (long *PDol, int *PCen)
    {
        *PDol = Dollars;
        *PCen = Cents;
    }
    void PrintAmount ( )
    {
        cout.fill ('0');
        cout.width (1);
        cout << '$' << Dollars << '.';
        cout.width (2);
        cout << Cents << '\n';
    }
    void SetAmount (long Dol, int Cen)
    {
        // проверка суммы центов, которая превышает 100:
        Dollars = Dol + Cen / 100;
        Cents = Cen % 100;
    }
    CCurrency operator+ (const CCurrency &Curr)
    {
        return CCurrency (Dollars + Curr.Dollars,
                           Cents + Curr.Cents) ;
    }
    CCurrency operator+ (long Dol)
    {
        return CCurrency (Dollars + Dol, Cents);
    }
    friend CCurrency operator+ (long Dol, const CCurrency &Curr);
};
CCurrency operator+ (long Dol, const CCurrency &Curr)
{
    return CCurrency (Dol + Curr.Dollars, Curr.Cents);
}

```

Ниже приведен пример использования всех трех определенных выше функций-операторов.

```
void main ()
{
    CCurrency Advertising (235, 42);
    CCurrency Rent (823, 68);
    CCurrency Entertainment (1024, 32);
    CCurrency Overhead;

    Overhead = Advertising + Rent + Entertainment;
    Overhead.PrintAmount ();
    Overhead = Overhead + 100;
    Overhead.PrintAmount ();

    Overhead = 100 + Overhead;
    Overhead.PrintAmount ();
}
```

На печать будут выведены значения.

```
$2083.42
$2183.42
$2283.42
```

Итак, определение трех версий функций-операторов дает возможность применить операцию сложения к двум объектам, объекту и константе, а также к константе и объекту. Четвертая возможная комбинация из двух констант дает стандартную операцию сложения. Далее в параграфе “Конструктор преобразования” рассмотрена технология создания специального конструктора, исключаящего из класса CCurrency две версии функции operator+.

Кроме приведенных усовершенствований для класса CCurrency можно добавить средства для обработки отрицательных значений денежных сумм и перегрузить другие операторы (обычно при присваивании отрицательной суммы результат непредсказуем). Можно определить функции-операторы для операции вычитания двух объектов или объекта и константы, а также для операций умножения или деления объекта на константу. При определении таких функций избегайте преобразований сумм долларов и центов в одно значение типа long, хранящееся как общее количество центов. Такое преобразование упрощает выполнение арифметических действий, но уменьшает максимальное значение суммы долларов и повышает вероятность переполнения.

## Общие принципы перегрузки операторов

Едва ли не любой из существующих бинарных и унарных операторов языка C++ можно перегрузить. Ниже приведен список операторов, допускающих перегрузку.

+	-	*	/	%	^	&		~	!	=	<	>	+=
-=	*=	/=	%=	^=	&=	=	<<	>>	>>=	<<=	==	!=	<=
>=	&&		++	--	,	->*	->	->	()	[]	new	delete	

Перегрузка оператора для класса определяет смысл указанной операции при ее использовании в выражении, содержащем хотя бы один объект класса. Тем не менее, нельзя изменить *синтаксис* оператора. И тем более нельзя изменить приоритет операторов, их ассоциативность или число операндов. Также нельзя переопределить оператор, если он используется в выражении, содержащем данные только встроенных типов. Пример перегрузки бинарного оператора вы уже видели. Можно перегрузить и унарный оператор, используя функцию-член класса *без* параметров.

```
// задается в определении класса CCurrency:
CCurrency operator- () // унарный оператор - {отрицание}
```

```

{
return CCurrency (-Dollars, Cents);
}

```

Унарный оператор можно перегрузить, используя дружественную функцию, которая имеет единственный параметр и не входит в класс. В приведенном ниже примере предполагается, что знак денежной суммы определяется переменной Dollars. Например, величина -5,75\$ будет сохранена в виде значения -5 в переменной Dollars и значения +75 в переменной Cents.

```

// определяется глобально:
CCurrency operator- (CCurrency &Curr)
{
return CCurrency (-Curr.Dollars, Curr.Cents);
}

```

При перегрузке некоторых операторов (например, “++”) нужно соблюдать специальные правила. В следующем параграфе рассмотрены основные правила перегрузки оператора присваивания. Информация о перегрузке других специальных операторов приведена в справочной системе.

Перегрузка операторов с помощью библиотеки MFC рассмотрена в части III данной книги. Принципы перегрузки операторов с использованием библиотеки iostream (например, часто встречающейся операции <<) описаны в справочной системе.

## Перегрузка оператора присваивания

В программах, написанных на языке C++, допустимо присваивание одного объекта класса другому объекту того же класса. По умолчанию компилятор обрабатывает оператор присваивания последовательным копированием всех переменных-членов. Например, следующий фрагмент

```

CCurrency Money1 (12, 95);
CCurrency Money2;

Money2 = Money1;

```

вызовет копирование переменной Money1.Dollars в Money2.Dollars, а Money1.Cents – в Money2.Cents (аналогично в последней версии языка C происходит присваивание одной структуры другой). Можно создать временный объект класса и присвоить его объявленному объекту, что является удобным способом повторной инициализации существующего объекта класса.

```

CCurrency Money (85, 25); // создание объекта
Money.PrintAmount ();    // печать его содержимого
Money = CCurrency (24, 65); // создание временного объекта и
                           // присваивание его существующему
                           // объекту

```

Для некоторых классов стандартная операция присваивания может не подходить. В этом случае следует перегрузить оператор присваивания. Например, рассмотрим предназначенный для хранения и вывода сообщений класс CMessage:

```

#include <string.h>
#include <iostream.h>
class CMessage
{
private:
char *Buffer; // хранит строку сообщения

public:
CMessage()

```

```

    {
        Buffer = new char('\0'); // инициализирует переменную
                                // Buffer пустой строкой
    }
    ~CMessage()
    {
        delete[] Buffer;          // освобождает память
    }
    void Display()
    {
        cout << Buffer << '\n'; // печатает сообщение
    }
    void Set (char *String)       // сохраняет новую строку сообщения
    {
        delete [] Buffer;
        Buffer = new char [strlen (String) + 1];
        strcpy (Buffer, String);
    }
};

```

При таком определении класса, если один объект класса CMessage присвоить другому, то переменная-член Buffer *обоих* объектов будет указывать на один и тот же блок памяти. Если затем вызвать функцию-член Set для изменения сообщения, хранимого в одном объекте, то этот блок памяти освободится, вследствие чего в переменной Buffer другого объекта будет храниться указатель на случайные данные. То же самое произойдет, если один объект будет уничтожен раньше другого, так как деструктор освобождает блок памяти, на который указывает переменная Buffer. Чтобы обеспечить корректное присваивание одного объекта класса CMessage другому, добавим следующую функцию-оператор в определение класса.

```

class CMessage
{
    // другие объявления:

public:
    void operator = (const CMessage &Message)
    {
        delete [] Buffer;
        Buffer = new char [strlen (Message.Buffer) + 1];
        strcpy (Buffer, Message.Buffer);
    }

    // другие объявления:
}

```

Перегруженный оператор “=” (в отличие от простого копирования из объекта-источника в объект-приемник *адреса* блока памяти, хранящегося в поле Buffer) создает совершенно новый блок памяти для объекта-приемника, а затем копирует строку в память. Таким образом, каждый объект имеет собственную копию строки. После этого можно безопасно присвоить один объект класса CMessage другому:

```

void main ()
{
    CMessage Message1;
    Message1.Set ("initial Message1 message");
    Message1.Display ( );
}

```

```

CMessage Message2;
Message2.Set ("initial Message2 message");
Message2.Display ( );

Message1 = Message2;
Message1.Display ( );
}

```

На печать будет выведено

```

initial Message1 message
initial Message2 message
initial Message2 message

```

Перегружая оператор “=”, необходимо использовать *функции-члены класса*. Нельзя использовать дружественные функции, которые не являются членами класса. Когда функция operator= определена так как показано в предыдущем коде, в выражение можно включить только *единственный* оператор присваивания. Однако если эта функция возвращает ссылку на объект-приемник класса CMessage, как в следующем примере,

```

CMessage & operator= (const CMessage &Message)
{
    delete [] Buffer;
    Buffer = new char [strlen (Message.Buffer) + 1];
    strcpy (Buffer, Message.Buffer);
    return *this;
}

```

можно объединить несколько операторов присваивания в строке.

```

void main ()
{
    CMessage Message1;
    CMessage Message2;
    CMessage Message3;

    Message1.Set ("hello");
    Message3 = Message2 = Message1;
}

```

В результате будут выполнены последовательные операции присваивания справа налево. В данном примере первой вызывается функция-оператор “=”, присваивающая объект Message1 объекту Message2. Она возвращает ссылку на Message2, содержащий копию строки “hello”. Далее компилятор вызывает функцию-оператор, присваивающую ссылку на Message2 объекту Message3. В результате все три объекта класса CMessage имеют отдельные копии строки “hello”.

Обратите внимание: функция-оператор “=” может возвращать объект класса CMessage, а не ссылку на него. Однако при этом генерируется избыточная операция копирования и, следовательно, несколько снижается эффективность программы. В окончательном варианте функции operator= должна быть предусмотрена защита от случайного вызова, присваивающего объект самому себе. Такое действие приведет к удалению содержимого буфера Buffer с последующей попыткой скопировать из него строку. Если объект присваивается самому себе, адрес объекта-источника (&Message) будет таким же, как и адрес текущего объекта (this). Если функция определит, что выполняется самоприсваивание, то она сразу же, не выполняя операцию копирования, должна вернуть значение.

```

CMessage & operator= (const CMessage &Message)
{
    if (&Message == this)
        return *this;
    delete [] Buffer;
    Buffer = new char [strlen (Message.Buffer) + 1];
    strcpy (Buffer, Message.Buffer);
    return *this;
}

```

Полный листинг класса CMessage с окончательной версией функции operator= приведен ниже.

```

#include <string.h>
#include <iostream.h>

class CMessage
{
private:
    char *Buffer;                // хранит строку сообщения

public:
    CMessage()
    {
        Buffer = new char ( ' \0' ) ; // инициализирует переменную
        // Buffer пустой строкой
    }
    ~CMessage ( )
    {
        delete [] Buffer;        // освобождает место в памяти
    }
    void Display ( )
    {
        cout << Buffer << '\n'; // выводит сообщение
    }
    void Set (char *String)      // сохраняет новую строку сообщения
    {
        delete [] Buffer;
        Buffer = new char [strlen (String) + 1];
        strcpy (Buffer, String);
    }
    CMessage & operator= (const CMessage &Message)
    {
        if (&Message == this)
            return *this;
        delete [] Buffer;
        Buffer = new char [strlen (Message.Buffer) + 1];
        strcpy (Buffer, Message.Buffer);
        return *this;
    }
};

```

## ***Конструкторы копирования и преобразования***

Существуют конструкторы с дополнительными параметрами и стандартными значениями, которые, следовательно, *могут* быть вызваны с единственным параметром.

- Если единственный (или первый) параметр конструктора является ссылкой на тот же тип данных, что и класс, то конструктор называется *конструктором копирования*.
- Если единственный (или первый) параметр конструктора имеет тип, отличающийся от класса конструктора, то такой конструктор называется *конструктором преобразования*.

Для каждого из этих конструкторов особые свойства будут рассмотрены отдельно. Сначала необходимо сформулировать *общее свойство конструкторов с единственным параметром*. Такие конструкторы позволяют инициализировать объект, используя знак равенства в самом определении объекта в отличие от синтаксиса традиционного конструктора. Например, если для класса CTest конструктор определен так

```
CTest (int Parm)
{
    // код конструктора ...
}
```

значит можно создать объект, используя

```
CTest Test (5);
```

или эквивалентный оператор

```
CTest Test = 5;
```

Знак равенства – альтернативный способ передачи единственного значения конструктору. Это операция *инициализации*, а не *присваивания*, поэтому перегрузка оператора “=” на выполнение данной операции не влияет.

## Конструктор копирования

Конструктор с единственным параметром, тип которого определен как ссылка на тип класса, называется *конструктором копирования класса*.

```
class CTest
{
    // ...
public:
    CTest (const CTest &Test)
    {
        // использует члены объекта Test для инициализации
        // нового объекта класса CTest ...
    }

    // ...
}
```

Параметр должен быть *ссылкой* на объект, а не самим объектом (пояснение см. ниже). Если конструктор копирования класса не определен, то компилятор генерирует его неявно. Конструктор, генерируемый компилятором, инициализирует новый объект, выполняя операцию поэлементного копирования переменных существующего объекта класса, передаваемого как параметр. Соответственно, используя объект того же типа, всегда можно инициализировать его, даже если конструктор копирования в классе не определен. Например, даже если класс CCurrency, рассмотренный ранее, не содержит конструктор копирования, то все равно будет корректной следующая инициализация:

```
CCurrency Money1 (59, 71);

CCurrency Money2 (Money1);
CCurrency Money3 = Money1;
```



Для инициализации объекта Money2, как и Money3, вызывается конструктор копирования, сгенерированный компилятором. В результате данной инициализации оба объекта (Money2 и Money3) будут содержать те же значения, что и Money1 (т.е. переменная Dollars будет равна 59, а Cents – 71).

Если необходимо инициализировать новые объекты с помощью существующих объектов того же типа, но операция поэлементного копирования, выполняемая конструктором копирования, сгенерированным компилятором, не подходит для создаваемого класса, то определите собственный конструктор копирования. Например, класс CMessage, рассмотренный выше в этой главе, *нельзя* инициализировать, используя простое копирование членов класса, так как он содержит переменные, являющиеся указателями на блок памяти. Для данного класса можно добавить конструктор копирования такого вида.

```
class CMessage
{
// ...

public:
    CMessage (const CMessage &Message)
    {
        Buffer = new char [strlen (Message.Buffer) + 1];
        strcpy (Buffer, Message.Buffer);
    }

// ...
};
```

Этот конструктор копирования позволяет инициализировать объекты, как показано в следующем примере.

```
CMessage Message1;
Message1.Set ("hello");

CMessage Message2 (Message1); // используется конструктор
                               // копирования
CMessage Message3 = Message1; // используется конструктор
                               // копирования
```

Компилятор автоматически вызывает конструктор копирования класса также в следующих двух случаях:

- при *передаче* объекта класса в качестве параметра функции;
- при *возврате* функцией объекта класса.

Рассмотрим функцию-оператор в качестве примера.

```
CCurrency operator+ (CCurrency Curr)
{
    return CCurrency (Dollars + Curr.Dollars, Cents + Curr.Cents);
}
```

Заданный у функции параметр Curr является объектом класса CCurrency. При каждом вызове функции он должен создаваться и *инициализироваться* с помощью объекта, передаваемого в функцию. Для инициализации параметра компилятор инициализирует вызов конструктора копирования, определенного явно или сгенерированного компилятором. Так как компилятор вызывает конструктор копирования каждый раз при передаче объекта класса в функцию, объект класса нельзя передавать как первый параметр в сам конструктор копирования. Напротив – необходимо передать *ссылку* на него. Передача объекта послужит причиной бесконечной рекурсии (в настройках Visual Studio можно установить поиск таких ошибок).

Тип значения, возвращаемого функцией, является объектом класса `CCurrency`, поэтому при вызове функции компилятор генерирует временный объект класса `CCurrency` и использует значение, определенное в операторе `return`, для инициализации этого объекта (для этого компилятор также вызовет конструктор копирования).

Можно исключить непроизводительные издержки при вызове конструктора копирования передачи и возвратом *ссылок* на объекты вместо самих объектов (если это возможно). Функция `operator+`, рассмотренная выше, *не* может возвращать ссылку на временный объект класса `CCurrency`. Передача ссылки на объект, который перестает существовать после выхода из функции, является плохим стилем программирования и может приводить к ошибкам исполнения.

## Конструктор преобразования

В языке C++ *конструктор преобразования класса* – это конструктор с единственным параметром, тип которого отличается от типа класса. Такой конструктор обычно инициализирует новый объект, используя данные существующих переменных или объектов другого типа. Например, в класс `CCurrency` можно добавить конструктор преобразования, который позволит инициализировать объект, используя сумму долларов и центов, выраженную одним действительным числом в формате с плавающей запятой. Заметьте: конструктор округляет число центов, сохраняемое в параметре `DolAndCen`, до ближайшего целого числа. Кроме того, обратите внимание: конструктор выполняет явное преобразование типов данных, используя разрешенный в языке C++ альтернативный синтаксис, отличающийся от традиционной записи операции приведения типов. Например, в конструкторе используется выражение `long (DolAndCen)`, а не традиционная запись `(long) DolAndCen`. Безусловно, в C++ допустимо использовать приведение типов, хотя новый синтаксис некоторых выражений облегчает понимание программы.

```
class CCurrency
{
// ...
public:
    // ...
    CCurrency (double DolAndCen)
    {
        Dollars = long (DolAndCen);
        Cents = int ((DolAndCen - Dollars) * 100.0 + 0.5);
    }
// ...
};
```

Приведенный выше конструктор преобразования позволяет инициализировать объекты класса `CCurrency`:

```
CCurrency Bucks1(29.63);
CCurrency Bucks2 = 43.247; // будет округлено до 43.25
CCurrency Bucks3 (2.0e9); // почти максимально допустимое
                        // число долларов
CCurrency *Bucks = new CCurrency (534.85);
```

Определение класса `CCurrency` содержит конструктор такого вида:

```
CCurrency (long Dol, int Cen)
{
    SetAmount (Dol, Cen);
}
```

Его можно переделать в конструктор преобразования, добавив стандартное значение для второго параметра.

```
CCurrency (long Dol, int Cen = 0)
{
    SetAmount (Dol, Cen);
}
```

Теперь этот конструктор может принимать один параметр. Такой конструктор можно использовать для инициализации объектов класса `CCurrency`, задавая только количество долларов. Символ `L` добавляется в конец каждой целочисленной константы, задавая ей тип `long`. Если данный символ не указан, константа будет рассматриваться как константа типа `int`. В этой ситуации компилятор не определит, преобразовывать ли тип `int` в `double`, чтобы вызвать конструктор для `double`, либо преобразовывать в `long` с тем, чтобы вызвать конструктор для `long`. Такая ситуация называется *неоднозначным вызовом* перегруженной функции и порождает ошибку компиляции. Преобразования типа `int` в `double` и `long` являются *стандартными преобразованиями*. Хотя переменные типов `int` и `long` имеют одинаковый размер в Visual C++, они рассматриваются как различные типы, требующие преобразования.

```
// установка значений: Dollars = 25 и Cents = 0
CCurrency Dough = 25L;
CCurrency *PDough = new CCurrency (25L);
```

Ниже приведен пример с добавлением в класс `CMessage` конструктора преобразования:

```
class CMessage
{
    // ...
public:
    // ...

    CMessage (const char *String) .
    {
        Buffer = new char [strlen (String)+1];
        strcpy (Buffer, String);
    }

    // ...
};
```

Имея такой конструктор, объект можно инициализировать одной строкой, как показано ниже:

```
CMessage Note = "do it now";
CMessage *PNote = new CMessage ("remember!");
```

Компилятор вызывает соответствующий конструктор преобразования и для преобразования переменной-члена какого-либо другого типа в объект класса. Другими словами, конструктор преобразования указывает компилятору, как преобразовывать объекты или переменные различных типов в объект данного класса. Например, два конструктора преобразования класса `CCurrency` позволят присвоить существующему объекту этого класса значение типа `double` или `long`.

```
CCurrency Bucks;

Bucks = 29.95;
Bucks = 35L;
```

В обоих присваиваниях компилятор в первую очередь преобразовывает константу в объект класса `CCurrency`, используя соответствующий конструктор преобразования, а затем присваивает его

объекту Bucks класса CCurrency. Для исключения неоднозначного вызова конструктора преобразования во втором операторе присваивания необходим символ L.

Предположим, что функция имеет параметр типа CCurrency. Так как для класса CCurrency определены оба конструктора преобразования, данной функции можно передать значение типа double или long, как объект класса CCurrency. Компилятор вызовет для преобразования аргумента в объект класса CCurrency соответствующий конструктор.

```
void Insert (CCurrency Dinero);
```

Создание конструкторов преобразования дает важное преимущество. Оно состоит в применении перегруженных операторов, определенных для класса, и в том, что пропадает необходимость в написании отдельной функции-оператора для каждой ожидаемой комбинации операндов. Например, в настоящий момент существует три функции operator+ в классе CCurrency:

```
class CCurrency
{
    // ...
public:
    // ...
    CCurrency operator+ (const CCurrency &Curr)
    {
        return CCurrency (Dollars + Curr.Dollars, Cents + Curr.Cents);
    }
    CCurrency operator+ (long Dol)
    {
        return CCurrency (Dollars + Dol, Cents);
    }
    friend CCurrency operator+ (long Dol, const CCurrency &Curr);
    // ...
};
```

Благодаря конструктору, преобразующему значения типа long в объекты класса CCurrency, можно исключить две функции operator+ и переписать функцию, которая не является членом класса CCurrency.

```
class CCurrency
{
    // ...
public:
    // ...
    friend CCurrency operator+ (const CCurrency &Curr1,
                                const CCurrency &Curr2);
    // ...
};
CCurrency operator+ (const CCurrency &Curr1,
                    const CCurrency &Curr2)
{
    return CCurrency (Curr1.Dollars + Curr2.Dollars,
                      Curr1.Cents + Curr2.Cents);
}
```

Приведенная функция-оператор может выполнять операции сложения в самых разных вариантах. Ниже, во втором и третьем выражениях компилятор сначала преобразовывает константу типа long

в объект `CCurrency`, используя соответствующий конструктор преобразования, а затем вызывает дружественную функцию-оператор для сложения двух объектов.

```
CCurrency Bucks1 (39, 95);
CCurrency Bucks2 (149, 85);
CCurrency Bucks3;

Bucks3 = Bucks1 + Bucks2;
Bucks3 = Bucks1 + 10L;
Bucks3 = 15L + Bucks1;
```

Теперь `CCurrency` имеет конструктор преобразования с параметром типа `double`, поэтому значения с плавающей запятой также могут обрабатываться единственной функцией-оператором.

```
CCurrency Bucks1 (39, 95);
CCurrency Bucks2 (149, 85);
CCurrency Bucks3;

Bucks3 = Bucks1 + 29.51;
Bucks3 = 87.64 + Bucks1;
```

Конструкторы преобразования класса указывают компилятору, как преобразовать какой-либо тип данных в объект класса. Кроме того, можно написать *функцию преобразования*, предоставляющую компилятору способ преобразования объекта класса в данные какого-либо другого типа. Функция преобразования определяется как функция-член и подобна функции-оператору, используемой для перегрузки стандартных операторов. Однако функция преобразования менее универсальна, чем аналогичный конструктор, и должна быть тщательно продумана во избежание неопределенности. Информация по созданию таких функций приведена в справочной системе.

Окончательные версии классов `CCurrency` и `CMessage`, в том числе новые конструкторы копирования и преобразования, приведены в листингах 6.1 и 6.2.

---

#### Листинг 6.1

```
// CCurr.h: файл заголовков класса CCurrency
#include <string.h>
#include <iostream.h>
class CCurrency
{
private:
    long Dollars;
    int Cents;
public:
    CCurrency () // конструктор по умолчанию
    {
        Dollars = Cents = 0;
    }
    CCurrency (long Dol, int Cen = 0) // конструктор преобразования
    {
        SetAmount (Dol, Cen);
    }
    CCurrency (double DolAndCen) // конструктор преобразования
    {
        Dollars = long (DolAndCen);
        Cents = int ((DolAndCen - Dollars) * 100.0 + 0.5);
    }
    void GetAmount (long *PDol, int *PCen)
```

```

    {
        *PDol = Dollars;
        *PCen = Cents;
    }
void PrintAmount ( )
{
    cout.fill ('0');
    cout.width (1) ;
    cout << '$' << Dollars << '.';
    cout.width (2);
    cout << Cents << '\n';
}
void SetAmount (long Dol, int Cen)
{
    Dollars = Dol + Cen / 100;
    Cents = Cen % 100;
}
friend CCurrency operator+ (const CCurrency &Curr1, const CCurrency &Curr2);
};

CCurrency operator+ (const CCurrency &Curr1, const CCurrency &Curr2)
{
    return CCurrency (Curr1.Dollars + Curr2.Dollars, Curr1.Cents + Curr2.Cents);
}

```

---

## Листинг 6.2

```

// CMess.h: файл заголовков класса CMessage
#include <string.h>
#include <iostream.h>

class CMessage
{
private:
    char *Buffer;
public:
    CMessage ( ) // конструктор по умолчанию
    {
        Buffer = new char ('\0');
    }
    CMessage (const CMessage &Message) // конструктор копирования
    {
        Buffer = new char [strlen (Message.Buffer) + 1];
        strcpy (Buffer, Message.Buffer);
    }
    CMessage (const char *String) // конструктор преобразования
    {
        Buffer = new char [strlen (String) + 1];
        strcpy (Buffer, String);
    }
    ~CMessage ( )
    {
        delete [] Buffer;
    }
    void Display ()

```

```

    {
        cout << Buffer << '\n';
    }
    void Set (char *String)
    {
        delete [] Buffer;
        Buffer = new char (strlen (String) + 1);
        strcpy (Buffer, String);
    }
    CMessage & operator= (const CMessage &Message)
    {
        if (&Message == this)
            return *this;
        delete [] Buffer;
        Buffer = new char [strlen (Message.Buffer) + 1];
        strcpy (Buffer, Message.Buffer);
        return *this;
    }
};

```

## Инициализация массивов

Мы уже рассматривали массивы объектов в гл. 4. Можно использовать стандартный синтаксис, унаследованный из языка C, для инициализации массива любого типа:

```
int Table [5] = {1, 2, 3, 4, 5};
```

Но здесь необходимо помнить о следующем ограничении. При инициализации массива объектов можно присвоить *единственное* значение каждому из его элементов и *нельзя* передать в конструктор группу значений. Однако конструкторы копирования и преобразования позволяют инициализировать объекты, используя единственный параметр, следовательно, бывают полезны при инициализации массивов объектов. Рассмотрим инициализацию массива объектов класса CCurrency различными способами в качестве примера:

```

CCurrency Money (95, 34);

CCurrency MoneyTable [5] =
{
    Money,
    CCurrency (15, 94),
    10L,
    12.23
};

```

В этом примере первый элемент (MoneyTable [0]) инициализируется существующим объектом класса CCurrency с помощью конструктора копирования; второй – инициализируется конструктором, создающим временный объект класса CCurrency (затем данный объект используется для инициализации элемента массива также конструктором копирования); третий – инициализируется константой типа long с помощью конструктора преобразования с параметром типа long; четвертый – константой типа double с помощью другого конструктора преобразования. Последний элемент массива инициализируется *неявно*; следовательно, для него компилятор вызывает конструктор по умолчанию. Как упоминалось в гл. 3, *нельзя* явно инициализировать массив объектов, созданных динамически с помощью оператора new. Вместо этого компилятор для каждого элемента вызывает конструктор по умолчанию.

## Резюме

---

Рассмотрена методика написания специальных функций для настройки объектов классов.

- *Функция-оператор* позволяет *перегрузить* стандартный оператор языка C++, т. е. определяет, каким образом указанная операция применима к объектам определенного класса. Функция-оператор может либо быть функцией-членом класса, либо ею не быть. Во втором случае она обычно объявляется внутри класса как *дружественная*, что позволяет ей получить прямой доступ к закрытым и защищенным переменным класса.
- *Конструктор копирования* класса имеет единственный параметр, представляющий собой ссылку на существующий объект того же класса. Если конструктор копирования не определен, компилятор сгенерирует его автоматически. При этом инициализация нового объекта будет осуществляться простым копированием всех переменных-членов из существующего объекта. Если такой механизм работы не подходит для вашего класса, определите собственный конструктор копирования. Компилятор вызывает конструктор копирования при передаче объекта класса в качестве параметра функции или при возврате функцией объекта класса.
- *Конструктор преобразования* класса имеет единственный параметр, тип которого отличается от типа класса. Его можно использовать для инициализации нового объекта с использованием существующей переменной или объекта другого типа. Компилятор также автоматически вызывает данный конструктор при необходимости преобразования переменной или объекта какого-либо типа в объект класса.
- *Действия с классами.* Язык C++ поддерживает типы, определенные пользователем (т.е. классы), почти так же, как и встроенные типы. Работая с классами, можно *объявлять* объекты класса так, как объявляют переменные встроенных типов. Более того, эти объекты будут подчиняться стандартным *правилам видимости*, установленным для переменных. Можно *инициализировать* объекты классов при их определении, подобно переменным встроенных типов. Допустимо *использование стандартных операторов* языка C++ для объектов классов так же, как и для переменных встроенных типов.
- Компилятор автоматически выполняет преобразование класса точно так же, как он выполняет стандартное преобразование встроенных типов. Так как компилятору не известна информация об особенностях определяемого класса, для полной поддержки этого типа данных необходимо сообщить, как инициализировать объекты этого типа (данная информация определяется в *конструкторе копирования*), как использовать стандартные операторы с объектами этого типа (с помощью *перегрузки операторов*), как преобразовывать этот тип данных в другие типы данных (с помощью *конструкторов и функций преобразования*).



# Глава 7

## Шаблоны в C++

---

- Шаблоны функций
- Шаблоны классов

Шаблоны языка C++ облегчают генерацию семейств функций или классов, оперирующих множеством данных различных типов. При этом не возникает необходимости создавать отдельную функцию или класс для каждого типа. Шаблоны – это способ написания единственного родового определения функции или класса, которое компилятор автоматически транслирует в специальную версию функции или класса для каждого из используемых программой типов данных.

### Шаблоны функций

---

Используя механизм шаблонов функций языка C++ можно создать единственное общее определение функции, использующееся с различными типами данных. Вспомним параграф “Перегруженные функции” гл. 3, в котором указывалось, что для использования одной и той же функции с различными типами данных нужно определить отдельную перегруженную версию этой функции для каждого типа. Если требуется функция, возвращающая абсолютную величину значения как типа `int`, так и типа `double`, то нужно написать две перегруженные функции (см. пример в гл. 3):

```
int Abs(int N)
{
    return N < 0 ? -N : N;
}

double Abs (double N)
{
    return N < 0.0 ? -N : N;
}
```

Шаблон языка C++ позволяет создать *единственное* определение, автоматически обрабатывающее значения типа `int`, `double` или любого другого подходящего типа. Такой шаблон выглядит так:

```
template <class T> T Abs (T N)
{
    return N < 0 ? -N : N;
}
```

Здесь идентификатор `T` является *параметром типа* (*type parameter*). Он переопределяет тип переменной или константы, передаваемой при вызове функции. Если программа вызывает функцию `Abs` и передает ей значение типа `int`, например, так как показано ниже,

```
cout << "absolute value of -5 is " << Abs (-5);
```

компилятор сгенерирует версию функции, в которой идентификатор `T` имеет тип `int`, и добавит в программу вызов данной версии функции. Генерируемая функция будет эквивалентна определенной явно функции.

```
int Abs (int N)
{
    return N < 0 ? -N : N;
}
```

Если же программа вызывает функцию Abs и передает ей значение типа double:

```
double D = -2.54;
cout << "absolute value of D is " << Abs (D);
```

компилятор автоматически сгенерирует версию функции, в которой тип идентификатора T будет заменен double, и добавит в программу вызов данной функции. Эта версия функции эквивалентна такой:

```
double Abs (double N)
{
    return N < 0 ? -N : N;
}
```

Аналогичным образом компилятор генерирует дополнительные версии функции для каждого вызова, в котором указывается новый числовой тип данных, например short или float. Генерация новой версии функции называется *созданием экземпляра (instantiating)* шаблона функции. При определении шаблона нужно использовать спецификаторы template и class вместе с угловыми скобками, как показано в приведенном выше примере: Для параметра типа T можно использовать любой корректный идентификатор имени, а в угловые скобки можно включать несколько параметров типа.

В шаблоне функции не следует путать понятия *параметр функции* и *параметр типа*. Параметр функции представляет собой значение, передаваемое в функцию при *выполнении* программы. Параметр типа, напротив, задает тип аргумента, передаваемого в функцию, и полностью обрабатывается при *компиляции*. Обратите внимание: в контексте определения шаблона спецификатор class в угловых скобках ссылается не на конкретный тип данных class, а на *любой* тип данных, фактически передаваемых при вызове (как встроенный, так и определенный программистом тип данных).

Определение шаблона само по себе не вызывает генерацию кода компилятором. Компилятор генерирует код функции только при ее фактическом вызове. Поэтому он не может обнаружить *ошибки в тексте шаблона* функции до вызова этой функции в исходном файле. Первый же вызов с определенным типом данных приводит к генерации компилятором кода соответствующей версии функции. Последующие вызовы с указанием тех же типов данных *не* сгенерируют дополнительные копии функции, а лишь вызовут ее первоначальную копию. Однако компилятор сгенерирует новую версию функции, если тип параметра не совпадет *в точности* с типом в предыдущем вызове. Рассмотрим случай, когда программа передает в шаблон функции параметр типа long, а компилятор генерирует соответствующую версию функции. Если затем программа передаст параметр типа int, компилятор сгенерирует полностью новую версию функции для обработки типа int. Он *не будет* для использования кода первой версии функции выполнять стандартное преобразование int в long.

По сравнению с перегруженными функциями у шаблонов есть ряд преимуществ, одно из которых состоит в том, что при использовании шаблона нет необходимости предвидеть, к какой версии функции произойдет обращение в программе. Вместо этого в программу включается единственное определение шаблона, а компилятор автоматически генерирует и сохраняет только те версии функции, которые будут фактически вызываться.

Рассмотрим пример шаблона функции, который генерирует функции, возвращающие большее из двух значений одинакового типа:

```
template <class T> T Max (T A, T B)
{
    return A > B ? A : B;
}
```

Так как оба параметра определены как имеющие тип идентификатора T, в вызове функции оба передаваемых параметра должны быть только одного типа. В противном случае компилятор не определит, какой тип соответствует параметру идентификатора T – тип первого или второго параметра. (Вспомните: значение параметра T определяется типом передаваемых параметров.) Поэтому такие вызовы функции допустимы.

```
cout << "The greater of 10 and 5 is " << Max (10, 5) << '\n';
cout << "The greater of 'A' and 'M' is " << Max ('A', 'M') << '\n';
cout << "The greater of 2.5 and 2.6 is " << Max (2.5, 2.6) << '\n';
```

Но приведенный далее вызов недопустим.

```
cout << "The greater of 15.5 and 10 is " << Max (15.5, 10)
<< '\n'; // ОШИБКА!
```

Компилятор *не* преобразует второй параметр int в double для приведения типов, хотя это стандартное преобразование.

Для передачи параметров различных типов, нужно определить шаблон функции такого вида.

```
template <class Type1, class Type2> Type1 Max (Type1 A, Type2 B)
{
    return Type1 (A > B ? A : B);
}
```

Здесь Type1 обозначает тип значения, передаваемого в качестве первого, а Type2 – второго параметров. Для новой версии шаблона следующий оператор является допустимым.

```
cout << "The greater of 15.5 and 10 is " << Max (15.5, 10)
<< '\n'; // теперь допустимо
```

Он напечатает значение 15.5. Заметьте: в новом определении Max параметр Type1 появляется внутри тела функции, где он используется для приведения возвращаемого значения к типу первого параметра функции (если это необходимо).

```
return Type1 (A > B ? A : B);
```

В языке C++ параметр типа можно использовать в любом месте кода, в котором используется имя типа. Так как возвращаемое значение преобразуется к типу первого параметра, то для предыдущего примера при изменении порядка параметров

```
cout << "The greater of 15.5 and 10 is " << Max (10, 15.5)
<< '\n';
```

результат сравнения (15.5) будет округлен и составит 15.

Каждый параметр типа, встречающийся внутри символов "<" и ">", должен также появляться в списке параметров функции. Значит, не допустимо следующее определение шаблона:

```
// ОШИБКА: список параметров функции должен
//          включать Type2 как параметр типа:
template <class Type1, class Type2> Type1 Max (Type1 A, Type1 B)
{
    return A > B ? A : B;
}
```

Компилятор при таком определении, встретив вызов функции, не сможет определить значение идентификатора Type2. Даже если идентификатор Type2 не использован, это – ошибка.

## Переопределение шаблона

В программах на C++ каждая версия функции, генерируемая с помощью шаблона, содержит одинаковый базовый код. Единственным изменяемым свойством функции будет значение параметра (или параметров) типа. Однако для отдельного параметра (или параметров) типа можно обеспечить специальную обработку. Для этого определяется обычная функция языка C++ с тем же именем, что и шаблон функции, но использующая уже имеющиеся типы данных, а не параметры типов. Обычная функция *переопределяет* шаблон. Т.е., если при интерпретации вызова компилятор обнаруживает, что типы переданных параметров соответствуют спецификации обычной функции, то он вызовет ее, а не сгенерирует функцию по шаблону. В качестве примера можно определить новую версию функции `Max`, которая будет работать с экземплярами класса `CCurrency`, описанного в параграфе “Перегрузка операторов” гл. 6. Вспомним: объект класса `CCurrency` хранит денежную сумму в виде числа долларов и числа центов. Очевидно, что код, определенный в шаблоне `Max`, не подходит для сравнения денежных величин, хранимых в двух объектах `CCurrency`. Следовательно, можно включить в программу *дополнительно* к уже имеющемуся шаблону `Max` приведенную ниже версию функции.

```
CCurrency Max (CCurrency A, CCurrency B)
{
    long DollarsA, DollarsB;
    int CentsA, CentsB;

    A.GetAmount (&DollarsA, &CentsA);
    B.GetAmount (&DollarsB, &CentsB);

    if (DollarsA > DollarsB || DollarsA == DollarsB
        && CentsA > CentsB)
        return A;
    else
        return B;
}
```

Теперь, если программа вызовет функцию `Max`, передав ей два объекта класса `CCurrency`, компилятор иницирует вызов приведенной выше функции вместо создания экземпляра функции по шаблону `Max`. Так, для случая:

```
CCurrency Bucks1 (29, 95);
CCurrency Bucks2 (31, 47);
Max (Bucks1, Bucks2).PrintAmount ();
```

результатом будет

```
$31.47
```

В данной конкретной задаче вместо переопределения функции `Max` для сравнения двух объектов `CCurrency` можно перегрузить оператор “>” (см. гл. 6), чтобы корректно сравнивать значения денежных сумм, сохраняемых в этих двух объектах. Тогда приведенный в примере шаблон `Max`, использующий оператор “>”, будет корректно обрабатывать объекты класса `CCurrency`.

## Шаблоны классов

Создание нового производного класса из существующего позволяет повторно использовать код, избегая при этом ненужного дублирования (см. гл. 5). Допустим, вы сконструировали класс для хранения списка 100 целочисленных значений. Целочисленные значения хранятся в закрытом массиве `Buffer`, а специально написанный конструктор по умолчанию инициализирует нулями все элементы

этого массива. Функции-члены SetItem и GetItem используются для присваивания или получения значений указанных элементов. Базовое определение класса может выглядеть так:

```
class IntList
{
public:
    IntList ();

    int SetItem (int Index, const int &Item);
    int GetItem (int Index, int &Item);
private:
    int Buffer [100];
};
```

Пусть теперь нужно получить подобный класс для хранения списка вещественных значений, структур или большого количества элементов (например, 250). В этих случаях определяется полностью новый класс. *Создание производного* класса от IntList не является решением задачи, так как необходимо не просто добавить несколько новых свойств, а изменить основные типы или константы, на которых основывается построение класса.

Но можно при создании программы написать единственный *шаблон класса*, который будет использоваться для автоматической генерации целого семейства взаимосвязанных классов, предназначенных для хранения данных различных типов и различного числа элементов. Простая версия такого класса может выглядеть так:

```
template <class T, int I> class CList
{
public:
    int SetItem (int Index, const T &Item);
    int GetItem (int Index, T &Item);

private:
    T Buffer [I];
};
```

Здесь T является параметром типа, а I – параметром-константой (точнее, в данном примере – это параметр-константа типа int). Как вы вскоре увидите, фактические значения для параметров T и I устанавливаются при создании определенного экземпляра класса. В его шаблон класса можно включить список из любого числа параметров, ограниченный символами “<” “>” (список должен содержать хотя бы один параметр). Параметры-константы могут иметь любой допустимый тип (не обязательно int, как в приведенном выше примере). Внутри определения класса параметр типа может использоваться в любом месте программы, в котором допустимо использование спецификации типа, а параметр-константа – в любом месте программы, в котором допустимо применение константного выражения описанного типа (в нашем примере int).

Функцию-член SetItem можно определить так, чтобы она возвращала значение 1 при удачном завершении или 0, если недопустимо заданное значение индекса.

```
template <class T, int I> int CList <T, I>::SetItem(int Index,
const T &Item)
{
    if (Index < 0 || Index > I - 1)
        return 0; // ошибка

    Buffer [Index] = Item ;
    return 1 ; // успешное завершение
}
```

Реализация функции, помещенной вне определения шаблона, должна содержать следующие два компонента (в дополнение к компонентам, обычно включаемым в определение функции-члена):

- Определение должно начинаться спецификатором `template`, за которым следует такой же список параметров в угловых скобках, как и в определении шаблона класса (в приведенном примере `template <class T, int I>`).
- За именем класса, предшествующим операции расширения области видимости, должен следовать список имен параметров шаблона (в нашем примере – `CList <T, I>`). Список используется для полного определения типа класса, к которому принадлежит функция.

Если функция-член шаблона реализована вне его определения, то эта реализация должна быть включена в каждый исходный файл программы, содержащий вызов функции. Тогда компилятор сможет сгенерировать код функции из ее определения. В программу с несколькими исходными файлами можно ввести как определение шаблона, так и определения всех его методов в одном файле заголовков, входящем во все исходные файлы. Для функций-членов шаблона класса включение определения функции в несколько исходных файлов *не приводит* к возникновению ошибки “multiple symbol definition” (многократное определение символического имени).

По аналогии метод `GetItem` можно определить следующим образом.

```
template <class T, int I> int CList <T, I>::GetItem
(int Index, T &Item)
{
    if (Index < 0 || Index > I - 1)
        return 0;
    Item = Buffer [Index];
    return 1;
}
```

Идентификатор `I` может использоваться в определении шаблона класса как значение размерности массива, поскольку он является *параметром-константой* шаблона, а не обычным параметром или переменной, которые нельзя использовать для задания размерности массива (язык C++ не позволяет динамически изменять размерность массива во время выполнения программы). Использование параметра-константы допустимо на этапе компиляции, так как значение этого параметра становится константой при запуске программы

## Порождение объектов по шаблонам

Создавая объект по шаблону класса, нужно точно определить значения параметров шаблона. По аналогии с приведенным выше примером класса, не являющегося шаблоном, можно создать экземпляр шаблона класса `CList` для хранения в списке не более 100 значений целочисленного типа.

```
CList <int, 100> IntList;
```

Приведенный пример объявления описывает `IntList` как экземпляр версии `CList`, в которой каждое вхождение параметра `T` заменяется типом `int`, а каждое вхождение параметра `I` – константой `100`. В результате, в полученном объекте элемент `Buffer` будет определен как массив из 100 целочисленных значений типа `int`, а функциям-членам `SetItem` и `GetItem` будут переданы ссылки на значения типа `int` (в соответствии со вторым параметром). Обратите внимание: согласно списку параметров в определении шаблона `CList` (т.е. `<class T, int I>`) при создании объекта необходимо присвоить первому параметру шаблона корректное описание типа, а второму – константное значение типа `int` (или другое значение, которое можно преобразовать в тип `int`) или константную переменную типа `int`, инициализируемую константным выражением. Нельзя передать второй параметр как неконстантную переменную или как константную переменную, инициализируемую другой переменной.

После создания объекта и задания всех типов в списке параметров, компилятор автоматически обеспечивает работу с этими типами при последующем обращении к переменным-членам или вызове функций-членов класса. Таким образом, в объекте `IntList` нужно передать два значения типа `int` в метод `SetItem`.

```
IntList.SetItem (0, 5); // первому элементу списка
                      // присваивается целое число
```

Для создания объекта, хранящего список строк, экземпляр `CList` можно определить следующим образом.

```
CList <char *, 25> StringList;
```

Для присваивания строки какому-либо элементу списка следует в функции `SetItem` передать указатель на массив типа `char` второму параметру.

```
StringList.SetItem (0, "Mike"); // присваивает строку первому
                               // элементу списка
```

Для хранения списка значений типа `double` объект создается так.

```
CList <double, 25> *DoubleList;
DoubleList = new CList <double, 25>;
```

При определении типа указателя `DoubleList`, так же как и при задании типа в операторе `new`, вместе с именем шаблона следует включить список его параметров `<double, 25>`. Список параметров шаблона является неотъемлемой частью определения типа. Имя шаблона адресует *семейство* типов, поэтому одного имени шаблона недостаточно для представления типа.

Объект для хранения структур, объединений или классов можно также создать при условии, что они определены глобально, т.е. вне любой функции. Например, объект для сохранения структур создается в следующем фрагменте.

```
// определение структуры Record на глобальном уровне:
struct Record
{
    char Name [25];
    char Phone [15];
}

void main ()
{
    // создание объекта для хранения списка, содержащего не более
    // 50 структур типа Record:
    CList <Record, 50> RecordList;

    // создание и инициализация экземпляра структуры Record:
    Record Rec =
    {
        "John" ,
        " 287-981-0119 "
    };

    // копирование содержимого объекта Rec в первый элемент списка:
    RecordList.SetItem (0, Rec) ;

    // продолжение функции main ...
}
```

Для функции-члена шаблона класса (как и для шаблона функции), компилятор не генерирует код до фактического вызова функции. Он генерирует различные версии кода для каждого объекта, которому соответствует уникальный набор параметров шаблона. Несмотря на это, можно использовать *явное создание экземпляра*, когда компилятор генерирует код для всех функций-членов класса или отдельных функций-членов без фактического вызова какой-либо функции-члена. Это удобно для создания файла библиотеки с расширением `.lib`, содержащего методы, сгенерированные из шаблонов классов. Информация о явном создании экземпляра шаблона приведена в справочной системе.

## Конструктор в шаблоне функции

В шаблон `CList` следует добавить конструктор для инициализации элементов списка. Чтобы инициализировать массив `Buffer`, конструктор должен получить параметр типа `T`. (Конструктор не может просто инициализировать элементы массива `Buffer` стандартными значениями, например 0, так как ему не известен тип данных этих элементов.) Например, конструктор может быть объявлен в разделе `public` класса `CList` так:

```
// внутри раздела public класса CList:
CList (T InitValue);
```

Можно реализовать конструктор следующим образом:

```
template <class T, int I> CList <T, I>::CList (T InitValue)
{
    for (int N = 0; N<I; ++N)
        Buffer [N] = InitValue;
}
```

Конструктор можно полностью описать внутри определения шаблона. Обратите внимание, что список параметров шаблона (`<T, I>`) не включается, когда название `CList` следует за операцией расширения области видимости. (Разработчики языка C++, по-видимому, посчитали повторение списка параметров излишним.) С помощью такого конструктора можно создать объект для хранения целочисленных значений и одновременно нулевыми значениями инициализировать все элементы списка.

```
CList <int, 100> IntList (0);
```

Аналогично в следующем фрагменте для хранения списка структур типа `Record` создается и инициализируется объект. В создаваемом объекте каждый элемент списка является структурой `Record`, в обоих полях которой записаны нулевые (пустые) строки.

```
struct Record
{
    char Name [25];
    char Phone [15];
};

// ...

Record Rec = {"", ""};
CList <Record, 50> RecordList (Rec);
```

Для класса `CList` определение может включать конструктор по умолчанию (т.е. конструктор без параметров), тогда объект может быть создан без инициализации списка. Вспомните, что если существует конструктор, имеющий один или более параметров, компилятор не генерирует автоматически конструктор по умолчанию. Однако в определение класса `CList` можно явно добавить пустой конструктор

```
CList (){};
```



Для шаблона класса *деструктор* определяется с использованием синтаксиса, подобного синтаксису конструктора. Например, деструктор класса CList объявлен в определении шаблона следующим образом

```
~CList ();
```

и реализован может быть так.

```
template <class T, int I> CList <T, I>::~~CList ()
{
    // код деструктора
}
```

Для создания объектов, хранящих семейства объектов различных типов, библиотека MFC предоставляет набор шаблонов. В гл. 11 рассматривается применение одного из этих шаблонов – CTypedPtrArray.

Окончательное определение класса CList и его методов приведено в листинге 7.1.

---

#### Листинг 7.1

```
// CList.h: файл заголовков шаблона класса CList
template <class T, int I> class CList
{
public:
    CList () {};
    CList (T InitValue);

    int SetItem (int Index, const T &Item);
    int GetItem (int Index, T &Item);

private:
    T Buffer [I];
};

template <class T, int I> CList <T, I>::CList (T InitValue)
{
    for (int N = 0; N < I; ++N)
        Buffer [N] = InitValue;
}

template <class T, int I> int CList <T, I>::SetItem (int Index,
const T &Item)
{
    if (Index < 0 || Index > I - 1)
        return 0;
    Buffer [Index] = Item;
    return 1;
}

template <class T, int I> int CList <T, I>::GetItem (int Index,
T &Item)
{
    if (Index < 0 || Index > I - 1)
        return 0;
    Item = Buffer [Index];
    return 1;
}
```

## Резюме

---

В этой главе рассмотрены шаблоны – относительно новое дополнение языка C++.

- *Шаблон* является специальным видом определения функции или класса, позволяющим генерировать целое семейство взаимосвязанных функций или классов, предназначенных для работы с данными определенного типа.
- *Шаблон функции* определяет родовую функцию, параметры которой принадлежат некоторому множеству различных типов. При обнаружении вызова функции компилятор автоматически генерирует версию функции, соответствующую типу данных, переданных при вызове. Определение единственного шаблона функции – удобная альтернатива определению набора перегруженных функций. В определении шаблона функции *параметры типа* описывают типы переменных, передаваемых при вызове функции. Для специальной обработки данных можно создать обычную функцию, которая имеет такое же имя, как шаблон функции, но использует определенные типы данных, а не параметры типа. Если функция вызывается с этими типами данных, то одноименная функция переопределяется обычной.
- *Шаблон класса*. Чтобы сгенерировать семейство родственных классов для работы с различными типами данных, необходимо определить единственный шаблон класса. При определении шаблона класса можно использовать *параметры-константы* и *параметры типов* вместо конкретных констант или типов. При создании объекта на основе шаблона переопределяются значения параметров-констант и параметров типов. Затем компилятор генерирует экземпляр версии класса, использующий указанные константы и типы.

## Глава 8

# Исключительные ситуации

---

- Программные исключения и их обработка
- Win32-исключения и их обработка

*Исключением* (exception) или *исключительной ситуацией* называется прерывание нормального потока выполнения программы в ответ на непредвиденное или аварийное событие, которое порождается ошибками в аппаратуре, например, делением на ноль или обращением к памяти по некорректному адресу, а также генерируется программно, например, функциями динамической библиотеки или Win32 API при возникновении ситуации, не позволяющей завершить задачу (в частности при получении недействительного указателя или дескриптора). Наконец, исключения допускается генерировать самостоятельно.

Для обработки исключений перечисленных типов язык C++ предоставляет простой механизм. Если обработчик для какого-либо исключения *не предусмотрен*, то стандартный обработчик генерирует сообщение об ошибке и завершает выполнение программы. При обработке исключения можно:

- либо исправить ошибки и сделать возможным *дальнейшее выполнение программы*;
- либо выполнить очистку и *корректно завершить программу*.

Рассмотрение вопроса начинается с объяснения правил обработки исключений, генерируемых в C++, т.е. с помощью оператора `throw`. Далее показано, как обрабатывать исключения Win32, генерируемые аппаратными событиями или функциями Win32 API. (Win32 API являются встроенными функциями Windows (см. гл. 9).) Учтите: механизм обработки исключений в C++ развивается и изменяется подобно средствам работы с шаблонами.

## Программные исключения и их обработка

---

Рассмотрим технику обработки исключений, генерируемых программно с использованием оператора `throw`, за которым следует некоторое значение. Значение это может быть константой, переменной или объектом (т.е. экземпляром класса, структуры или объединения). Оно предназначено для передачи информации об исключении. Эта информация может использоваться обработчиком исключения. Например, в приведенном ниже фрагменте программы при некорректном выделении памяти генерируется исключение с поясняющей ошибку строкой.

```
char *Buffer = new char [1000];
if (Buffer == 0)
    throw "out of memory"; // нехватка памяти
```

Для случая, когда исключение сгенерировано, но в программе *не предусмотрена* его обработка, механизм исключений вызывает из динамической библиотеки функцию завершения `terminate`, которая выдает сообщение "abnormal program termination" и прекращает выполнение программы. Для обработки такого исключения необходимо предусмотреть операторы `try` и `catch`:

```
try
{
    // операторы ...

    char *Buffer = new char [1000];
```

```

    if (Buffer == 0)
        throw "out of memory"; // генерация исключения

    // операторы...
}
catch (char *ErrorMsg)
{
    // обработка исключения
    cout << ErrorMsg << '\n';

    // обработка ошибки и возобновление выполнения программы или
    // вызов функции exit для останова программы

}
// здесь выполнение программы продолжается...

```

В случае, когда исключение генерируется в любом месте программы, следующем за оператором `try` (или внутри любой функции в этом блоке), то управление передается за пределы блока. Если за `try` следует подходящий блок `catch`, то управление переходит к нему.

Блок `catch` начинается с объявления в круглых скобках. Если тип параметра в этом объявлении совпадает с типом значения в операторе `throw`, генерирующем исключение, то управление передается данному блоку `catch`. При несовпадении типов параметров программа ищет другой обработчик, как показано ниже. После выполнения кода в блоке `catch` управление передается первому оператору, следующему за этим блоком, и программа возобновляет работу в нормальном режиме (если блок `catch` не содержит оператор `return` или вызов функции `exit`).

Таким образом операторами `try` и `catch` предотвращается завершение программы стандартным обработчиком исключений. Блок `try` называют *охраняемым разделом* кода. Если исключение не сгенерировано, то поток управления “перепрыгнет” `catch` и перейдет на первый, следующий за ним, оператор. В приведенном выше примере оператор `throw` содержит строку “out of memory”. Так как тип параметра в объявлении `catch (char *)` такой же, то данный блок `catch` получает управление при вызове этого исключения. Обратите внимание: оператор `catch` объявляет параметр типа `char *` с именем `ErrorMsg`, что позволяет внутри блока `catch` получить доступ к значению, заданному в `throw`. Данный механизм очень похож на механизм передачи параметров функции. Чтобы понять это, представьте оператор `throw` как: вызов функции, в котором значение (“out of memory”) передается в функцию.

```

throw "out of memory";

```

Представьте блок `catch` как вызываемую функцию, а объявление `char * ErrorMsg` как объявление ее формальных параметров. Как и параметры функции, переменная `ErrorMsg` доступна только внутри `catch`. Заметьте: в операторе `catch` *должно* содержаться только описание типа без имени параметра.

```

catch (char *)
{
    // не использует значение, определенное
    // в операторе throw
}

```

Для рассматриваемого случая блок `catch`, как и прежде, получит управление по оператору `throw "out of memory"`, но без доступа к значению типа `char *`. Если оператор `catch` определяет какой-либо другой тип параметра (например, `int`), блок *не* получит управление при возникновении данного исключения.

```

catch (int ErrorCode)
{
    // управление НЕ будет получено после выполнения оператора
    // throw "out of memory";
};

```

Поместив несколько catch за блоком try, можно управлять различными типами исключений. Например, в следующем фрагменте программы обрабатываются исключения с аргументом типа int или char \*.

```

try
{
    // ...

    throw "string"; // это исключение обрабатывается первым
                    // блоком catch

    // ...

    throw 5; // это исключение обрабатывается вторым блоком catch
}
catch (char *ErrorMsg)
{
    // обработка любого исключения типа char *...
}
catch (int ErrorCode)
{
    // обработка любого исключения типа int...
}

```

Блок catch, объявление которого содержит многоточие, получает управление в ответ на исключение *любого типа*, сгенерированное предыдущими try-блоками.

```

catch (...)
{
    // получает управление в ответ на исключения
    // любого типа
}

```

Поскольку данный блок не содержит параметров, он не имеет доступа к значению, передаваемому при вызове исключения. Блок catch с многоточием записывается *последним*. Программа ищет блоки catch в порядке их следования и активизирует первый подходящий по типу исключения. Так как catch с многоточием является последним, то в первую очередь управление получает catch с точно совпадающим типом исключения, а не универсальный блок с многоточием. Более подробная информация о механизме исключений и установлении соответствия между операторами throw и блоками catch помещена в справочной системе.

В листинге 8.1 приведен код программы ExcTypes.cpp, в которой генерируются исключения различных типов. Программа разработана как консольное (см. гл. 2) приложение.

---

#### Листинг 8.1

```

// ExcTypes.cpp: программа, демонстрирующая вызов
// и обработку исключений различных типов

#include <iostream.h>

class CExcept

```

```

{
    public:
        CExcept (int ExCode)
        {
            m_ExCode = ExCode;
        }
        int GetExCode ()
        {
            return m_ExCode;
        }
    private:
        int m_ExCode;
};

void main ()
{
    char Ch;

    try
    {
        cout << "at beginning of try block" << '\n';

        cout << "throw 'char *' exception? (y/n): "; cin >> Ch;
        if (Ch == 'Y' || Ch == 'y')
            throw "error description";

        cout << "throw 'int' exception? (y/n): "; cin >> Ch;
        if (Ch == 'Y' || Ch == 'y')
            throw 1;

        cout << "throw 'class CExcept' exception? (y/n): ";
        cin >> Ch;
        if (Ch == 'Y' || Ch == 'y')
            throw CExcept (5);

        cout << "throw 'double' exception? (y/n): "; cin >> Ch;
        if (Ch == 'Y' || Ch == 'y')
            throw 3.1416;

        cout << "at end of try block (no exceptions thrown)"
            << '\n';
    }
    catch (char *ErrMsg)
    {
        cout << "'char *' exception thrown; exception message:"
            << ErrMsg << '\n';
    }
    catch (int ErrorCode)
    {
        cout << "'int' exception thrown; exception code: "
            << ErrorCode << '\n';
    }
    catch (CExcept Except)
    {
        cout << "'class CExcept' exception thrown; code:"
            << Except.GetExCode () << '\n';
    }
}

```

```

catch (...)
{
    cout << "unknown type of exception thrown" << '\n';
}
cout << "after last catch block" << '\n';
}

```

Здесь каждому исключению, сгенерированному в блоке `try` (кроме исключения с типом `double`, после которого управление передается универсальному обработчику), предоставляется специальный обработчик `catch`. Если выбрать для инициации исключение с типом `class CExcept`, то программа создаст временный объект `CExcept`, передавая целочисленное значение в конструктор класса `CExcept`. Далее инициализируется результирующий объект

```
throw CExcept (5);
```

В результате выполняется копирование объекта в параметр `Except` соответствующего блока `catch`. В объект класса можно включить функции для передачи блоку `catch` информации об ошибке и для обработки ошибочной ситуации. Заметьте: как только при выполнении программы возникает исключительная ситуация, последующие операторы внутри блока `try` не выполняются. После выполнения блока `catch`, управление передается оператору, следующему непосредственно за последним блоком `catch`, что приводит к нормальному завершению программы.

## Catch-блоки

Блок `catch` разрабатывается для эффективной обработки каждого исключения. На обработку исключения влияет как общий тип исключения, так и информация, передаваемая через параметр блока `catch`. Механизм обработки исключения таков, что выполнение программы продолжается *не* с оператора, следующего за оператором, вызвавшим исключение. Выполнение возобновляется только с точки, находящейся за последним обработчиком `catch`. Поэтому принятая в C++ модель исключений называется *невозобновляемой (nonresumable)*. Существует *три основных способа обработки исключений* в блоке `catch`. Они перечислены ниже:

- *Продолжение программы.* Если возникшая проблема не фатальна, то `catch` уведомляет о ней и самостоятельно исправляет ошибочную ситуацию. После передачи управления выполнение программы продолжается с оператора, следующего за последним блоком `catch`. Блок `catch` удаляет программные ресурсы (например, блоки памяти или дескрипторы файлов), которые должен был удалить блок `try`, если бы не было сгенерировано исключение. Как показано далее, автоматически удаляются любые локальные параметры, переменные или объекты, созданные в блоке `try` и находящиеся в области видимости в момент возникновения исключительной ситуации.
- *Завершение программы.* Если возникшая проблема достаточно серьезна, обработчик `catch` выполняет требуемую очистку (например, закрывает дескрипторы файлов), сообщает об ошибке, а затем вызывает библиотечную функцию `exit` для завершения программы.
- *Отказ от обработки исключения.* Исключение не обрабатывается данным блоком `catch`. Для этого используется оператор `throw` без конкретизирующего значения. Этот оператор генерирует исключение с *таким же* типом и значением, как и первоначальное исключение, переданное `catch`. Программа ищет другой обработчик исключения по всему видимому коду (и вызывает оператор `terminate`, если он не найден), как описано в параграфе “Вложенные исключения”. Помните: оператор `throw` используется без указания типа *только* в блоке `catch` или внутри вызываемой им функции.

## Универсальный или специальный обработчик?

Ниже приведен пример, где исключение генерируется оператором `throw`, находящемся в блоке `try`. Оператор `throw` также размещается в теле функции, вызываемой в блоке `try`, но управление по-прежнему “перепрыгивает” в присоединенный блок `catch` (если только `throw` не находится внутри *вложенного* блока `try`), как будет описано в следующем разделе. Исключение может также генерироваться библиотечной функцией C++, вызываемой в программе. Например, многие из MFC-функций при обнаружении ошибки генерируют специфические типы исключений (гл. 9). Можно также сгенерировать исключение с помощью оператора `new` (с объектом класса `malloc`) вместо передачи значения 0 при возникновении ошибки выделения памяти. Инструкции по этому методу находятся в справочной системе. Если в программе используется библиотека MFC, то при ошибке в операторе `new` автоматически генерируется исключение, а не возвращается 0. Более подробная информация приведена во врезке, посвященной исключениям MFC, в конце гл. 9.

Следует размещать блоки `try` и `catch` на наиболее удобном уровне программы. Включение больших объемов кода в один блок `try` позволяет уменьшить число обработчиков исключений. Например, можно поместить всю программу внутри одного блока `try` и написать единственный универсальный обработчик исключений для всех типов.

```
void main ( )
{
    try
    {
        // весь программный код находится внутри или
        // вызывается из этого блока ...
    }
    catch (...)
    {
        // единственная универсальная подпрограмма
        // обработки исключений ...
    }
}
```

Но размещение кода большого объема в блоке `try` приводит к сложности или невозможности продолжать выполнение программы после появления исключения. Если увеличивается объем кода в `try`, то увеличивается и объем кода, “перепрыгиваемого” при инициации исключения. Размещение кода малого объема в `try` требует создания большего количества обработчиков исключений, но обеспечивает специализированную обработку каждого его вида. Это упрощает возврат из исключения и продолжение выполнения программы.

Если в блоке `try` возникает исключение, то поток управления пропускает несколько блоков кода. Он покидает `try` и пропускает блоки, вложенные в него, а также функции, вызываемые из блока `try`, и вложенные блоки. Следовательно, механизм обработки исключений должен обеспечить удаление всех автоматических переменных, автоматических объектов или параметров функции, объявленных в любом из этих блоков. Вспомните: автоматические переменные и объекты, как и параметры, уничтожаются при выходе из области видимости, т.е. когда управление выходит из блока, в котором они объявлены. Для удаления этих объектов механизм обработки исключений корректирует содержимое стека и вызывает деструкторы, определенные для автоматических объектов. Деструкторы вызываются в порядке, обратном порядку создания объектов. Этот процесс известен как *возврат стека* (unwinding the stack) и происходит сразу после инициализации формального параметра в блоке `catch`, но до выполнения кода внутри него. Если исключительная ситуация возникла при инициализации параметра блока `catch` или при возврате стека, механизм обработки исключений вызывает функцию `terminate`, не пытаясь найти другой обработчик исключений и программа завершается.



Выполняя возврат стека, программа *не* удаляет статические переменные или объекты, так как они удаляются при выходе из программы. Переменные или объекты, динамически созданные оператором `new`, необходимо удалить явно, используя оператор `delete`. Если появление исключения препятствует этому, данные объекты необходимо корректно удалить в блоке `catch`. В программе `Unwind.cpp` (листинг 8.2) исключение генерируется функцией, вызываемой из блока `try`. В этом примере показано удаление объектов, происходящее во время возврата стека.

---

#### Листинг 8.2

```
// Unwind.cpp : программа демонстрирующая удаление
// объектов во время возврата стека при возникновении исключения
#include <iostream.h>
class CA
{
public:
    CA ()
    {
        cout << "вызов конструктора класса CA" << '\n';
    }
    ~CA ()
    {
        cout << "вызов деструктора класса CA" << '\n';
    }
};
class CB
{
public:
    CB ()
    {
        cout << "вызов конструктора класса CB" << '\n';
    }
    ~CB ( )
    {
        cout << "вызов деструктора класса CB" << '\n';
    }
};
class CC
{
public:
    CC ()
    {
        cout << "вызов конструктора класса CC" << '\n';
    }
    ~CC ()
    {
        cout << "вызов деструктора класса CC" << '\n';
    }
};
CC *PCC = 0; // определение глобального указателя на класс CC
void Func ()
{
    CB B;          // определение экземпляра класса CB
    PCC = new CC; // динамическое создание экземпляра класса CC
    throw "сообщение об исключении";
}
```

```

        cout << "вы никогда не увидите это сообщение!"
        delete PCC;
    }

void main ()
{
    cout << "начало функции main () " << '\n';
    try
    {
        CA A; // определение экземпляра класса CA

        Func ( ) ;
        cout << "конец блока try" << '\n';
    }
    catch (char * ErrorMessage)
    {
        cout << ErrorMessage << '\n';

        delete PCC;
    }
    cout << "конец функции main () " << '\n'
}

```

Результат работы этой программы обычно выглядит так:

```

начало функции main ()
вызов конструктора класса CA
вызов конструктора класса CB
вызов конструктора класса CC
вызов деструктора класса CB
вызов деструктора класса CA
сообщение об исключении
вызов деструктора класса CC
конец функции main ()

```

После начала блока `try`, но до точки генерирования исключения в этом блоке создаются три объекта (по одному экземпляру для каждого из трех классов: CA, CB и CC). Экземпляры классов CA и CB объявлены как автоматически создаваемые объекты. При возникновении исключения управление выходит из блоков, в которых определялись эти объекты, поэтому они удаляются (вызываются их деструкторы) до получения управления блоком `catch`. Удаляются объекты в порядке, обратном порядку их создания. Так как объект класса CC создается динамически с помощью оператора `new`, то он *не* удаляется автоматически при возникновении исключения. Заметьте: код, удаляющий объект класса CC (последний оператор в функции `Func`), пропускается при возникновении исключительной ситуации, поэтому этот объект удаляется `catch` явно.

## Вложенные исключения

Программное управление в момент возникновения исключения может находиться на втором или более глубоком уровне вложения блока `try`. Например, в приведенном ниже коде программа `main` вызывает функцию `Func`, генерирующую исключение. При появлении исключения поток управления входит в два блока `try`, не выполняя ни один из них. Таким образом, исключение генерируется в контексте двух динамически вложенных блоков. Динамическое вложение `try` зависит от действительной последовательности вызовов функций перед возникновением исключения. Например, если бы

функция Func была вызвана из какой-либо другой точки программы, то исключение возникло бы только внутри одного try-блока.

```
void Func()
{
    try
    {
        // ...
        throw "help!" ;
        // ...
    }
    catch (char *Msg)
    {
        // ...
    }
    catch (...)
    {
        // ...
    }
}

void main ( )
{
    // другие операторы ...
    try
    {
        // ...
        FuncA();
        // ...
    }
    catch (char *Msg)
    {
        // ...
    }
    catch (...)
    {
        // ...
    }
    // другие операторы ...
}
```

Программа ищет обработчик catch при возникновении исключительной ситуации следующим образом: выполняется поиск соответствующего блока catch, связанного с самым внутренним блоком try. В приведенном примере это try внутри функции Func. Программа ищет блоки catch в порядке их определения и активизирует первый блок, совпадающий по типу исключения или определенный с многоточием. Если подходящий блок catch не найден или код внутри активизированного catch сам генерирует исключение, то программа ищет блоки catch, связанные со *следующим* вложенным блоком try. В нашем примере это блок из main. Поиск продолжается, пока не будет найден catch, связанный с самым внешним блоком try. Если обработчик *не* найден (либо исключение возникло во время инициализации параметра catch или при возврате стека), то вызывается функция terminate, которая завершает программу, генерируя сообщение об ошибке.

## Win32-исключения и их обработка

Исключения могут генерироваться системным кодом Win32 API в ответ на сбой аппаратного или программного обеспечения (наряду с генерацией операторами `throw`). Такие исключения называют *исключениями Win32*, *структурированными исключениями* или *исключениями языка C*. Вспомните: Win32 является интерфейсом низкого уровня, используемым в 32-разрядных консольных программах и программах с графическим интерфейсом (см. гл. 2). Термины *структурированное исключение* и *исключение языка C* возникли вследствие того, что исключения Win32 обрабатываются посредством множества операторов и структур языка C. Механизм их использования, описанный в этом параграфе, достаточно сложен. Для получения более детальной информации о средствах обработки исключений пользуйтесь справочной системой.

Генерирующие исключения Win32 *аппаратные ошибки*:

- *деление на ноль*;
- *некорректное обращение к памяти*.

В ответ на определенные *ошибки программного обеспечения* (например, нехватка памяти), некоторые функции Win32 API также генерируют исключение, а не просто возвращают код ошибки. Например, если функции `:HeapAlloc` передается флаг `HEAP_GENERATE_EXCEPTIONS`, то при неудачной попытке распределения памяти генерируется исключение. Для каждой функции Win32 API, которая может генерировать исключения, в документации описаны все случаи, в которых происходит вызов исключения. Отметим: функции Win32 API вызываются из консольных приложений (см. гл. 2). Но их можно вызывать и из программ с графическим интерфейсом, даже написанных с использованием MFC (поскольку функции MFC сами вызывают функции Win32 API). Если возникло исключение Win32, но программа не предоставляет для него обработчик, выдается сообщение об ошибке и программа завершается. Пусть, например, деление на ноль имеет место в следующей программе

```
int I = 0;
int J = 5 / I;
```

а обработчик исключения Win32 не предоставлен. В результате на экран выводится окно сообщения и программа завершается. После щелчка в окне сообщения на кнопке `Details>>` на экран выводится более подробная информация об исключении.

Для обработки исключений Win32 можно воспользоваться одним из двух способов.

1. Первый способ обработки исключений Win32 состоит в задании блока *catch* с *многоточием*, который получает управление в ответ на любое сгенерированное исключение или исключение Win32. В последнем случае блок может обработать ошибку и продолжить выполнение программы. (Чтобы отобразить окно сообщения об ошибке, и завершить работу программы, *catch* должен сгенерировать исключение Win32, используя оператор `throw` без значения).

```
try
{
    // ...
    // генерация исключения Win32:
    int I = 0;
    int J = 5 / I;
    // ...
}
catch (...)
{
    // будет получено управление в ответ на исключение Win32;
    // обработка исключения ...
}
```

Рассматриваемый способ обработки исключений порождает проблему, которая состоит в том, что блок `catch` не получает информацию о причинах возникновения конкретного исключения, так как параметр ему не передается.

- Второй способ обработки Win32-исключений основан на определении *функции трансляции исключения* (exception translator function). Имя этой функции

```
void SETranslate
(unsigned int ExCode,
_EXCEPTION_POINTERS *PtrExPtrs)
```

передается в библиотечную функцию `_set_se_translator`.

```
_set_se_translator (SETranslate);
```

Можно изменить имя функции или ее параметров, но не их типы. Для вызова `_set_se_translator` необходимо включить в программу файл заголовков `Eh.h`. Теперь при каждом появлении исключения Win32 будет вызываться функция трансляции исключения. Первый параметр, передаваемый в эту функцию (`ExCode`), содержит *код произошедшего исключения*, второй – указывает на *структуру с подробной информацией об исключении*. В табл. 8.1 приведен список некоторых общих кодов исключений Win32, задаваемых для параметра `ExCode`. Чтобы получить доступ к константам первого столбца этой таблицы, просто включите в программу файл `WINDOWS.H`. Описание информации, доступной с помощью параметра `PtrExPtrs`, приведено в документации по функции Win32 API `::GetExceptionInformation` в справочной системе.

Табл. 8.1. Коды исключений Win32

Код исключения	Описание исключения
EXCEPTION_ACCESS_VIOLATION	Попытка обращения к невыделенному адресу виртуальной памяти
EXCEPTION_ARRAY_BOUNDS_EXCEEDED	Попытка доступа к элементу массива за его пределами (для контроля границ массива требуется аппаратная поддержка)
EXCEPTION_DATATYPE_MISALIGNMENT	Попытка доступа к переменной по адресу с неправильным выравниванием: например, 16-битовая переменная должна выравниваться по 2-байтовой границе, а 32-битовая – по 4-байтовой границе
EXCEPTION_FLT_DENORMAL_OPERAND	Один из операндов с плавающей запятой слишком мал для представления в данном формате
EXCEPTION_FLT_DIVIDE_BY_ZERO	Делитель в операции деления с плавающей запятой равен нулю
EXCEPTION_FLT_INEXACT_RESULT	Результат операции с плавающей запятой нельзя точно представить в виде десятичной дроби
EXCEPTION_FLT_INVALID_OPERATION	Общая ошибка операции с плавающей запятой, т.е. ошибка, отсутствующая в этом списке
EXCEPTION_FLT_OVERFLOW	Результат операции с плавающей запятой слишком велик для представления в этом формате
EXCEPTION_FLT_STACK_CHECK	Операция с плавающей запятой приводит к переполнению стека либо к заему из стека
EXCEPTION_FLT_UNDERFLOW	Результат операции с плавающей запятой слишком мал для представления в этом формате
EXCEPTION_INT_DIVIDE_BY_ZERO	Делитель операции целочисленного деления равен нулю

Табл. 8.1. (Окончание)

Код исключения	Описание исключения
EXCEPTION_INT_OVERFLOW	Результат целочисленной операции слишком велик для сохранения в виде целого значения (т.е. потерян один или больше старших битов)
EXCEPTION_PRIV_INSTRUCTION	Попытка выполнения машинной команды, запрещенной на текущем уровне привилегий

Генерируемые операциями с плавающей запятой исключения (коды которых начинаются с EXCEPTION\_FLT) по умолчанию запрещаются. Если ошибка происходит, операция формирует ноль или максимальное действительное значение. Для осуществления обработки таких исключений следует использовать директиву компилятора `_controlfp`.

Фактически функция трансляции исключения преобразует исключение Win32 в исключение C++. Заметьте: если транслирующая функция *не* генерирует исключение явно, то механизм обработки исключений возобновляет поиск обработчика `catch (...)` так, как если бы транслирующая функция не была предусмотрена (или завершает работу программы выводом окна сообщения, если такой обработчик не найден). Транслятор генерирует исключение C++ с использованием значения, переданного в качестве параметра и, возможно, предоставляет дополнительную информацию, например:

```
void SETranslate
(unsigned int ExCode,
_EXCEPTION_POINTERS *PtrExPtrs)
{
    throw ExCode;
}
```

*В ситуации, когда*

1. код, вызвавший первоначально исключение Win32, находится в блоке `try`, и
2. существует блок `catch` соответствующего типа (в примере – `unsigned int`)

`catch` активизируется и может использовать информацию из своего параметра для обработки исключения. Например, таким образом:

```
catch (unsigned int ExCode)
{
    // Параметр ExCode содержит идентификатор исключения Win32;
    // далее оно обрабатывается соответствующим образом ...
}
```

Работу функции трансляции исключений демонстрирует приведенная ниже программа `ExTrans.cpp` (листинг 8.3).

### Листинг 8.3

```
// ExTrans.cpp: программа, демонстрирующая работу
// функции трансляции исключений Win32 в исключения C++
#include <windows.h>
#include <iostream.h>
#include <eh.h>

class CSExcept
{
public:
    CSExcept (unsigned int ExCode)
```

```

    {
        m_ExCode = ExCode;
    }
    unsigned int GetExCode ( )
    {
        return m_ExCode;
    }

private:
    unsigned int m_ExCode;
};

void SETranslate
(unsigned int ExCode,
_EXCEPTION_POINTERS *PtrExPtrs)
{
    throw CSExcept (ExCode);
}

void main ()
{
    char Ch;

    _set_se_translator (SETranslate);

try
{
    // ...

    cout << "сгенерировать исключение 'целочисленное деление на ноль'?"
        "(y/n): ";
    cin >> Ch;
    if (Ch== 'y' || Ch == 'Y')
    {
        int I=0;
        int J=5/I;
    }

    cout << "сгенерировать исключение 'нарушение доступа'? (y/n): ";
    cin >> Ch;
    if (Ch == 'y' || Ch == 'Y')
    {
        *((char *)0) = 'x';
    }
    // операторы, генерирующие другие исключения...
}
catch (CSExcept SEexcept)
{
    switch (SEexcept.GetExCode ())
    {
        case EXCEPTION_INT_DIVIDE_BY_ZERO:
            cout << "произошло исключение 'целочисленное деление на ноль'"
                << '\n';
            break;

        case EXCEPTION_ACCESS_VIOLATION:
            cout << "произошло исключение 'нарушение доступа'"

```

```

        << '\n';
        break;

    default:
        cout << "произошло неизвестное исключение Win32" << '\n';
        throw;
        break;
    }
}
}

```

Программа ExTrans.cpp устанавливает функцию трансляции исключений SETranslate и генерирует одно из двух исключений Win32: *целочисленное деление на ноль* или *нарушение доступа*. Второе исключение возникает при попытке обращения к памяти по нулевому виртуальному адресу (не заданному для процесса). Механизм обработки исключений вызывает функцию SETranslate, генерирующую исключение C++ с объектом класса CSExcept. Программа определяет класс CSExcept специально для обработки исключений Win32. Конструктор класса хранит код исключения в переменной m\_ExcCode с атрибутом доступа private. Когда генерируется исключение C++, управление передается в блок catch, следующий за блоком try, сгенерировавшим исключение Win32. Блок catch обращается к классу CSExcept для получения кода исключения, используемого для его обработки (в нашем примере – для вывода сообщения). Если catch не распознает код исключения Win32, то он использует оператор throw для инициализации стандартной обработки исключения, т.е. для завершения программы с выводом окна сообщения. Обратите внимание: программа включает файл WINDOWS.H, поэтому может использовать константы EXCEPTION\_INT\_DIVIDE\_BY\_ZERO и EXCEPTION\_ACCESS\_VIOLATION. Если оператор new не может удовлетворить запрос на выделение памяти, то он возвращает значение 0. Чтобы не проверять возвращаемое значение всякий раз, когда выполняется оператор new, создайте *единственную* подпрограмму обработки ошибок в блоке catch, обрабатывающую исключение Win32 EXCEPTION\_ACCESS\_VIOLATION (как в приведенном примере программы). Если оператор new возвращает нулевой указатель, то при любой попытке использовать указатель для обращения к памяти управление получает catch.

## Резюме

В этой главе были рассмотрены особенности обработки исключений, в том числе следующие:

- *Исключения.* Исключение это прерывание обычного потока выполнения программы, происходящее в ответ на некоторые типы сбоев аппаратного или программного обеспечения. *Программное исключение* генерируется оператором throw и следующим за ним значением, содержащим информацию об ошибке. *Тип* исключения определяется типом значения, задаваемого в throw.
- *Обработка исключений.* Для обработки исключений, происходящих внутри определенного фрагмента кода, поместите этот код внутри блока try и определите один или несколько блоков catch, которые могут обрабатывать исключения конкретного или произвольного типа (в последнем случае вместо объявления типа используют многоточие). Если исключение генерируется в блоке try, то управление передается следующему блоку catch, подходящему по типу исключения (если таковой имеется). Блок catch обрабатывает ошибку, выполняет требуемую очистку и выдает необходимые сообщения. Затем выполнение программы продолжается с оператора, следующего за последним catch. Кроме того, catch может завершить выполнение программы либо повторно сгенерировать исключение. Обработчики исключений могут быть динамически вложенными. При обработке исключения осуществляется поиск соответствующего обработчика catch, начиная



с наиболее глубоко вложенного блока `try`. Если программа *не* предоставляет обработчик конкретного исключения, то завершается она в аварийном режиме.

- *Win32-исключения*. Подсистема Win32 генерирует исключение в ответ на аппаратную ошибку (например, попытку обращения к памяти по недействительному адресу) или программную ошибку в функции Win32 API (например, `: :HeapAlloc`). Такие исключения называют исключениями Win32, *структурированными* исключениями или исключениями языка C. В отличие от исключения C++, исключение Win32 *не* имеет типа. Соответственно, если исключение Win32 происходит в `try`, то активизируется только `catch` без конкретного типа исключения (т.е. определенный с помощью многоточия). Вместо того чтобы обрабатывать исключения Win32 в универсальном блоке `catch`, можно определить функцию трансляции исключений и установить эту функцию, вызвав библиотечную функцию `_set_se_translator`. С этого момента при возникновении исключительной ситуации Win32 будет вызываться функция трансляции, которая может использовать оператор `throw` для генерирования исключения определенного типа. Это исключение затем перехватывается соответствующим обработчиком `catch`.

# Часть III

## Программирование графического интерфейса



## Глава 9

# Программа с графическим интерфейсом

---

- Как спроектировать графический интерфейс
- Проектирование программы
- Состав проекта
- Как работает программа

В этой части книги спроектированы программы с *графическим интерфейсом* (GUI – *graphical user interface*), которые могут создавать одно или более окон приложений с элементами интерфейса (меню, панелями инструментов, строками состояния, полосами прокрутки и т. д.). Эти программы могут выводить рисунки, растровые изображения и текст с использованием шрифтов, доступных в операционной системе. Вы научитесь создавать простые программы с графическим интерфейсом, используя возможности мастера генерации приложений Application Wizard и библиотеку MFC.

## Как спроектировать графический интерфейс

---

В рамках Visual C++ предусмотрено несколько способов написания программ с графическим интерфейсом.

1. Можно писать такие программы на языке C или C++ *с непосредственным обращением к основным функциям Win32 API*, которые являются частью операционных систем Windows 98/Me и Windows NT/2000. При таком подходе, прежде чем перейти к решению целевой задачи разрабатываемого приложения требуется написать множество строк программного кода.
2. Такие программы можно строить *с помощью библиотеки MFC*, содержащей большой набор готовых классов и вспомогательный код для выполнения стандартных задач программирования в среде Windows (например, создания окон и обработки сообщений). Кроме того, MFC используется для быстрого добавления в программы панелей инструментов, многопанельных окон, поддержки OLE. Эта библиотека применяется для создания элементов ActiveX, которые многократно используются программными компонентами и отображаются в Web-браузерах и других контейнерных приложениях (гл. 25). Использование MFC позволяет упростить программы с графическим интерфейсом и значительно облегчить процесс программирования.
  - Функции MFC содержат вызовы функций Win32 API. Говорят, что Win32 API “упакован” в библиотеку MFC, предоставляющую более высокоуровневые и мобильные средства программного интерфейса. Кроме того, в MFC-программах можно свободно вызывать функции Win32 API.
3. Подобные программы можно проектировать *на языке C++ с использованием библиотеки MFC и различных мастеров*. Преимущество третьего подхода состоит в использовании не только уже написанного кода MFC, но и сгенерированного исходного кода, позволяющего решить многие рутинные задачи программирования. Библиотека MFC и мастера освобождают вас от необходимости создавать средства визуального интерфейса вручную и обеспечивают соответствие этого интерфейса требованиям Microsoft. В этой книге описан третий, самый высокоуровневый, способ написания графических приложений. Возможности мастеров не ограничиваются генерацией простых оболочек программ. Они позволяют создавать программы с большим набором сложных компонентов. К таковым относятся панели инструментов, строки состояния, контекстная справка.

объекты OLE, средства доступа к базам данных и даже законченные меню с частично или полностью функционирующими командами открытия и сохранения файлов, печати, предварительного просмотра печати и выполнения других задач. Вам остается только добавить собственный код, определяющий логику работы программы, после генерации основного кода программы с помощью мастеров.

- *Master Application Wizard* используется для генерации основы исходных файлов программ.

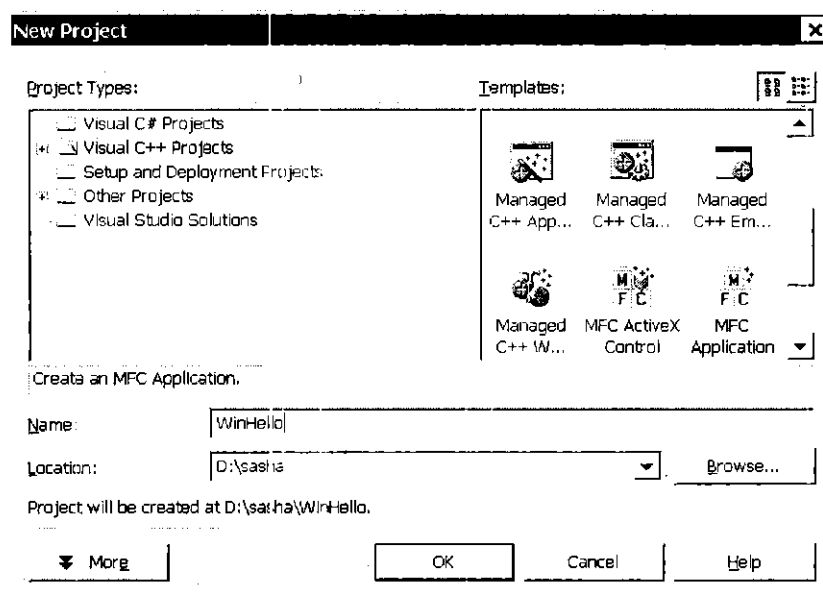
## Проектирование программы

В этом разделе рассматривается создание простейшего приложения на VC7 при помощи мастера создания новых приложений – Application Wizard. Сначала будет сгенерирован исходный код программы WinHello. Результирующий исполняемый модуль будет создан после внесения ряда изменений.

### Как сгенерировать исходный код

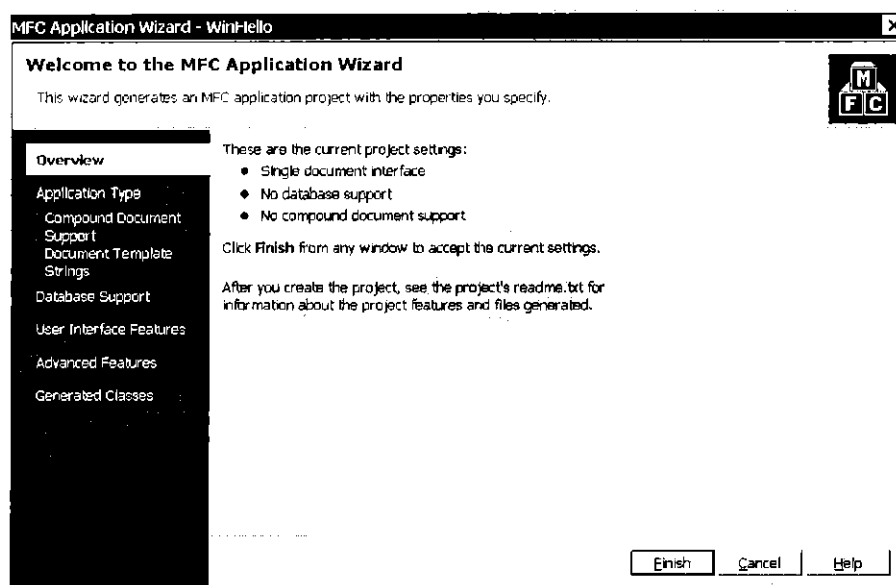
Чтобы сгенерировать программу, с помощью мастера Application Wizard следует создать новый проект необходимого типа. Затем в совокупности вкладок окна мастера указываются необходимые свойства приложения. Для создания нового проекта необходимо выполнить следующие действия:

1. На панели инструментов Standard нажать кнопку New Project (если панель не видна, ее отображение можно включить командой View/Toolbars/Standard).
2. В меню File выбрать команду New и в открывшемся подменю из двух пунктов выбрать Project... (данной команде соответствует сочетание клавиши Ctrl+Shift+N).



3. Вышеописанные действия приведут к открытию диалогового окна New Project. В поле Project Types показана иерархия типов проектов Visual Studio. При выборе папки Visual C++ Projects в поле Templates отобразятся различные шаблоны проектов, доступные в VC7. В данном случае необходимо выбрать шаблон типа MFC Application. В поле Name необходимо ввести WinHello, в поле Location – задать рабочую папку создаваемого проекта. При необходимости рабочую папку можно выбрать кнопкой Browse...

4. После задания необходимых параметров следует нажатием на кнопку ОК запустить мастер MFC Application Wizard. В левой части окна MFC Application Wizard в виде гиперссылок оформлены ярлыки восьми вкладок, на которых задаются различные свойства создаваемой программы.



- Изначально показывается вкладка Overview, на которой отражены общие характеристики создаваемой программы, заданные на других вкладках.
- На вкладке Application Type задаются общие свойства приложения (язык используемых ресурсов, поддержка одно- или многооконного интерфейса, использование библиотек MFC и некоторые другие). Выберите тип приложения Single Document (однооконное приложение), и укажите использование библиотеки MFC как статического ресурса (а не как распространяемой вместе с приложением DLL-библиотеки), установив опцию Use MFC as a static library. Если включена опция Document/View Architecture Support, то мастер сгенерирует отдельные классы для хранения и отображения данных программы и код для чтения и записи данных на диске. Обычно данная опция остается включенной, хотя ее отключение приводит к некоторому уменьшению объема кода программы.
- Следующая вкладка – Compound Document Support – управляет включением в создаваемое приложение поддержки технологии ActiveX. Установленное по умолчанию отсутствие поддержки подходит для создаваемой программы, поэтому на этой вкладке нет необходимости что-либо изменять.
- Вкладка Document Template Strings задает шаблонные имена для документов создаваемого приложения. В данном случае здесь нет необходимости вносить изменения.
- Вкладка Database support содержит опции, управляющие поддержкой приложением баз данных. Выбранное по умолчанию значение None – отсутствие поддержки – подходит в данном случае, так что изменения не требуются.
- Вкладка User Interface Features задает внешний вид и свойства пользовательского интерфейса приложения. Флажки в левом столбце вкладки:
  - Thick frame – толстая рамка с возможностью изменения размера окна приложения (опция установлена по умолчанию).

- Minimize box – наличие в строке заголовка кнопки для сворачивания (минимизации) окна приложения (опция установлена по умолчанию).
  - Maximize box – наличие в строке заголовка кнопки для максимизации/разворачивания до полноэкранного размера окна приложения (опция установлена по умолчанию).
  - Minimized – запуск приложения в фоновом режиме (вместо окна приложения изначально будет отображена его кнопка на панели задач).
  - Maximized – запуск приложения с разворачиванием его окна до полноэкранного режима.
  - System menu – наличие в строке заголовка окна приложения кнопки для вызова системного меню (опция установлена по умолчанию).
  - About box – включение в меню Help создаваемого приложения пункта About..., вызывающего окно с информацией о приложении.
  - Initial status bar – наличие в окне приложения строки состояния. Она содержит индикаторы включения Num Lock, Caps Lock и Scroll Lock и показывает подсказки к элементам меню и кнопкам панели инструментов. Включение данной опции добавляет к командам меню View создаваемого приложения команду показа/сокрытия строки состояния (опция установлена по умолчанию).
  - Split Window – дает приложению возможность деления его окна на части с независимыми элементами прокрутки изображения.
  - Флажки Child Frame Styles управляют подчиненными окнами создаваемой программы.
  - Переключатель Select toolbars указывает наличие/отсутствие в создаваемом приложении панели инструментов. Установка None указывает на отсутствие панели инструментов, установка Standard docking – на наличие стандартной паркующей панели инструментов.
  - Флажок Browser Style придает элементам панели инструментов стиль Internet Explorer.
  - Вкладка Advanced Features задает дополнительные свойства программы – наличие/отсутствие встроенной системы контекстной помощи, поддержки печати и т.д. Также на этой вкладке указывается длина запоминаемого списка последних открытых файлов, показываемого в конце меню File создаваемого приложения. В данном случае можно отключить установленную по умолчанию поддержку печати документов и поддержку компонентов ActiveX.
  - Последняя вкладка Generated Classes носит в основном справочный характер и содержит описания классов, которые мастер сгенерирует для создаваемого приложения.
5. После установки всех необходимых параметров и нажатия кнопки Finish мастер создает необходимые файлы.

## **Внесение изменений в сгенерированный код**

Сгенерированные мастером Application Wizard исходные файлы достаточны для построения функционирующей программы. Иначе говоря, сразу после генерации исходных файлов мастером Application Wizard можно построить (скомпилировать и скомпоновать) загрузочный модуль и запустить программу (хотя она и не выполняет ничего существенного). В рассматриваемом примере, если созданный код не изменять, программа отобразит пустое окно. В этом параграфе вы узнаете, как добавить код, отображающий строку “Привет!” в центре окна программы.

1. Откройте исходный файл WinHelloDoc.h. Самый простой способ открыть файл из текущего проекта – дважды щелкнуть на его имени во вкладке Solution Explorer. Файл WinHelloDoc.h содержит описание *класса документа* программы, называемого CWinHelloDoc и порожденного MFC-классом CDocument. Как показано далее в этой главе, класс документа отвечает за чтение, запись и сохранение данных программы. В нашем тривиальном случае класс документа

только хранит представляющую собой данные программы строку фиксированного сообщения ("Привет!").

2. Добавьте в определение класса CWinHelloDoc защищенную переменную-член с именем `m_Message`, хранящую указатель на строку сообщения, а также открытую функцию-член `GetMessage`, возвращающую указатель на строку. Чтобы сделать все это, введите строки, выделенные полужирным шрифтом. Обратите внимание, что в примерах программ этой части книги именам переменных-членов предшествует префикс `m_`, позволяющий отличать их от параметров и других переменных, которые не являются членами класса, что соответствует соглашению о наименованиях, принятому в MFC. В приведенном ниже фрагменте показано начало определения класса CWinHelloDoc и содержится код, сгенерированный мастером Application Wizard, а также строки кода, набранные вручную и выделенные полужирным шрифтом. Все строки кода, набранные или изменяемые вами, будут отображаться полужирным шрифтом. Хотя добавляются и изменяются только эти строки, в книге обычно приводится более крупный фрагмент программы, чтобы облегчить поиск внутри сгенерированного исходного файла соответствующей позиции.

```
class CWinHelloDoc : public CDocument
{
protected:
    char *m_Message;

public:
    char *GetMessage ()
    {
        return m_Message;
    }
protected: // используются только для сериализации
    CWinHelloDoc();
    DECLARE_DYNCREATE(CWinHelloDoc) // оставшаяся часть
                                     //определения класса CWinHelloDoc ...
```

### Примечание

Представление программных строк в книге может немного отличаться от того, которое использовано на экране редактора Visual Studio.NET. Это связано с тем, что некоторые из строк программы, сгенерированные мастером Application Wizard, пришлось разбить на части, чтобы разместить на страницах данной книги. Кроме того, комментарии, сгенерированные мастером по мере возможности переведены.

Далее в этой главе приведены листинги (9.1 — 9.8), которые содержат полный текст всех исходных файлов программы WinHello, включая добавления и изменения.

3. Откройте файл WinHelloDoc.cpp, реализующий класс документа программы, CWinHelloDoc. В конструкторе этого класса добавьте выражение, выделенное полужирным шрифтом в приведенном ниже фрагменте кода.

```
// Конструктор/деструктор CWinHelloDoc

CWinHelloDoc::CWinHelloDoc()
{
    // TODO: добавьте сюда одноразовый код конструктора
    m_Message="Привет!";
}
```

Эта строка автоматически присвоит значение “Привет!” при создании экземпляра класса переменной `m_Message`.

### Примечание

Комментарии, начинающиеся со слова `TODO` (в русском языке может интерпретироваться как указание “выполнить”) вставляются мастером `Application Wizard` в те позиции сгенерированного кода, куда программист обычно добавляет собственный код. Следует заметить, что мастер помечает далеко не все подобные позиции, а только наиболее типичные и безопасные с точки зрения внесения изменений.

4. Откройте реализующий класс *представления* программы файл `WinHelloView.cpp`. Этот класс имеет имя `CWinHelloView` и порождается из MFC-класса `CView`. Класс представления отвечает за обработку информации, вводимой пользователем, и за управление используемыми для отображения данных программы окнами представления.
5. В функцию-член `OnDraw` класса `CWinHelloView` в файле `WinHelloView.cpp` добавьте выражения, выделенные полужирным шрифтом.

```
//Отображение окна класса CWinHelloView

void CWinHelloView::OnDraw(CDC* pDC)
{
    CWinHelloDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    // TODO: сюда добавьте код отображения собственных данных

    RECT ClientRect;
    GetClientRect (&ClientRect);
    pDC->DrawText
        (pDoc->GetMessage (), //получить строку
        -1,
        &ClientRect,
        DT_CENTER | DT_VCENTER | DT_SINGLELINE);
}
```

Функция-член `OnDraw` класса представления `CWinHelloView` вызывается MFC-классами каждый раз, когда требуется нарисовать или перерисовать окно программы. Это делается, например, при первоначальном создании окна, изменении размера или отображении окна, ранее скрытого под другим окном. Код, добавленный в функцию `OnDraw`, отображает строку “Привет!”, хранящуюся в классе документа. Функция `OnDraw` получает указатель на класс документа программы, вызывая функцию-член `GetDocument` класса `CView`. Затем она использует этот указатель для вызова добавленной в пункте 2 функции-члена `GetMessage` класса `CWinHelloDoc` с целью получения указателя на строку сообщения. Это сложный метод получения простой строки, но здесь он используется для того, чтобы продемонстрировать типичный способ, с помощью которого класс представления получает программные данные от класса документа для их последующего отображения. Функция `OnDraw` принимает указатель на объект контекста устройства, являющийся экземпляром класса `CDC` библиотеки MFC (объект контекста устройства описан в гл. 18). Объект контекста устройства связан с определенным устройством (в `WinHello` – с окном представления) и обеспечивает набор функций-членов для отображения выводимой информации на этом устройстве.

Чтобы поместить строку в центре окна представления, функция `OnDraw` вызывает функцию-член `GetClientRect` класса `CWnd` для получения текущих размеров окна представления. Затем эти размеры структурой `RECT` передаются в функцию `DrawText` вместе с признаками, задающими центрирование строки по горизонтали и вертикали в соответствии с указанными размерами (`DT_CENTER` и `DT_VCENTER`).



При разработке приложения с полным набором функций, на базе сгенерированного мастером Application Wizard кода, придется внести в него намного больше изменений. Для этого используется множество инструментов, описываемых в последующих главах, например, редактор ресурсов.

#### Примечание

При вставке в исходный код проекта фрагментов русского текста (в том числе и комментариев) в некоторых случаях VC производит автоматическое изменение установок сохранения для правильной записи на диск русского текста (устанавливает для текста кодировку Windows 1251). Если такое изменение автоматически не произведено, то сохранение и загрузка русских символов будут производиться некорректно. Убедиться в правильной выборке кодировки можно в окне, вызываемом командой File/Advanced Save Options...

## Как скомпоновать и запустить программу

Чтобы построить программу:

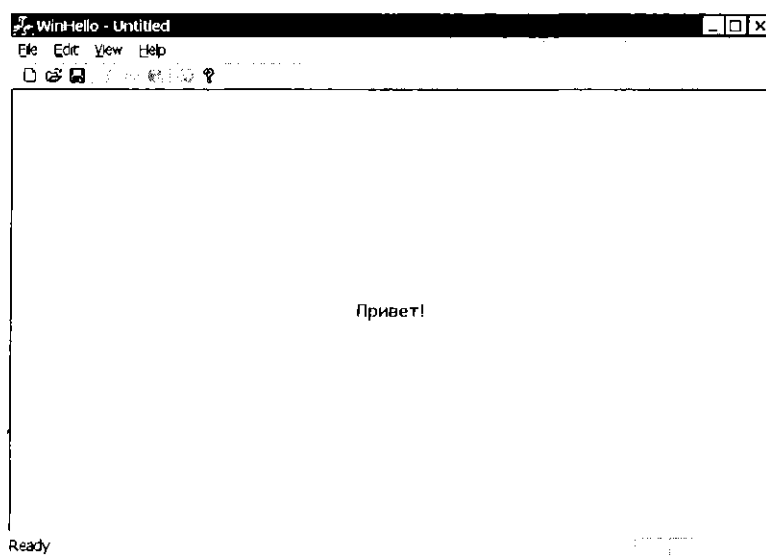
- выберите в меню Build команду Build;
- или нажмите сочетание клавиш Ctrl+Shift+B;
- или щелкните на кнопке Build в панели инструментов Build.

#### Примечание

Visual Studio строит проект, используя текущую активную конфигурацию. При изначальном создании проекта мастером Application Wizard активной конфигурацией будет Win32 Debug. В дальнейшем ее можно изменить, выбрав в меню Build команду Configuration Manager... Если программа строится с использованием конфигурации Win32 Debug, то все выходные файлы размещаются в подпапке Debug папки проекта, а если вы строите ее с Win32 Release, то все выходные файлы размещаются в Release-подпапке.

Если ошибки в ходе построения программы не обнаружены, то ее можно выполнить:

- выбрав в меню Debug команду Run;
- или нажав F5.



Мастер Application Wizard создал код для отображения всего меню. Команда Exit в меню File и команда About в меню Help полностью работоспособны. Это значит, что мастер Application Wizard сгенерировал весь код, необходимый для реализации этих команд. Команды в меню Edit не функционируют, т. е. Application Wizard не генерирует кода реализации этих команд, поэтому они недоступны. Команды меню View для отображения/сокрытия строки состояния и панели инструментов работают. Как можно создать код для обработки некоторых из этих команд вы узнаете в гл. 10 и 11. Команды в меню File функционируют частично (кроме Exit). Это значит, что мастер Application Wizard создал часть кода, необходимого для реализации этих команд. Если выбрать команду Open..., программа отобразит стандартное диалоговое окно Open. Если в этом диалоговом окне выбрать файл и щелкнуть на ОК, имя файла отобразится в строке заголовка (заместив имя "Untitled", отображаемое при запуске программы), но в действительности содержимое файла не читается и не отображается. Если теперь выбрать команду New, программа снова отобразит имя "Untitled" в строке заголовка, но в действительности инициализация нового документа не происходит. Как включить в программу код инициализации нового документа описано в гл. 11.

Если выбрать команду Save As... (или Save с документом "Untitled"), код мастера Application Wizard отобразит диалоговое окно Save As. Если задать имя и щелкнуть на кнопке ОК, программа создаст пустой файл с указанным именем, но не запишет в этот файл данных. Как задать в программе код для выполнения реальных операций чтения и записи командами Open..., Save и Save As... вы узнаете в гл. 12. Если "открыть" существующий файл с помощью команды Open..., а затем выбрать команду Save, то первоначальное содержимое файла будет удалено без предупреждения. Будьте осторожны с командами меню File программы WinHello!

Если несколько файлов открывались с помощью команды Open..., то в меню File отобразится их список, содержащий (по умолчанию) имена четырех файлов, открывавшихся последними. При выходе из программы код мастера Application Wizard сохраняет этот список в файле инициализации программы WinHello.ini, размещаемом в каталоге Windows, поэтому список восстанавливается при каждом следующем ее выполнении. В следующей главе вы научитесь использовать редактор ресурсов Visual Studio для удаления ненужных пунктов меню и конструирования своего значка программы. Обратите внимание, что программа WinHello отображает стандартный значок библиотеки MFC.

## Состав проекта

---

Спроектированную ранее в этой главе программу WinHello относят к категории приложений с *однооконным интерфейсом* (*single document interface – SDI*). Это означает, что в каждый конкретный момент времени в ней может отображаться только один документ в одном окне. В главе 17 будут изучены главные классы и исходные файлы, генерируемые мастером Application Wizard для приложений с *многооконным интерфейсом* (*MDI – Multiple Documented Interface*).

Когда мастер Application Wizard создает приложение SDI, он порождает 4 главных класса:

- класс документа;
- класс представления (view);
- класс главного окна;
- класс приложения.

Решаемые программой задачи распределяются по этим четырем главным классам, и мастер Application Wizard создает восемь отдельных исходных файлов (файл заголовка и файл реализации для каждого из классов). По умолчанию он порождает имена классов и исходных файлов по имени проекта, хотя, используя мастер Application Wizard, при генерации программы можно указать альтернативные имена. Экземпляры главных классов обращаются друг к другу и обмениваются данными, вызывая открытые функции-члены другого класса и посылая *сообщения*. Сообщения будут описаны в гл. 10.

- *Класс документа* в WinHello называется CWinHelloDoc. Он порождается из класса CDocument библиотеки MFC. Заголовочный файл CWinHelloDoc имеет имя WinHelloDoc.h, а файл реализации – WinHelloDoc.cpp. Эти файлы устроены так, как описано в параграфе “Размещение определения класса в программе” гл. 4. Класс документа отвечает за хранение данных программы и за чтение и запись данных на диск. Класс документа WinHello не выполняет операций ввода/вывода информации на диск и хранит только одну строку сообщения.
- *Класс представления* в WinHello называется CWinHelloView и порождается от MFC-класса CView. Заголовочный файл CWinHelloView называется CWinHelloView.h, а имя файла реализации – CWinHelloView.cpp. Класс представления отвечает за отображение данных программы (на экране, принтере или другом устройстве) и за обработку информации, вводимой пользователем. Этот класс управляет *окном представления (view window)*, которое используется для отображения данных программы на экране. Класс представления в WinHello просто отображает внутри окна представления строку сообщения.
- *Класс главного окна* в WinHello называется CMainFrame и порождается от MFC-класса CFrameWnd. Заголовочный файл CMainFrame называется MainFrm.h, а имя файла реализации – MainFrm.cpp. Класс главного окна управляет главным окном программы, которое является *объемлющим* окном и включает рамку окна, строку заголовка, строку меню и системное меню. Объемлющее окно также содержит кнопки Minimize, Maximize, Close, а иногда и другие элементы пользовательского интерфейса, например, панель инструментов, строку состояния. Окно представления, управляемое классом представления, располагается в пустой области объемлющего окна, называемой *клиентской областью* главного окна. Окно представления не имеет видимых элементов, кроме текста и графики, отображающихся явно (например, строки “Привет”, отображаемой программой WinHello). Окно представления – *дочернее* окно главного окна, поэтому оно всегда отображается внутри границ рабочей области главного окна.
- *Класс приложения* назван CWinHelloApp и порожден из MFC-класса CWinApp. Файл заголовков класса CWinHelloApp назван WinHello.h, а файл реализации – WinHello.cpp. Класс приложения управляет программой в целом. Это значит, что он решает общие задачи, не относящиеся к каким-либо другим трем классам (например, выполняет инициализацию программы и ее завершение). Каждая MFC-программа должна создать в точности один экземпляр класса, порожденного из класса CWinApp.

В табл. 9.1 сведены функции четырех главных классов программы WinHello.

В дополнение к перечисленным файлам главных классов мастер Application Wizard и Visual Studio создают еще несколько исходных и установочных файлов. Основные из них кратко описаны в табл. 9.2. Кроме этих файлов для хранения информации различного типа компонент Visual Studio создает следующие файлы: WinHello.aps, WinHello.ncb, WinHello.opt и WinHello.plg. Кроме того, мастер Application Wizard создает файл с именем ReadMe.txt, описывающий большую часть исходных файлов вашей программы. Набор файлов, создаваемый мастером Application Wizard, зависит от компонентов программы, выбранных при ее создании. Специальные файлы, относящиеся к различным функциям программы, будут рассмотрены далее при описании этих функций в последующих главах. Файлы, генерируемые мастером Application Wizard, и перечисленные в табл. 9.1 и 9.2, размещены в папке проекта WinHello, заданном при создании исходных файлов программы, и в подпапке \res папки проекта. Перечисленные файлы *не* включают файлы, создаваемые при построении программы (например, .obj, .res и .exe).

Табл. 9.1. Классы и файлы главной программы

Класс	Имя класса	Порожден из	Заголовочный файл	Файл реализации	Назначение
Document (Документ)	CwinHelloDoc	CDocument	WinHelloDoc.h	WinHelloDoc.cpp	Хранение данных программы. Сохранение на диске и загрузка с диска данных программы
View (Представление)	CwinHelloView	CView	WinHelloView.h	WinHelloView.cpp	Отображение данных программы. Обработка вводимой информации. Управление окном представления
Main frame window (Главное окно)	CmainFrame	CFrameWnd	MainFrm.h	MainFrm.cpp	Управление главным окном программы
Application (Приложение)	CwinHelloApp	CwinApp	WinHello.h	WinHello.cpp	Общие задачи программы

Табл. 9.2. Дополнительные файлы, сгенерированные мастером Application Wizard

Файл	Назначение
Resource.h	Определение констант для ресурсов программы. Сопровождается редактором ресурсов Visual Studio (вы не можете редактировать его непосредственно). Подключается неявным образом ко всем главным файлам .cpp и главным файлам определения ресурсов (WinHello.rc)
StdAfx.cpp и StdAfx.h	Используются для создания предварительно компилируемых заголовков
WinHello.dsp	Установки и другая информация проекта WinHello
WinHello.sln	Информация о рабочей области проекта WinHello. Рабочая область проекта управляет одним или несколькими проектами (см. гл. 2). Чтобы открыть проект WinHello в VC, выберите в меню File команду Open... и файл WinHello.sln
WinHello.rc	Главный файл определения ресурсов программы. Определяет таблицу командных клавиш, диалоговое окно "About", меню, таблицу строк, информацию о версии программы. Сопровождается редактором ресурсов Visual Studio (его нельзя редактировать непосредственно). Обработывается компилятором ресурсов Microsoft (RC.EXE) при построении программы
res\WinHello.ico	Файл значка главной программы. Первоначально хранит стандартный значок MFC. Его можно редактировать, используя графический редактор Visual Studio. Значок отображается в верхнем левом углу окна WinHello, на полосе задач и в других местах. WinHello.rc содержит оператор ICON, включающий в ресурсы программы этот значок
res\WinHelloDoc.ico	Файл значка документа. Отображается он в программах MDI (гл. 17). Программа WinHello его не отображает
res\WinHello.rc2	Определение ресурсов программы вручную, т. е. без использования интерактивного редактора ресурсов, предоставляемого Visual Studio. Изначально не содержит определений. При желании определить ресурсы вручную можно ввести их определения в этот файл. Подключается с помощью оператора #include к главному файлу ресурсов WinHello.rc

Листинги 9.1—9.8 содержат тексты файлов заголовков и файлов реализации для четырех главных классов программы, код, созданный Application Wizard, и код, добавленный вручную в упражнениях этой главы. Файлы, созданные в упражнениях, должны совпадать с приведенными ниже (кроме перевода комментариев и построчной разбивки строк, выполненной так, чтобы поместить листинг в формат книги). В качестве примеров программ в книге приводятся только файлы на языке C++, определяющие и реализующие главные классы, так как именно с ними приходится работать при написании программы. Файлы StdAfx на C++ (StdAfx.h и StdAfx.cpp) и другие исходные файлы, перечисленные в табл. 9.2, редко непосредственно просматриваются или редактируются. Они создаются и обслуживаются с помощью различных инструментов разработки неявным образом.

---

### Листинг 9.1

```
// WinHello.h : основной файл заголовков приложения WinHello
//
#pragma once

#ifdef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCN
#endif

#include "resource.h"          // основные символы

// CWinHelloApp:
// Смотрите реализацию данного класса в файле WinHello.cpp

class CWinHelloApp : public CWinApp
{
public:
    CWinHelloApp();

    // Переопределения
public:
    virtual BOOL InitInstance();

    // Реализация
    afx_msg void OnAppAbout();
    DECLARE_MESSAGE_MAP()
};
```

---

### Листинг 9.2

```
// WinHello.cpp : Управляет работой класса приложения

#include "stdafx.h"
#include "WinHello.h"
#include "MainFrm.h"

#include "WinHelloDoc.h"
#include "WinHelloView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```

// Класс CWinHelloApp

BEGIN_MESSAGE_MAP(CWinHelloApp, CWinApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
    // Стандартные команды работы с файлами документов
    ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
END_MESSAGE_MAP()

//Конструктор CWinHelloApp

CWinHelloApp::CWinHelloApp()
{
    // TODO: добавьте сюда собственный код конструктора
    // Поместите все существенные команды инициализации
    // в функцию InitInstance
}

// Единственный объект класса CWinHelloApp

CWinHelloApp theApp;

// Инициализация класса CWinHelloApp

BOOL CWinHelloApp::InitInstance()
{
    CWinApp::InitInstance();

    // Стандартная инициализация
    // Если вы не используете эти функции и хотите уменьшить размер
    // результирующего исполняемого модуля, удалите ненужные
    // команды, выполняющие специфическую инициализацию
    // Замените код регистрации; под которым хранятся ваши установки
    // TODO: Замените эту строку на что-нибудь подходящее,
    // например, название вашей фирмы
    SetRegistryKey(_T("Local AppWizard-Generated Applications"));
    LoadStdProfileSettings(4); // Загрузка стандартных установок из
                               // INI-файла (включая список
                               // последних открытых файлов)

    // Регистрация шаблонов документов приложения. Шаблоны
    // документов служат связью между документами, окном
    // представления и главным окном
    CSingleDocTemplate* pDocTemplate;
    pDocTemplate = new CSingleDocTemplate(
        IDR_MAINFRAME,
        RUNTIME_CLASS(CWinHelloDoc),
        RUNTIME_CLASS(CMainFrame),           // главное окно
                                              // SDI-приложения
        RUNTIME_CLASS(CWinHelloView));
    AddDocTemplate(pDocTemplate);
    // Просмотр командной строки при запуске приложения. Поиск
    // стандартных команд оболочки, DDE, открытия файла
    CCommandLineInfo cmdInfo;
    ParseCommandLine(cmdInfo);
}

```

```

        // Исполнение команд, обнаруженных в командной строке.
        // Будет возвращено FALSE, если приложение было запущено
        // с /RegServer, /Register, /Unregserver или /Unregister.
        if (!ProcessShellCommand(cmdInfo))
            return FALSE;
        // Было инициализировано единственное окно, которое будет
        // отображено и обновлено
        m_pMainWnd->ShowWindow(SW_SHOW);
        m_pMainWnd->UpdateWindow();
        // вызвать DragAcceptFiles только, если есть приставка
        // В SDI-приложении, это произойдет после ProcessShellCommand
        return TRUE;
    }

    // CAboutDlg диалог, используемый в окне About приложения

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

        // Данные для диалога
        enum { IDD = IDD_ABOUTBOX };

    protected:
        virtual void DoDataExchange(CDataExchange* pDX);    //поддержка
                                                           //DDX/DDV

        // реализация
    protected:
        DECLARE_MESSAGE_MAP()
    };

    CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
    {
    }

    void CAboutDlg::DoDataExchange(CDataExchange* pDX)
    {
        CDialog::DoDataExchange(pDX);
    }

    BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    END_MESSAGE_MAP()

    // Команда приложения, выполняющая диалог
    void CWinHelloApp::OnAppAbout()
    {
        CAboutDlg aboutDlg;
        aboutDlg.DoModal();
    }

    // CWinHelloApp обработчики сообщений

```

### Листинг 9.3

```
// WinHelloDoc.h : интерфейс класса CWinHelloDoc

#pragma once

class CWinHelloDoc : public CDocument
{
protected:
    char *m_Message;

public:
    char *GetMessage ()
    {
        return m_Message;
    }
protected: // используются только для сериализации
    CWinHelloDoc();
    DECLARE_DYNCREATE(CWinHelloDoc)

    // Атрибуты
public:

    // Операции
public:

    // Переопределения
public:
    virtual BOOL OnNewDocument();
    virtual void Serialize(CArchive& ar);

    // Реализация
public:
    virtual ~CWinHelloDoc();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif
protected:

    // Сгенерированные обработчики сообщений
protected:
    DECLARE_MESSAGE_MAP()
};
```

---

### Листинг 9.4

```
// WinHelloDoc.cpp : реализация класса CWinHelloDoc

#include "stdafx.h"
#include "WinHello.h"

#include "WinHelloDoc.h"

#ifdef _DEBUG
```



```

#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

//Класс CWinHelloDoc

IMPLEMENT_DYNCREATE(CWinHelloDoc, CDocument)

BEGIN_MESSAGE_MAP(CWinHelloDoc, CDocument)
END_MESSAGE_MAP()

// Конструктор/деструктор CWinHelloDoc

CWinHelloDoc::CWinHelloDoc()
{
    // TODO: добавьте сюда одноразовый код конструктора
    m_Message="Привет!";
}

CWinHelloDoc::~CWinHelloDoc()
{
}

BOOL CWinHelloDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;

    // TODO: добавьте сюда код реинициализации
    // (SDI-документы будут использовать этот документ многократно)

    return TRUE;
}

// Сериализация CWinHelloDoc

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

// Диагностика класса CWinHelloDoc

#ifdef _DEBUG
void CWinHelloDoc::AssertValid() const
{
    CDocument::AssertValid();
}

```

```

void CWinHelloDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}
#endif // _DEBUG

// Другие функции класса CWinHelloDoc

```

---

### Листинг 9.5

```

// MainFrm.h : интерфейс класса CMainFrame

#pragma once
class CMainFrame : public CFrameWnd
{
protected: // используются только для сериализации
    CMainFrame();
    DECLARE_DYNCREATE(CMainFrame)

// Атрибуты
public:

// Операции
public:

// Переопределения
public:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);

// Реализация
public:
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected: // встроенные члены панели управления
    CStatusBar m_wndStatusBar;
    CToolBar m_wndToolBar;

// Сгенерированные обработчики сообщений
protected:
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    DECLARE_MESSAGE_MAP()
};

```

---

### Листинг 9.6

```

// MainFrm.cpp : реализация класса CMainFrame

#include "stdafx.h"
#include "WinHello.h"

```

```

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

//Класс CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
    ON_WM_CREATE()
END_MESSAGE_MAP()

static UINT indicators[] =
{
    ID_SEPARATOR,           // индикатор строки состояния
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
};

//Конструктор/деструктор класса CMainFrame

CMainFrame::CMainFrame()
{
    // TODO: добавьте сюда код инициализации членов класса
}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD |
WS_VISIBLE | CBRS_TOP | CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY |
CBRS_SIZE_DYNAMIC) || !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
    {
        TRACE0("Failed to create toolbar\n");
        return -1;      // не удалось создать
    }

    if (!m_wndStatusBar.Create(this) ||
        !m_wndStatusBar.SetIndicators(indicators,
            sizeof(indicators)/sizeof(UINT)))
    {
        TRACE0("Failed to create status bar\n");
        return -1;      // не удалось создать
    }
}

```

```

        // TODO: Удалите эти три строки, если вы не хотите, чтобы панель
        // инструментов была паркующей
        m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
        EnableDocking(CBRS_ALIGN_ANY);
        DockControlBar(&m_wndToolBar);

        return 0;
    }

    BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
    {
        if( !CFrameWnd::PreCreateWindow(cs) )
            return FALSE;
        // TODO: Измените класс или стили окна здесь
        //      изменением полей структуры cs

        return TRUE;
    }

    //Диагностика класса CMainFrame

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
    CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
    CFrameWnd::Dump(dc);
}

#endif //_DEBUG

// Обработчики сообщений CMainFrame

```

---

## Листинг 9.7

```

// WinHelloView.h : интерфейс класса CWinHelloView
//

#pragma once

class CWinHelloView : public CView
{
protected: // используется только для сериализации
    CWinHelloView();
    DECLARE_DYNCREATE(CWinHelloView)

// Атрибуты
public:
    CWinHelloDoc* GetDocument() const;

// Операции
public:

```

```

// Переопределения
public:
    virtual void OnDraw(CDC* pDC);    // переопределяется для
                                     // отображения окна
                                     //представления
virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
protected:

// Реализация
public:
    virtual ~CWinHelloView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Сгенерированные обработчики сообщений
protected:
    DECLARE_MESSAGE_MAP()
};

#ifdef _DEBUG // отладочная версия находится в файле WinHelloView.cpp
inline CWinHelloDoc* CWinHelloView::GetDocument() const
{ return (CWinHelloDoc*)m_pDocument; }
#endif

```

---

### Листинг 9.8

```

// WinHelloView.cpp : реализация класса CWinHelloView
//

#include "stdafx.h"
#include "WinHello.h"

#include "WinHelloDoc.h"
#include "WinHelloView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

//Класс CWinHelloView

IMPLEMENT_DYNCREATE(CWinHelloView, CView)

BEGIN_MESSAGE_MAP(CWinHelloView, CView)
END_MESSAGE_MAP()

//Конструктор/деструктор класса CWinHelloView

```

```

CWinHelloView::CWinHelloView()
{
    // TODO: добавьте сюда собственный код конструктора
}

CWinHelloView::~CWinHelloView()
{
}

BOOL CWinHelloView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Измените класс и стили окна здесь,
    // изменяя поля структуры cs

    return CView::PreCreateWindow(cs);
}

//Отображение окна класса CWinHelloView
void CWinHelloView::OnDraw(CDC* pDC)
{
    CWinHelloDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    // TODO: сюда добавьте код отображения собственных данных

    RECT ClientRect;
    GetClientRect (&ClientRect);
    pDC->DrawText
        (pDoc->GetMessage (), //получить строку
        -1,
        &ClientRect,
        DT_CENTER | DT_VCENTER | DT_SINGLELINE);
}

//Диагностика класса CWinHelloView
#ifdef _DEBUG
void CWinHelloView::AssertValid() const
{
    CView::AssertValid();
}

void CWinHelloView::Dump(CDumpContext& dc) const
{
    CView::Dump(dc);
}

CWinHelloDoc* CWinHelloView::GetDocument() const // неотладочная
                                                    // версия встроена
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CWinHelloDoc)));
    return (CWinHelloDoc*)m_pDocument;
}
#endif // _DEBUG

//Обработчики сообщений класса CWinHelloView

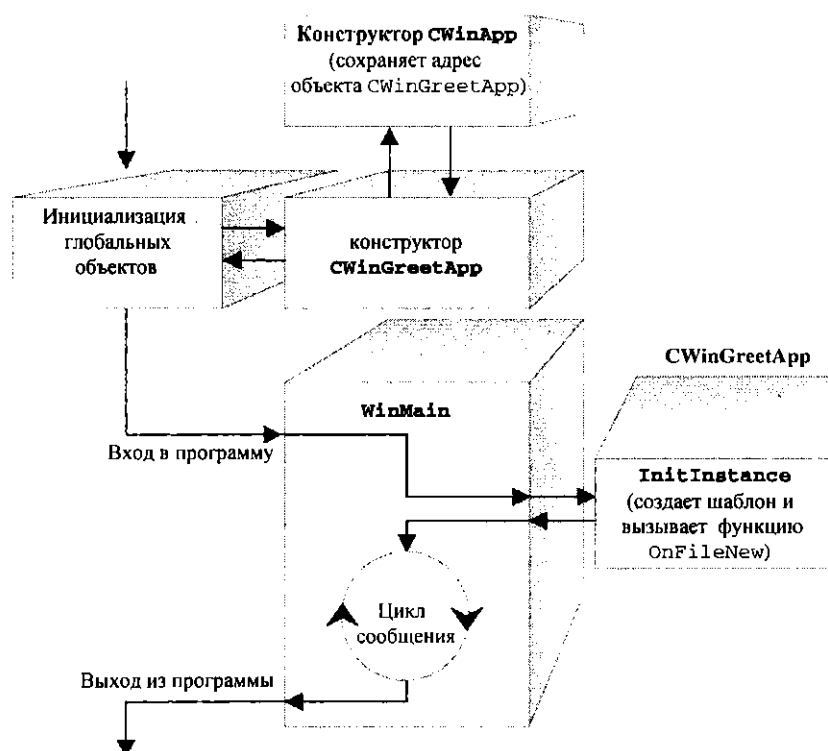
```

## Как работает программа

Для отладки программы важно понимать, как спроектированная программа WinHello работает: как получает управление, что выполняет, каким образом завершается и т.д. В этом параграфе кратко описывается общий поток программного управления, а затем рассматриваются действия, выполняемые функцией инициализации приложения `InitInstance`. В следующих главах вы изучите работу других частей кода (например, обработчики сообщений в гл. 10).

### Последовательность выполнения программы

При запуске программы WinHello происходит ряд событий, список которых приведен ниже. Эти *пять событий* были выбраны из множества выполняемых действий программы, так как именно они помогут лучше всего понять, как работает WinHello, и проиллюстрируют назначение различных частей исходного кода. На иллюстрации показана эта последовательность событий, а в следующих параграфах подробно описано каждое из них.



#### 1. Вызов конструктора класса `CWinApp`.

Приложение, построенное на основе MFC, должно определять в точности один экземпляр класса приложения. Файл `WinHello.cpp` создает экземпляр класса `CWinHelloApp` приложения WinHello посредством следующего глобального определения.

```
// Единственный объект класса CWinHelloApp  
  
CWinHelloApp theApp;
```

Объект класса CWinHelloApp определен глобально, поэтому конструктор класса вызывается *перед* тем, как входная функция WinMain получает управление. Конструктор CWinHelloApp, сгенерированный мастером Application Wizard (также находящийся в файле WinHello.cpp), не выполняет ничего.

```
//Конструктор CWinHelloApp

CWinHelloApp::CWinHelloApp()
{
    // TODO: добавьте сюда собственный код конструктора
    // Поместите все существенные команды инициализации
    // в функцию InitInstance
}
```

Даже такой ничего не выполняющий конструктор (см. гл. 5) приводит к вызову компилятором конструктора базового класса CWinApp (предоставляемого библиотекой MFC), который выполняет две важные операции:

- *Проверяет*, объявлен ли в программе *только один* объект приложения (т.е. только один объект принадлежит классу CWinApp или классу, порожденному от него).
- *Сохраняет* адрес объекта CWinHelloApp в глобальном указателе, объявляемом в библиотеке MFC. Это необходимо для того, чтобы MFC могла впоследствии *вызывать* (на этапе 3) функции класса CWinHelloApp.

2. *Получение управления функцией WinMain.*

Когда все глобальные объекты созданы, управление передается входной функции WinMain, которая определена внутри MFC. Она присоединяется к программе WinHello при построении выполняемого файла и решает множество задач, описанных далее.

3. *Вызов функции InitInstance функцией WinMain.*

Получив управление, функция WinMain вскоре вызывает функцию InitInstance класса CWinHelloApp, используя адрес объекта, сохраненный конструктором класса CWinApp на этапе 1. Функция InitInstance служит для инициализации приложения и будет описана далее в этой главе. MFC сохраняет в указателе класса CWinApp адрес объекта CWinHelloApp, используемый для вызова функции InitInstance. Так как последняя является виртуальной функцией (см. гл. 5), то управление передается переопределенной версии InitInstance, заданной внутри класса CWinHelloApp. Класс CWinApp содержит еще несколько виртуальных функций, которые можно переопределить. Например, чтобы выполнить завершающую очистку, можно переопределить функцию ExitInstance, выполняемую непосредственно перед завершением работы приложения. Информация о переопределяемых функциях класса CWinApp приведена в справочной системе.

4. *Обработка сообщений функцией WinMain.*

Завершив инициализацию, функция WinMain входит в цикл, содержащий системные вызовы для получения и распределения всех *сообщений*, посланных объектам внутри программы WinHello. (На самом деле этот цикл содержится в функции Run, вызываемой из WinMain.) Обработка сообщений будет описана в гл. 10. В процессе выполнения приложения управление остается внутри цикла. Однако используемый в Windows 98 и Windows NT режим приоритетной многозадачности (см. гл. 22) позволяет одновременно выполняться другим программам.

5. *Выход из функции WinMain и завершение программы.*

Если в меню File программы WinHello выбрать команду Exit или команду Close в системном меню, или нажимать на кнопку Close строки заголовка окна, MFC уничтожает окно программы и вызывает функцию Win32 API ::PostQuitMessage для выхода из цикла обработки сообщений. Вслед за этим происходит возврат из функции WinMain, в результате работа приложения завершается.



## Как работает функция *InitInstance*

Функция *InitInstance* принадлежит классу *CWinHelloApp* и определяется в файле *WinHello.cpp*. Библиотека MFC вызывает эту функцию из функции *WinMain*. Ее работа заключается в инициализации приложения. При вызове функции *InitInstance* большинство традиционных графических приложений создают только главное окно программы, как того требует модель представления документа, используемая библиотекой MFC.

Однако мастер *Application Wizard* выполняет более сложные операции. Он создает *шаблон документа*, сохраняющий информацию о классах документа: *главного окна* и *представления* (не путайте термин *шаблон* в этом контексте с шаблонами языка C++, описанными в гл. 7). Шаблон документа также содержит идентификатор ресурсов программы, используемый при отображении и управлении документом (в частности, при выводе меню, значка и т.д.). Когда программа запускается впервые и создает новый документ, то с помощью шаблона документа она создает:

- *объект класса документа* – для сохранения документа;
- *объект класса представления* – для создания окна представления, отображающего документ;
- *объект класса главного окна* – для вывода главного окна программы, охватывающего окно представления.

Шаблон документа является объектом языка C++. Для SDI-приложений, таких как *WinHello*, это экземпляр MFC-класса *CSingleDocTemplate*. Шаблон документа создается и сохраняется функцией *InitInstance* внутри объекта приложения.

```
// Регистрация шаблонов документов приложения.
// Шаблоны документов служат связью между
// документами, окном представления и главным окном
CSingleDocTemplate* pDocTemplate;
pDocTemplate = new CSingleDocTemplate(
    IDR_MAINFRAME,
    RUNTIME_CLASS(CWinHelloDoc),
    RUNTIME_CLASS(CMainFrame),           // главное окно SDI-
                                         // приложения
    RUNTIME_CLASS(CWinHelloView));
AddDocTemplate(pDocTemplate);
```

Приведенный код работает так:

1. Создается указатель *pDocTemplate* на шаблон документа.
2. Для динамического создания шаблона документа, т. е. экземпляра класса *CSingleDocTemplate*, используется оператор *new*. При этом указателю *pDocTemplate* присваивается адрес объекта.
3. Конструктору класса *CSingleDocTemplate* передаются четыре параметра. Первый – идентификатор ресурсов программы, используемых при отображении и управлении документом (таблица командных клавиш, значок, меню и строка заголовка). Остальные три параметра содержат информацию о классах главного окна и представления. Информация о каждом классе передается при вызове макроса *RUNTIME\_CLASS*, возвращающего указатель на класс *CRuntimeClass*. Эта информация позволяет программе при первоначальном создании нового документа динамически создавать объекты каждого класса.
4. В функцию *AddDocTemplate* класса *CWinApp*, сохраняющую шаблон документа в объекте приложения, передается указатель на объект шаблона таким образом, чтобы шаблон был открыт.

Завершив создание шаблона документа, функция *InitInstance* вызовом функции *ParseCommandLine* извлекает командную строку, если она была передана в программу при запуске. Затем вызывается функция *ProcessShellCommand* класса *CWinApp* для обработки командной строки.

```

// Просмотр командной строки при запуске приложения. Поиск
// стандартных команд оболочки, DDE, открытия файла
CCommandLineInfo cmdInfo;
ParseCommandLine(cmdInfo);

// Исполнение команд, обнаруженных в командной строке. Будет
// возвращено FALSE, если приложение было запущено с
// /RegServer, /Register, /Unregserver или /Unregister.
if (!ProcessShellCommand(cmdInfo))
    return FALSE;

```

Если функция `ProcessShellCommand` обнаружит, что командная строка содержит имя файла, то она попытается открыть этот файл. Открытие файлов описано в гл. 12. Однако в программе `WinHello` код для открытия файла реализован не полностью. При запуске программы `WinHello` (например, посредством Visual Studio) командная строка чаще всего будет пустой. В этом случае функция `ProcessShellCommand` вызывает функцию `OnFileNew` класса `CWinApp` для создания нового пустого документа.

При вызове функции `OnFileNew` программа использует шаблон документа для создания объектов типа `CWinHelloDoc`, `CMainFrame`, `CWinHelloView`, а также для связи с главным окном и окном представления. Ресурсы, используемые для главного окна (меню, значок и т.д.), определяются идентификатором ресурсов в шаблоне документа. Так как эти объекты и окна создаются внутри функции `OnFileNew`, то явных определений объектов и вызовов функций для создания окон в программе `WinHello` нет. Функция `OnFileNew` также вызывается каждый раз, когда пользователь выбирает команду `New` в меню `File`. Однако в SDI-приложениях эти вызовы не приводят к созданию новых программных объектов. Вместо этого используются объекты и окна, созданные при первом вызове функции `OnFileNew`.

Чтобы отобразить окно и его содержимое на экране, `InitInstance` вызывает функции `ShowWindow` и `UpdateWindow` объекта главного окна. Эти функции вызываются с помощью указателя на объект главного окна, сохраненного в наследуемой из класса `CWinThread` переменной `m_pMainWnd` класса `CWinHelloApp`.

```

// Было инициализировано единственное окно, которое будет
// отображено и обновлено
m_pMainWnd->ShowWindow(SW_SHOW);
m_pMainWnd->UpdateWindow();

```

Для сохранения установок программы в системном реестре Windows (а не в файле `.ini`) функция `InitInstance` вызывает функцию `SetRegistryKey` класса `CWinApp`. При этом задается имя раздела, в котором они сохраняются.

```

// Замените код регистрации, под которым хранятся ваши
// установки
// TODO: Замените эту строку на что-нибудь подходящее,
// например, название вашей фирмы
SetRegistryKey(_T("Local AppWizard-Generated Applications"));

```

Для своего приложения, вы, конечно, можете задать другое имя раздела для сохранения параметров программы (например, использовать в качестве этого имени название вашей компании). Для этого измените строку, передаваемую в качестве параметра в функцию `SetRegistryKey`. Обратите внимание на то, что макрос `_T` преобразует строку в кодировку Unicode. В этом формате каждый символ представляется 16-битовым значением, что позволяет кодировать символы любого языка. Основные установки, сохраненные в системном реестре, – это список последних открытых документов, отображаемых в меню `File`. Иногда его называют списком MRU (Most Recently Used – самые последние использовавшиеся). Функция `InitInstance` загружает этот список, как и другие установки программы, сохраненные в системном реестре, вызывая функцию `CWinApp::LoadStdProfileSettings`.

```
LoadStdProfileSettings(4);    // Загрузка стандартных установок
                             // из INI-файла (включая список
                             // последних открытых файлов)
```

Функция `InitInstance` является местом размещения соответствующего кода при инициализации приложения.

---

## Обработка исключений в библиотеке MFC

Использование механизма разрешения исключительных ситуаций для обработки ошибок программы, генерирующей исключения, рассматривалось в гл. 8. Библиотека MFC полностью поддерживает стандартные исключения языка C++, и для работы с определенными типами ошибок MFC можно использовать методику, описанную в гл. 8. Например, при определенных условиях некоторые функции библиотеки MFC генерируют исключительные ситуации, а не просто возвращают код ошибки. В документации по функциям MFC, которые могут генерировать исключительные ситуации, описываются:

- *обстоятельства*, при которых они генерируются;
- *тип* формируемого объекта.

Эту информацию можно использовать, чтобы написать соответствующий обработчик `catch`. Функции библиотеки MFC всегда выдают указатель на объект класса, порожденного от класса `CException` (например, классов `CFileException` и `CMemoryException`). Когда блок `catch` закончит работу с объектом исключения, обработчик вызовет функцию `Delete` объекта, чтобы правильно его удалить. Например, в следующем фрагменте программы вызывается конструктор MFC-класса `CFile` для создания объекта файла и открытия файла. В гл. 12 рассматривается применение класса `CFile` для выполнения ввода/вывода файлов....

```
try
{
    CFile File ("EXISTS.NOT", CFile::modeRead);
}
catch (PtrFileException *PtrFileException)
{
    switch (PtrFileException->m_cause)
    {
        case CFileException::fileNotFound:
            AfxMessageBox ("File not found.");
            break;
        case CFileException::badPath:
            AfxMessageBox ("Bad path specifcation.");
            break;

        default:
            AfxMessageBox ("File open error.");
            break;
    }

    PtrFileException->Delete();
}
```

Конструктор класса `CFile` пробует открыть файл, указанный в первом параметре. В документации по конструктору утверждается, что при ошибке во время открытия файла конструктор выдает исключение класса `CFileException`. Это означает, что выдаваемое значение является *указателем* на объект класса `CFileException`. Переменная `m_cause` данного класса содержит код, определяющий ошибку доступа к файлу. Класс `CFileException` содержит множество констант, которые

можно присвоить переменной `m_cause` (например, `CFileException::fileNotFound` и `CFileException::badPath`). Заключительная операция в блоке `catch` удаляет объект исключения библиотеки MFC.

При неудачной попытке распределения памяти в MFC-программе оператор `new` вызывает исключение класса `CMemoryException`, а не просто возвращает 0. В приведенном ниже примере: если оператор `new` выполняется неудачно, то вместо того, чтобы вернуть 0, он вызывает исключение класса `CMemoryException`. Это приводит к передаче управления блоку `catch`.

```
try
{
    // ...
    char *PtrBigBuffer = new char [200000000]; // слишком
                                              // большой блок!
    // ...
}
catch (CMemoryException *)
{
    AfxMessageBox ("CMemoryException thrown");
}
```

Чтобы сэкономить место и сосредоточиться на сути приемов программирования, в примерах не приведены коды обработки исключений. Однако в собственных программах можно обрабатывать исключения по методике, описанной в гл. 8. Информация об исключениях дана в документации по MFC-функциям. Дополнительная информация по MFC-исключениям имеется в справочной системе.

---

## Резюме

---

Мы рассмотрели библиотеку MFC, мастера генерации приложений Application Wizard, а также методику создания простого графического приложения. Перечислим наиболее общие понятия и подходы.

- Приложения в Visual C++ пишутся в рамках одного из трех базовых подходов:
  - программирование вручную с обращением к функциям Win32 API;
  - программирование вручную с использованием MFC;
  - создание MFC-программ с помощью мастеров. Третий подход – единственный подробно рассмотренный в данной книге – для создания стандартных программ Windows является самым простым.
- Графические приложения – это программы, использующие графический интерфейс пользователя (GUI) Windows.
- При создании нового проекта в Visual Studio.NET основные исходные файлы для графических приложений генерируются выбором типа проекта “MFC Application” и заданием на вкладках диалогового окна мастера Application Wizard требуемых свойств программы.
- Собственные функции добавляются:
  - либо непосредственным редактированием исходного кода;
  - либо с использованием различных инструментов Visual C++ (например, редактора ресурсов, предоставляемого пакетом Visual Studio.NET).
- Созданные мастером Application Wizard графические MFC-программы содержат четыре главных класса. Задачи программы распределены между этими классами.
- Производный от MFC-класса `CDocument` класс документа, отвечает за хранение данных программы, а также за чтение этих данных с носителя и запись на носитель.

- Производный от `CView` класс представления, управляет окном представления. Он отвечает за отображение документа внутри окна представления и на других устройствах, за обработку информации, вводимой пользователем.
- Производный от `CFrameWnd` класс главного окна, управляет главным окном программы, которое отображает такие объекты пользовательского интерфейса, как рамка окна; меню; строка заголовка; кнопки сворачивания, разворачивания и закрытия окна; иногда панель инструментов или строку состояния. Окно представления располагается внутри этого окна в пустой области.
- Производный от `CWinApp` класс приложения, управляет приложением в целом и решает такие общие задачи, как инициализация приложения и очистка перед завершением.
- Точка входа в программу – функция `WinMain` – определена внутри библиотеки MFC. Эта функция вызывает функцию `InitInstance` класса приложения и входит в цикл обработки передаваемых объектам программы сообщений.
- Для инициализации программы используется функция `InitInstance`. Она создает и сохраняет шаблон документа, хранящий информацию о классах документа программы, окна представления и главного окна. Затем `InitInstance` вызывает функцию `OnFileNew` класса `CWinApp`, использующую шаблон документа для создания экземпляров этих трех классов, а также окна представления и главного окна.

## Глава 10

### Как оформить представление

---

- Простой графический редактор ScratchBook
- Простой текстовый редактор MyScribe

Обычно, *представлением* называют часть программы, которая использует библиотеку MFC для решения следующих задач:

- управление окном просмотра;
- обработка вводимой пользователем информации;
- отображения документа в окне.

В предыдущей главе мы рассмотрели, как при создании программы с помощью мастера Application Wizard от класса CView библиотеки MFC порождается специальный *класс управления представлением*, который служит шаблоном для внесения программистом собственного кода. В гл. 9 в класс представления программы WinHello добавлен текст для отображения строки сообщения в окне представления. В этой главе описана первая версия простой программы рисования, называемой ScratchBook. После того как Application Wizard создаст основной шаблон программы, в класс представления будет добавлен код для отслеживания действий мыши и рисования прямых линий в окне представления. Для создания обработчиков сообщений мыши и настройки окна представления используется окно Properties для отдельно выбранного класса CScratchBookView. Для изменения меню программы, панели инструментов и конструирования значка – редакторы ресурсов, предоставляемые пакетом Visual Studio.NET. В следующих главах разработаны более сложные версии программы ScratchBook.

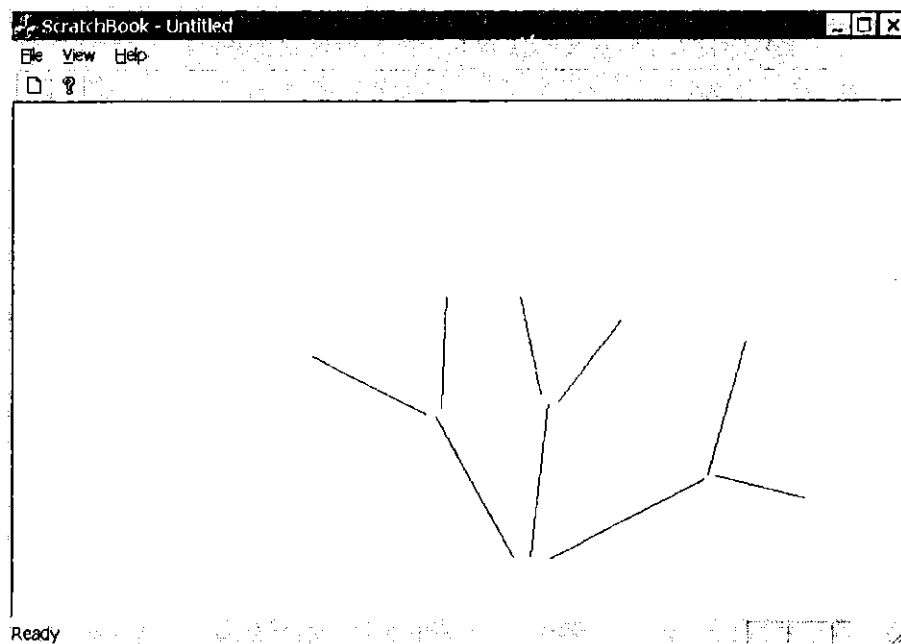
Далее в этой главе описана первичная версия простого текстового редактора MyScribe. При создании функционально полного текстового редактора класс представления порождается от MFC-класса CEditView, а не CView. В следующих главах рассмотрены более сложные версии программы MyScribe.

### Простой графический редактор

---

В качестве примера, позволяющего продемонстрировать реализацию основных функций класса представления, в этой главе генерируется первая версия программы ScratchBook. Программа позволяет рисовать прямые линии внутри окна представления. Чтобы нарисовать линию, необходимо:

1. Поместить указатель мыши в начальную позицию линии.
2. Нажать левую кнопку мыши.
3. Перетащить указатель в конечную позицию.
4. Отпустить кнопку мыши.



В данном простом примере при каждой перерисовке окна (например, при изменении размеров окна, удалении перекрывающего окна или при выборе команды New в меню File) все линии удаляются. В гл. 11 реализованы основные функции класса документа, *сохраняющие* данные для каждой нарисованной линии, что позволяет при перерисовке окна восстанавливать линии. В отличие от программы WinHello, рассмотренной в предыдущей главе, ScratchBook *не реализует* функцию OnDraw, так как в такой версии программы не сохраняются данные, необходимые для восстановления линий. Однако версия программы ScratchBook, рассмотренная в гл. 11, запоминает эти данные и реализует упомянутую функцию. В следующих главах в эту программу добавится множество других компонентов, таких как инструменты для рисования различных фигур и команды сохранения/загрузки рисунков с диска.

## Исходные файлы редактора

Как и во всех примерах, приведенных в этой части книги, исходный текст программы можно создать и изменить самостоятельно, следуя инструкциям, приведенным ниже. Для создания исходных файлов программы используйте мастер Application Wizard так же, как и в предыдущей главе (в параграфе “Проектирование программы”). В поле *Name* диалогового окна New Project введите имя ScratchBook, а в поле *Location* – путь к каталогу проекта. В диалоговых окнах мастера Application Wizard убедитесь в выборе *тех же* установок, что и при генерации программы WinHello в предыдущей главе. После завершения генерации исходных файлов Application Wizard отображает их имена во вкладке Solution Explorer.

При редактировании исходных файлов программ и ресурсов (ресурсы будут рассмотрены позже) желательно периодически сохранять результаты своей работы. Простейший способ сохранения – выбор команды Save All из меню File или щелчок на Save All панели инструментов Standard. Чтобы возобновить работу с программой после выхода из среды VS.NET или закрытия проекта ScratchBook:

1. Выберите в меню File команду Open...
2. Выберите файл “решения” (solution) проекта ScratchBook.sln.

## Переменные класса представления

Добавьте в класс представления несколько переменных: `m_ClassName`, `m_Dragging`, `m_HCross`, `m_PointOld` и `m_PointOrigin`. Для этого откройте файл `ScratchBookView.h` и добавьте в начало определения класса `CScratchBookView` выражения, выделенные полужирным шрифтом (назначение этих элементов описано ниже):

```
class CScratchBookView : public CView
{
protected:
    CString m_ClassName;
    int m_Dragging;
    HCURSOR m_HCross;
    CPoint m_PointOld;
    CPoint m_PointOrigin;

protected:    // используется только для сериализации
    CScratchBookView();
    DECLARE_DYNCREATE(CScratchBookView)
```

В конструктор класса `CScratchBookView` в файле `ScratchBookView.cpp` следует добавить для переменных `m_Dragging` и `m_HCross` код инициализации.

```
// Конструктор/деструктор класса CScratchBookView

CScratchBookView::CScratchBookView()
{
    // TODO: добавьте сюда собственный код конструктора

    m_Dragging = 0;
    m_HCross = AfxGetApp()->LoadStandardCursor (IDC_CROSS);
}
```

В переменной `m_HCross` хранится вид дескриптора указателя мыши, отображаемого программой во время, когда этот указатель находится внутри окна представления. Функция `AfxGetApp` возвращает указатель на объект класса приложения (класс `CScratchBookApp`, порожденный от класса `CWinApp`), где он используется для вызова функции `LoadStandardCursor` класса `CWinApp`. Эта функция при получении в качестве входного параметра константы `IDC_CROSS` возвращает дескриптор стандартного крестообразного указателя. Позже вы увидите, как программа `ScratchBook` отображает указатель в окне представления (параграф “Функция `OnMouseMove`”). В следующей таблице (табл. 10.1) приведены идентификаторы констант, которые можно передать в качестве параметра в функцию `LoadStandardCursor` для получения дескрипторов других стандартных указателей.

## Обработчики сообщений

Пользователь сможет рисовать линии внутри окна представления с помощью мыши только в случае, если программа будет реагировать на события, происходящие внутри этого окна. Для обработки сообщения, передаваемого мышью, в класс представления необходимо добавить *функцию обработки сообщений*. Кратко рассмотрим сами сообщения Windows, прежде чем перейти к процедуре создания обработчиков сообщений.

### Сообщения и реакции на них

В графической программе с каждым окном связана функция, называемая *процедурой окна*. Когда происходит некоторое событие, операционная система вызывает эту функцию, передавая ей



идентификатор происшедшего события и необходимые данные для его обработки. Подобный процесс называется *передачей сообщения окну*. Процедура окна с помощью библиотеки MFC создается автоматически. Все окна в примерах программ, приведенных в этой части книги, управляются MFC. Процедуры окон выполняют минимальную стандартную обработку переданного сообщения.

Табл. 10.1. Стандартные указатели Windows, которые можно передавать функции LoadStandardCursor

Константа	Указатель
IDC_ARROW	Стандартный указатель-стрелка
IDC_CROSS	Перекрестье, используемое для позиционирования на графике
IDC_IBEAM	Указатель для позиционирования в тексте
IDC_SIZEALL	Четырехнаправленная стрелка для изменения позиции
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	Стрелка, направленная строго вверх
IDC_WAIT	“Песочные часы”, используемые при длительном выполнении операции

Если необходима собственная обработка сообщения, то создается функция обработки сообщения, являющаяся членом класса управления окном. Для определения обработчика сообщения можно воспользоваться окном Properties для выбранного класса, как описано ниже. Например, если указатель находится внутри окна представления, то при нажатии левой кнопки мыши передается сообщение с идентификатором WM\_LBUTTONDOWN. Чтобы предусмотреть собственную обработку этого сообщения, используйте окно Properties для создания обрабатывающей данное сообщение функции класса представления.

Библиотека MFC обеспечивает специальную обработку сообщений, генерируемых *объектами пользовательского интерфейса* (стандартными элементами, поддерживаемыми библиотекой MFC):

- меню;
- сочетания клавиш;
- кнопки панелей инструментов;
- строки состояния;
- элементы управления диалоговых окон.

Термин *объект* в данном случае относится не к объектам языка C++. Меню рассматриваются в этой и следующей главах, остальные объекты пользовательского интерфейса – в гл. 14. Сообщения, генерируемые объектами пользовательского интерфейса, называются *командными сообщениями*. Процедура их обработки выглядит так:

1. Каждый раз, когда пользователь выбирает объект интерфейса или когда один из этих объектов необходимо обновить, объект передает командное сообщение главному окну.
2. Библиотека MFC сразу направляет сообщение объекту окна представления. Если он не имеет нужного обработчика, библиотека MFC направляет сообщение объекту документа.
3. Если объект документа не содержит обработчика, библиотека MFC направляет сообщение объекту главного окна программы.
4. Если главное окно также не располагает обработчиком, сообщение направляется объекту приложения.

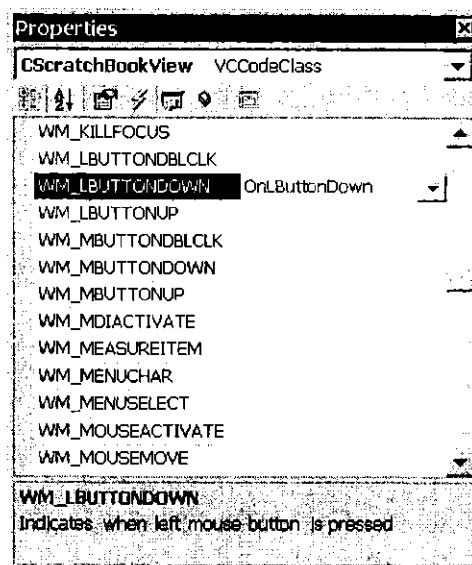
5. Наконец, если объект приложения не обеспечивает обработку, то сообщение обрабатывается стандартным образом.

Благодаря описанной процедуре библиотека MFC *расширяет* основной механизм обработки сообщений, чтобы командные сообщения обрабатывались не только объектами, управляющими окнами, но и любыми другими объектами приложения. Каждый из них принадлежит классу, прямо или косвенно порожденному от класса `CCommandTarget`, 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 Application Wizard, команда Exit в меню File обрабатывается классом приложения, так как эта команда воздействует на приложение в целом. Но 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, как создавать обработчики сообщений и научиться определять, какому именно классу следует обрабатывать конкретное сообщение.

## Обработчик сообщений `OnLButtonDown`

Далее необходимо разработать обработчик сообщения `WM_LBUTTONDOWN`, которое передается при каждом нажатии левой кнопки мыши, если указатель находится внутри окна представления. Для определения функции обработчика:

1. Убедитесь, что открыт для редактирования файл `ScratchBookView.cpp`, откройте вкладку Class View в меню View, и, выбрав на ней класс `CScratchBookView`, из его контекстного меню выберите команду Properties. Откроется окно Properties.
2. В окне Properties (для класса `CScratchBookView`) откройте страницу Messages, позволяющую определить обработчик сообщений.
3. В списке выберите пункт `CScratchBookView`. Выбор класса `CScratchBookView` задает функцию-член для обработки любого уведомляющего сообщения, переданного окну представления, что позволяет переопределить одну из виртуальных функций, которую класс `CScratchBookView` наследует от `CView` и других базовых классов библиотеки MFC. Выбор одного из пунктов списка в нижней части вкладки Properties создает основу для кода обработчика выбранного сообщения.



4. В списке сообщений выберите идентификатор WM\_LBUTTONDOWN, обрабатываемый определяемой функцией. Список сообщений (Messages) содержит идентификаторы всех уведомляющих сообщений, которые передаются окну представления (идентификаторы сообщений – это константы, записанные заглавными буквами и начинающиеся с префикса WM\_). Список переопределяемых функций содержит имена всех виртуальных функций, принадлежащих классу CView. Их можно выбирать, чтобы переопределять стандартные функции (параграф “Удаление данных” гл. 11). При выборе конкретного идентификатора сообщения или виртуальной функции соответствующее краткое описание появляется в нижней части вкладки.
5. Выполните щелчок левой кнопкой мыши на кнопке Add Function. Теперь будет создан шаблон обработчика сообщения с именем OnLButtonDown. В частности, функция будет объявлена в определении класса CScratchBookView в файле ScratchBookView.h, в файл ScratchBookView.cpp будет внесено ее определение, и функция будет добавлена в *карту сообщений* класса (ее структура описана ниже в этом параграфе).
6. В окне редактора в откройте файл ScratchBookView.cpp и найдите в нем функцию OnLButtonDown. Добавьте в эту функцию выделенные полужирным шрифтом операторы.

```
void CScratchBookView::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: Вставьте сюда собственный код обработчика
    // или вызов стандартного

    m_PointOrigin = point;
    m_PointOld = point;
    SetCapture ();
    m_Dragging = 1;
    RECT Rect;
    GetClientRect (&Rect);
    ClientToScreen (&Rect);
    ::ClipCursor (&Rect);

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

### Примечание

При создании обработчика сообщений в функцию OnLButtonDown автоматически включается строка, вызывающая версию функции OnLButtonDown, определенную в базовом классе. Это делается для того, чтобы базовый класс мог выполнять стандартную обработку сообщений.

После нажатия левой кнопки мыши (если указатель находится в окне представления) 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 используется другими обработчиками сообщений для получения информации о положении указателя мыши в момент предыдущего сообщения.

Функция SetCapture класса CWnd выполняет *захват мыши*, и все последующие сообщения мыши 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 другие обработчики сообщений о выполнении операции рисования. Так как класс CScratchBookView неявно порождается от MFC-класса CWnd, он наследует все его функции. (CScratchBookView является потомком CView,

который, в свою очередь, порождается от CWnd.) В этой книге, когда встречается ссылка на определенную функцию, указывается класс, в котором она определена. Например, SetCapture называется “функцией SetCapture класса CWnd” или (при использовании операции расширения области видимости) просто “CWnd::SetCapture”. Чтобы узнать все функции, доступные классу, обратитесь к иерархии классов, описанной в документации по классам в справочной системе. Затем просмотрите документацию по каждому из базовых классов, а также описание функций-членов.

Далее в программе расположен фрагмент, предназначенный для ограничения перемещения указателя мыши границами окна представления. Функция CWnd::GetClientRect возвращает текущие координаты окна представления, а CWnd::ClientToScreen преобразует их в *экранные* (т.е. координаты, заданные по отношению к левому верхнему углу экрана). Наконец, функция ::ClipCursor ограничивает перемещения указателя в пределах заданных координат, удерживая его в окне представления. Функция ::ClipCursor принадлежит Win32 API, а не 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. Чтобы показать, что функция не принадлежит библиотеке MFC, в этой книге каждой функции Win32 API всегда предшествует операция расширения области видимости.

---

## Карта сообщений

Когда создается обработчик сообщения, то помимо объявления и определения функции-члена он также добавляет ее в специальную структуру MFC, называемую *картой сообщений (message map)* и связывающую функции с обрабатываемыми сообщениями. Карта сообщений позволяет библиотеке MFC вызывать для каждого типа сообщения соответствующий обработчик. Мастер Application Wizard создает необходимый код для реализации карты сообщений, основанной на наборе MFC-макросов. После определения обработчиков трех видов сообщений мыши, описанных в этой главе, в файл ScratchBookView.h будут добавлены следующие объявления функций и макросы:

```
// Сгенерированные функции карты сообщений
protected:
    DECLARE_MESSAGE_MAP()
public:
    afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
    afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
    afx_msg void OnMouseMove(UINT nFlags, CPoint point);
```

Следующие макросы будут добавлены в файл реализации представления ScratchBookView.cpp:

```
BEGIN_MESSAGE_MAP(CScratchBookView, CView)
    ON_WM_LBUTTONDOWN()
    ON_WM_LBUTTONUP()
    ON_WM_MOUSEMOVE()
END_MESSAGE_MAP()
```

При передаче сообщения объекту класса:

1. MFC обращается к карте сообщений, чтобы определить, есть ли в классе обработчик такого сообщения. Если обработчик найден, ему передается управление.
2. 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.
3. Если в иерархии обработчик отсутствует, то будет выполнена стандартная обработка сообщения.

4. Если же это командное сообщение, то оно перенаправляется следующему объекту в описанной ранее последовательности.

Некоторые из классов библиотеки MFC, от которых порождены классы программы, содержат обработчики сообщений. 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). Следовательно, даже если производный класс не определяет обработчик сообщения, обработчик базового класса может обеспечить соответствующую обработку.

## Функция OnMouseMove

На следующем шаге необходимо 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 выполните последовательность действий, приведенную в предыдущем параграфе. Однако на шаге 4 выберите в списке сообщение WM\_MOUSEMOVE вместо WM\_LBUTTONDOWN. Будет создана функция OnMouseMove. В окне редактора введите для этой функции операторы, выделенные на приведенном ниже листинге полужирным шрифтом:

```
void CScratchBookView::OnMouseMove(UINT nFlags, CPoint point)
{
    // TODO: Вставьте сюда собственный код обработчика
    // или вызов стандартного

    ::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);
}
```

Перемещение указателя мыши внутри окна представления вызывает через определенные промежутки времени функцию OnMouseMove. Добавленный в нее код обеспечивает решение двух задач:

- Вызов API-функции **::SetCursor** для *отображения крестообразного указателя* вместо стандартного указателя-стрелки (дескриптор крестообразного указателя получен в конструкторе класса). Последующие версии программы ScratchBook отображают указатели в зависимости от выбранного инструмента. Техника отображения стандартного указателя в виде песочных часов во время выполнения длительной операции описана ниже.
- Выполнение *операции рисования* (значение переменной m\_Dragging отлично от нуля). При этом выполняется стирание линии, нарисованной при получении предыдущего сообщения WM\_MOUSEMOVE (если она имеется). Затем рисуется новая линия от начальной точки, в которой была нажата левая кнопка мыши с координатами, сохраняемыми в переменной m\_PointOrigin.

до текущей позиции указателя, заданной параметром `point`. В завершение в переменной `m_PointOld` сохраняется текущая позиция указателя.

Чтобы выполнить рисование внутри окна, функция `OnMouseMove` создает объект контекста устройства, связанный с окном представления. (Объекты контекста устройства описаны в гл. 19. Объект класса `CClientDC` позволяет рисовать внутри окна представления с помощью функции, *отличающейся* от `OnDraw`.) 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 рисовании в той же самой позиции – невидима. Таким образом, обработчики сообщения легко отображают и удаляют группы временных линий.

Линии выводятся с помощью функции `CDC::MoveTo`, указывающей положение одного конца линии, и `CDC::LineTo`, задающей положение другого конца. Объекты контекста устройства, функции `SetROP2`, `CDC::MoveTo` и `CDC::LineTo` описаны в гл. 19. Окончательный результат использования `OnMouseMove` состоит в том, что при перемещении указателя с нажатой кнопкой мыши внутри окна представления временная линия всегда располагает свое начало в текущей позиции указателя. Эта линия показывает, какой будет постоянная линия, если пользователь отпустит кнопку мыши в данный момент.

Любой обработчик сообщений мыши получает два параметра: `nFlags` и `point`:

- `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. 10.2). Например, в следующем фрагменте проверяется, была ли нажата клавиша `Shift` при перемещении мыши:

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

Табл. 10.2. Битовые маски для параметра `nFlags`

Битовая маска	Содержание бита устанавливается, если нажата
<code>MK_CONTROL</code>	клавиша <code>Ctrl</code>
<code>MK_LBUTTON</code>	левая кнопка мыши
<code>MK_MBUTTON</code>	средняя кнопка мыши
<code>MK_RBUTTON</code>	правая кнопка мыши
<code>MK_SHIFT</code>	клавиша <code>Shift</code>

- `point` – это структура `CPoint`, задающая координаты указателя мыши в тот момент, когда произошло событие мыши. Координаты определяют местоположение указателя относительно верхнего левого угла окна представления. 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 пересечения его линий. Переменная `x` (`point.x`) содержит горизонтальную координату острия указателя, переменная `y` (`point.y`) – вертикальную.

## Функция `OnLButtonUp`

В завершение, необходимо определить функцию для обработки передаваемого при отпускании левой кнопки мыши сообщения `WM_LBUTTONDOWN`. Чтобы создать функцию, действуйте так же, как и

для двух предыдущих сообщений. Однако в списке Messages диалогового окна мастера выберите идентификатор WM\_LBUTTONDOWN. Когда функция будет сгенерирована, добавьте в ее определение в файле ScratchBookView.cpp следующий текст, выделенный полужирным шрифтом:

```
void CScratchBookView::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: Вставьте сюда собственный код обработчика
    // или вызов стандартного
    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);
    }

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

Перемещение пользователем указателя мыши с нажатой кнопкой (значение переменной m\_Dragging отлично от нуля) приведет к выполнению добавленного кода: программа завершает операцию рисования и строит постоянную линию. При этом переменной m\_Dragging присваивается значение 0, что информирует другие обработчики сообщений о завершении операции рисования. Затем вызывается API-функция ::ReleaseCapture для завершения захвата мыши. После этого сообщения мыши будут снова передаваться любому окну, в котором находится указатель. Далее указатель NULL передается в API-функцию ::ClipCursor, что дает пользователю возможность снова перемещать указатель мыши по всему экрану. Затем выполняется стирание временной линии, выведенной предыдущим обработчиком (для сообщения WM\_MOUSEMOVE). И в завершение от начальной точки до текущей позиции указателя выводится постоянная линия. WM\_LBUTTONDOWN – это последнее сообщение мыши, которое нужно обработать в программе ScratchBook. В следующей таблице приведен полный список сообщений мыши.

Табл. 10.3. Сообщения мыши и вызвавшие их события

Сообщение	Причина (событие)
WM_LBUTTONDOWNBLCLK	Выполнен двойной щелчок левой кнопкой
WM_LBUTTONDOWN	Нажата левая кнопка
WM_LBUTTONUP	Отпущена левая кнопка
WM_MBUTTONDOWNBLCLK	Выполнен двойной щелчок средней кнопкой
WM_MBUTTONDOWN	Нажата средняя кнопка
WM_MBUTTONUP	Отпущена средняя кнопка
WM_MOUSEMOVE	Указатель мыши перемещен на новое место внутри рабочей области
WM_RBUTTONDOWNBLCLK	Выполнен двойной щелчок правой кнопкой
WM_RBUTTONDOWN	Нажата правая кнопка
WM_RBUTTONUP	Отпущена правая кнопка

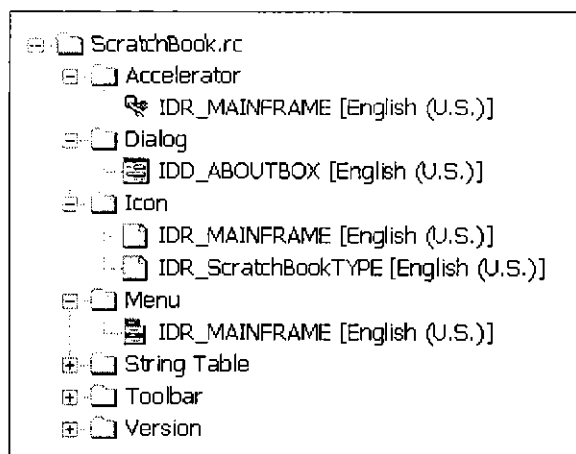
## Песочные часы – индикатор задержки

Если выполнение операции занимает достаточно много времени, то программа должна временно отобразить стандартный курсор песочных часов, тем самым сообщая пользователю о паузе в работе. (Альтернативный вариант – использование отдельного потока без прерывания обработки сообщений – описан в гл. 22.) Для этого перед началом длительной операции вызывается функция `CCmdTarget::BeginWaitCursor`, а после ее завершения – `CCmdTarget::EndWaitCursor`. Две эти функции можно вызвать из любого класса главной программы (т.е. из классов документа, представления, главного окна или приложения), так как все перечисленные классы косвенно порождены от `CCmdTarget`. Ниже приведен пример.

```
CCmdTarget::BeginWaitCursor (); // отображение курсора
                                // "песочные часы";
::Sleep (7000);                // приостановка на 7 секунд для
                                // имитации длительной операции
EndWaitCursor();               // восстановление предыдущего курсора
```

В примере вызывается API-функция `::Sleep`, которая создает паузу в 7 секунд, моделирующую длительную операцию. В этом примере подразумевается, что приведенный код содержится внутри функции класса, прямо или косвенно порожденного от `CCmdTarget` (в противном случае можно создать объект класса `CCmdTarget`, чтобы использовать его для вызова функции).

## Ресурсы программы *ScratchBook*



Пора приступить к настройке ресурсов программы *ScratchBook*. Настроим:

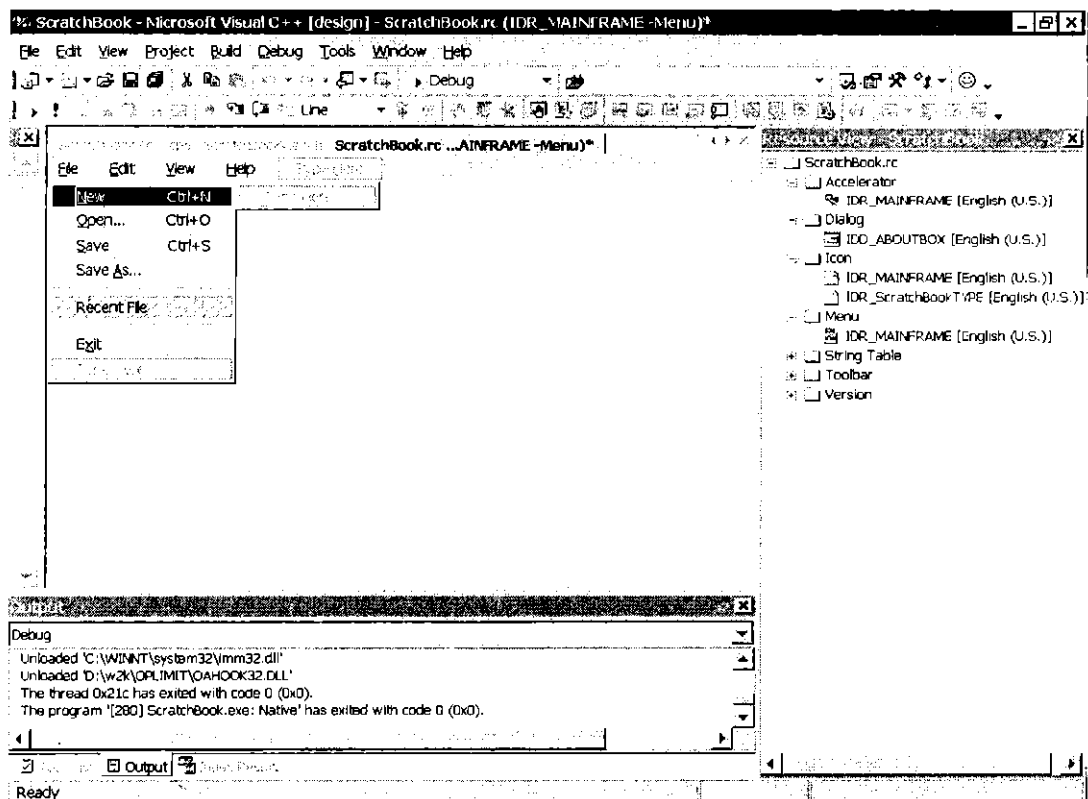
- меню;
- панель инструментов;
- значок.

В меню View Visual Studio.NET выберите команду отображения вкладки Resource View и разверните граф ресурсов программы всех категорий: Accelerator, Dialog, Icon, Menu, String Table, Toolbar и Version. Чтобы настроить меню программы:

1. Дважды щелкните на идентификаторе `IDR_MAINFRAME` в разделе Menu. Обратите внимание: идентификатор `IDR_MAINFRAME` используется также для таблицы сочетаний клавиш, панели инструментов и главного значка программы. Будет открыта вкладка редактора меню, отобра-



жающая меню программы ScratchBook, созданное мастером Application Wizard. Это окно показано на следующей иллюстрации.

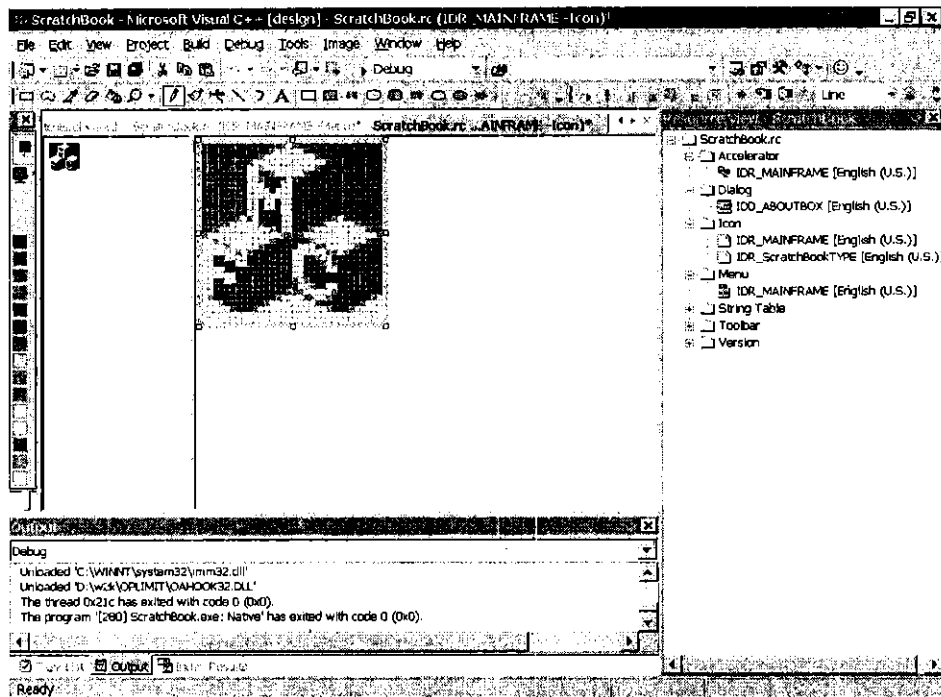


2. Щелкните на меню File на вкладке редактора. Удалите все пункты, кроме команд New, Exit и разделителя между ними.
  - Чтобы удалить пункт меню (команду или разделитель), щелкните на нем и нажмите клавишу Del.
  - Нельзя удалять пустое поле (строку) в нижней части меню. Эта строка предназначена для добавления новых пунктов и при выполнении программы в меню отсутствует.
3. Выполните щелчок мышью на меню Edit и нажмите клавишу Del, чтобы удалить его. Щелкните на ОК в окне подтверждения.

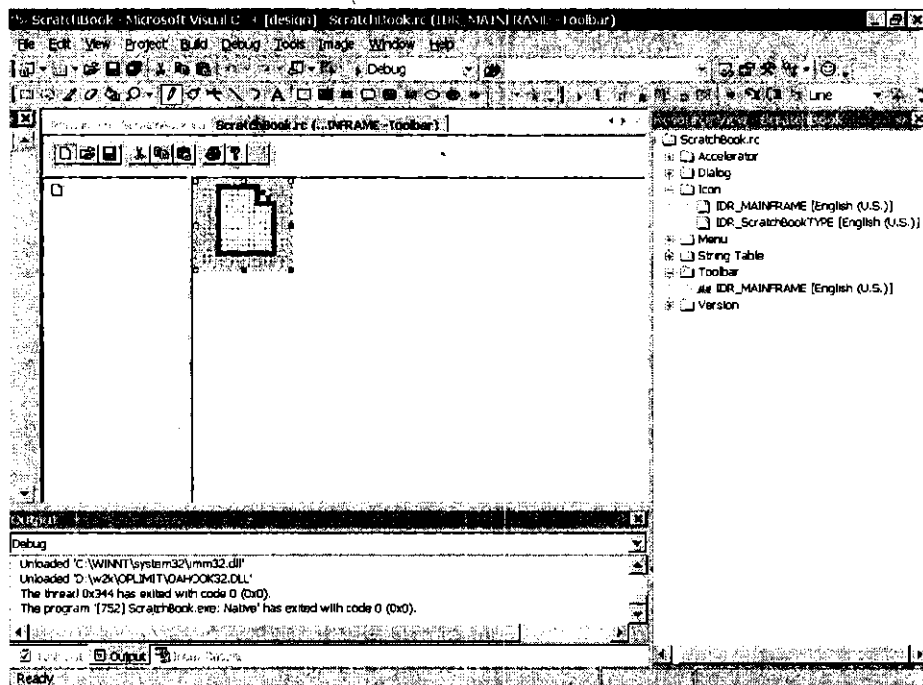
Теперь все неиспользуемые пункты меню удалены. Закройте окно редактора меню, выполнив двойной щелчок на его системном меню или щелкнув на кнопке Close.

Чтобы сконструировать свой значок программы ScratchBook (для замены стандартного значка, предоставляемого библиотекой MFC), дважды щелкните на идентификаторе IDR\_MAINFRAME в разделе Icon дерева ресурсов программы. Будет открыто окно графического редактора, отображающее текущий значок программы. На следующей иллюстрации показан графический редактор, используемый для редактирования значков.

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



Для редактирования элементов панели инструментов выполните двойной щелчок на элементе IDR\_MAINFRAME в папке Toolbar дерева ресурсов программы. Будет открыто окно редактирования панели инструментов. В нем можно изменить значки, назначенные отдельным ее элементам, или удалить эти элементы. Для удаления некоторого элемента следует выбрать его мышью на отображаемой панели инструментов и перетащить за пределы панели.



Внесенные в ресурсы программы изменения следует сохранить, выбрав в меню File команду Save All.

- Основная информация о ресурсах хранится в *файле определения ресурсов* ScratchBook.rc. 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.
- *Информация о значке* хранится в файле ScratchBook.ico подкаталога \res каталога проекта.

## Конфигурирование окна ScratchBook

Есть две проблемы, которые ставит перед нами первоначальный вариант программы ScratchBook:

- Обработчик сообщения WM\_MOUSEMOVE отображает требуемый указатель крестообразной формы. Но Windows *также* пытается отобразить стандартный курсор-стрелку, назначенный окну представления библиотекой MFC. В результате возникает неприятное мерцание из-за переходов между этими двумя формами при перемещении указателя.
- Если пользователь выбирает на системной Панели управления (утилита “Дисплей”) темный цвет для фона окна приложения, то линии, нарисованные в окне представления, становятся невидимыми или едва заметными. При создании окна библиотека MFC присваивает ему параметры, заданные в Панели управления системы.

Для устранения этих проблем следует добавить небольшой фрагмент кода в функцию PreCreateWindow класса CScratchBookView. При генерации программы мастер Application Wizard определяет шаблон функции CScratchBookView::PreCreateWindow, переопределяющей виртуальную функцию PreCreateWindow класса CView, которую MFC вызывает непосредственно перед созданием окна представления. Так как CView::PreCreateWindow – важная виртуальная функция, мастер Application Wizard автоматически генерирует перегруженную версию функции в классе представления программы. Как сгенерировать перегруженную версию *любой* виртуальной функции библиотеки MFC, вы узнаете в параграфе “Удаление данных” гл. 11. Чтобы настроить окно представления ScratchBook, добавьте операторы, отмеченные полужирным шрифтом, в функцию PreCreateWindow в файле ScratchBookView.cpp.

```
BOOL CScratchBookView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Модифицируйте класс или стили окна,
    // добавляя и изменяя поля структуры cs

    m_ClassName = AfxRegisterWndClass
        (CS_HREDRAW | CS_VREDRAW,    // стили окна
         0,                          // без указателя
         (HBRUSH)::GetStockObject (WHITE_BRUSH),
                                     // задать белый фон
         0);                         // без значка
    cs.lpszClass = m_ClassName;

    return CView::PreCreateWindow(cs);
}
```

При вызове функции PreCreateWindow ей передается ссылка на структуру CREATESTRUCT. Поля этой структуры хранят свойства окна (координаты, стили и т.д.), задаваемые библиотекой MFC при его создании. Эти значения можно изменить.

В структуре `CREATESTRUCT` есть специальное поле `lpszClass`, которое 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 имя класса присваивается полю `lpszClass` структуры `CREATESTRUCT`. Таким образом окно представления обладает настраиваемыми свойствами, сохраняемыми внутри данного класса. Обратите внимание: `AfxRegisterWndClass` – глобальная функция, предоставляемая библиотекой MFC. При вызове функции `AfxRegisterWndClass` задаются следующие параметры.

- *Стили окна.* Первый параметр задает стили `CS_HREDRAW` и `CS_VREDRAW`, позволяющие перерисовать окно при изменении его размеров по горизонтали и вертикали.
- *Наличие и форма указателя.* Второй параметр задает форму указателя, автоматически отображаемого в окне `Windows`. Этому параметру присваивается значение 0, поэтому `Windows` не пытается отобразить указатель с помощью функции `OnMouseMove`. Таким образом нежелательное мерцание устраняется, а отображение курсора выполняется только обработчиком сообщений. Программа `ScratchBook` отображает указатель, присваивая значение *no* указателю в классе окна, а затем отображая желаемый указатель из обработчика сообщения `OnMouseMove`. Альтернативным методом является задание нужного указателя путем передачи его дескриптора во втором параметре функции `AfxRegisterWndClass`. Однако этот метод не позволяет легко изменять форму указателя во время выполнения программы. В более поздних версиях программы `ScratchBook` форма указателя изменяется каждый раз при выборе нового инструмента.
- *Цвет заливки.* Третий параметр задает стандартную белую кисть, используемую для заливки фона окна представления (кисти рассмотрены в гл. 19). В результате цвет фона окна всегда будет белым, а черные линии – видимыми, независимо от выбранного в панели управления цвета “Window”.
- *Значок окна.* Последний параметр определяет значок окна. Так как окно представления его не отображает (значок программы задается для главного окна), параметру присваивается значение 0.

Код и ресурсы начальной версии программы `ScratchBook` готовы. Теперь можно ее построить и выполнить.

## Тексты программы *ScratchBook*

Тексты составляющих элементов программы `ScratchBook` приведены в листингах с 10.1 по 10.8. Эти листинги включают программу, сгенерированную `Application Wizard`, и уже описанные дополнения и изменения.

---

### Листинг 10.1

```
// ScratchBook.h : главный заголовочный файл приложения ScratchBook

#pragma once

#ifdef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"          // основные символы

// CScratchBookApp:
// Смотрите реализацию этого класса в файле ScratchBook.cpp
//

class CScratchBookApp : public CWinApp
```

```

{
public:
    CScratchBookApp();

    // Переопределения
public:
    virtual BOOL InitInstance();

    // Реализация
    afx_msg void OnAppAbout();
    DECLARE_MESSAGE_MAP()
};

```

---

## Листинг 10.2

```

// ScratchBook.cpp : Определяет поведение классов приложения
//

#include "stdafx.h"
#include "ScratchBook.h"
#include "MainFrm.h"

#include "ScratchBookDoc.h"
#include "ScratchBookView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// CScratchBookApp

BEGIN_MESSAGE_MAP(CScratchBookApp, CWinApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
    // Стандартные команды работы с файлами документов
    ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
END_MESSAGE_MAP()

// Конструктор CScratchBookApp

CScratchBookApp::CScratchBookApp()
{
    // TODO: добавьте сюда код конструктора
    // Поместите весь существенный код инициализации
    // в функцию InitInstance
}

// Единственный объект класса CScratchBookApp

CScratchBookApp theApp;

// Инициализация CScratchBookApp

BOOL CScratchBookApp::InitInstance()

```

```

{
    CWinApp::InitInstance();

    // Стандартная инициализация
    // Если вы не используете какие-то из предоставленных
    // возможностей и хотите уменьшить размер конечного
    // исполняемого модуля, удалите из последующего кода
    // отдельные команды инициализации элементов, которые
    // вам не нужны
    // Измените ключ, под которым ваши установки
    // хранятся в реестре
    // TODO: Измените строку параметра на что-нибудь подходящее,
    // например, на имя вашей компании или организации
    SetRegistryKey(_T("Local AppWizard-Generated Applications"));
    LoadStdProfileSettings(4); // Загрузка стандартных установок
                                // из INI-файла (включая MRU)

    // Регистрация шаблона документа приложения. Шаблоны
    // документов служат связью между документами, окнами
    // документов и окнами приложений
    CSingleDocTemplate* pDocTemplate;
    pDocTemplate = new CSingleDocTemplate(
        IDR_MAINFRAME,
        RUNTIME_CLASS(CScratchBookDoc),
        RUNTIME_CLASS(CMainFrame), // главное окно
                                // SDI-приложения
        RUNTIME_CLASS(CScratchBookView));
    AddDocTemplate(pDocTemplate);
    // Просмотр командной строки для обнаружения стандартных
    // команд оболочки, DDE, открытия файлов
    CCommandLineInfo cmdInfo;
    ParseCommandLine(cmdInfo);
    // Выполнение команд, указанных в командной строке.
    // Вернет FALSE, если приложение было запущено с
    // /RegServer, /Register, /Unregserver или /Unregister.
    if (!ProcessShellCommand(cmdInfo))
        return FALSE;
    // Показ и обновление единственного
    // проинициализированного окна
    m_pMainWnd->ShowWindow(SW_SHOW);
    m_pMainWnd->UpdateWindow();

    return TRUE;
}

// CAboutDlg диалог, используемый в App About

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

    // Данные для диалога
    enum { IDD = IDD_ABOUTBOX };

protected:
    virtual void DoDataExchange(CDataExchange* pDX);

```

```

        // Поддержка DDX/DDV

// Реализация
protected:
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
END_MESSAGE_MAP()

// Команда приложения для вывода диалога
void CScratchBookApp::OnAppAbout()
{
    CAboutDlg aboutDlg;
    aboutDlg.DoModal();
}

// Обработчики сообщений класса CScratchBookApp

```

---

### Листинг 10.3

```

// ScratchBookDoc.h : интерфейс класса CScratchBookDoc
//

#pragma once

class CScratchBookDoc : public CDocument
{
protected: // используется только для сериализации
    CScratchBookDoc();
    DECLARE_DYNCREATE(CScratchBookDoc)

// Атрибуты
public:

// Операции
public:

// Переопределения
    public:
        virtual BOOL OnNewDocument();
        virtual void Serialize(CArchive& ar);

// Реализация
public:
    virtual ~CScratchBookDoc();
#ifdef _DEBUG

```

```

        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Сгенерированные функции карты сообщений
protected:
    DECLARE_MESSAGE_MAP()
};

```

---

## Листинг 10.4

```

// ScratchBookDoc.cpp : реализация класса CScratchBookDoc
//

#include "stdafx.h"
#include "ScratchBook.h"

#include "ScratchBookDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// CScratchBookDoc

IMPLEMENT_DYNCREATE(CScratchBookDoc, CDocument)

BEGIN_MESSAGE_MAP(CScratchBookDoc, CDocument)
END_MESSAGE_MAP()

// Конструктор/деструктор класса CScratchBookDoc

CScratchBookDoc::CScratchBookDoc()
{
    // TODO: добавьте сюда код одноразового конструктора
}

CScratchBookDoc::~CScratchBookDoc()
{
}

BOOL CScratchBookDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;

    // TODO: добавьте сюда код повторной инициализации
    // (SDI-документы будут использовать этот документ
    // многократно)

    return TRUE;
}

```



```

// Сериализация CScratchBookDoc

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

// Диагностика класса CScratchBookDoc

#ifdef _DEBUG
void CScratchBookDoc::AssertValid() const
{
    CDocument::AssertValid();
}

void CScratchBookDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}
#endif //_DEBUG

// Команды класса CScratchBookDoc

```

---

### Листинг 10.5

```

// ScratchBookView.h : интерфейс класса CScratchBookView
//

#pragma once

class CScratchBookView : public CView
{
protected:
    CString m_ClassName;
    int m_Dragging;
    HCURSOR m_HCross;
    CPoint m_PointOld;
    CPoint m_PointOrigin;

protected: // используется только для сериализации
    CScratchBookView();
    DECLARE_DYNCREATE(CScratchBookView)

// Атрибуты
public:
    CScratchBookDoc* GetDocument() const;

// Операции
public:

```

```

// Переопределения
public:
    virtual void OnDraw(CDC* pDC);
    // переопределена для прорисовки этого вида
virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
protected:

// Реализация
public:
    virtual ~CScratchBookView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Сгенерированные функции карты сообщений
protected:
    DECLARE_MESSAGE_MAP()
public:
    afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
    afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
    afx_msg void OnMouseMove(UINT nFlags, CPoint point);
};

#ifdef _DEBUG // отладочная версия в файле ScratchBookView.cpp
inline CScratchBookDoc* CScratchBookView::GetDocument() const
    { return (CScratchBookDoc*)m_pDocument; }
#endif

```

---

## Листинг 10.6

```

// ScratchBookView.cpp : реализация класса CScratchBookView
//

#include "stdafx.h"
#include "ScratchBook.h"

#include "ScratchBookDoc.h"
#include "ScratchBookView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// CScratchBookView

IMPLEMENT_DYNCREATE(CScratchBookView, CView)

BEGIN_MESSAGE_MAP(CScratchBookView, CView)
    ON_WM_LBUTTONDOWN()
    ON_WM_LBUTTONUP()

```

```

        ON_WM_MOUSEMOVE()
    END_MESSAGE_MAP()

    // Конструктор/деструктор класса CScratchBookView

    CScratchBookView::CScratchBookView()
    {
        // TODO: добавьте сюда собственный код конструктора

        m_Dragging = 0;
        m_HCross = AfxGetApp()->LoadStandardCursor (IDC_CROSS);
    }

    CScratchBookView::~CScratchBookView()
    {
    }

    BOOL CScratchBookView::PreCreateWindow(CREATESTRUCT& cs)
    {
        // TODO: Модифицируйте класс или стили окна,
        // добавляя и изменяя поля структуры cs

        m_ClassName = AfxRegisterWndClass
            (CS_HREDRAW | CS_VREDRAW, // стили окна
            0, // без указателя
            (HBRUSH)::GetStockObject (WHITE_BRUSH), // задать белый фон
            0); // без значка
        cs.lpszClass = m_ClassName;

        return CView::PreCreateWindow(cs);
    }

    // Прорисовка CScratchBookView

    void CScratchBookView::OnDraw(CDC* pDC)
    {
        CScratchBookDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);

        // TODO: вставьте сюда код прорисовки собственных данных
    }

    // Диагностика класса CScratchBookView

#ifdef _DEBUG
    void CScratchBookView::AssertValid() const
    {
        CView::AssertValid();
    }

    void CScratchBookView::Dump(CDumpContext& dc) const
    {
        CView::Dump(dc);
    }
#endif

```

```

CScratchBookDoc* CScratchBookView::GetDocument() const
// не отладочная версия встроена
{
    ASSERT(m_pDocument->
           IsKindOf(RUNTIME_CLASS(CScratchBookDoc)));
    return (CScratchBookDoc*)m_pDocument;
}
#endif // _DEBUG

// Обработчики сообщений класса CScratchBookView

void CScratchBookView::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: Вставьте сюда собственный код обработчика
    // или вызов стандартного

    m_PointOrigin = point;
    m_PointOld = point;
    SetCapture ();
    m_Dragging = 1;
    RECT Rect;
    GetClientRect (&Rect);
    ClientToScreen (&Rect);
    ::ClipCursor (&Rect);

    CView::OnLButtonDown(nFlags, point);
}

void CScratchBookView::OnLButtonUp(UINT nFlags, CPoint point)
{
    // TODO: Вставьте сюда собственный код обработчика
    // или вызов стандартного
    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);
    }
    CView::OnLButtonUp(nFlags, point);
}

void CScratchBookView::OnMouseMove(UINT nFlags, CPoint point)
{
    // TODO: Вставьте сюда собственный код обработчика
    // или вызов стандартного

    ::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);
    }
}

```

---

### Листинг 10.7

```

// MainFrm.h : интерфейс класса CMainFrame
//

#pragma once
class CMainFrame : public CFrameWnd
{
protected: // используется только для сериализации
    CMainFrame();
    DECLARE_DYNCREATE(CMainFrame)

// Атрибуты
public:

// Операции
public:

// Переопределения
public:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);

// Реализация
public:
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif
protected: // встроенные элементы управления
    CStatusBar m_wndStatusBar;
    CToolBar m_wndToolBar;

// Сгенерированные функции карты сообщений
protected:
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    DECLARE_MESSAGE_MAP()
};

```

---

## Листинг 10.8

```
// MainFrm.cpp : реализация класса CMainFrame
//

#include "stdafx.h"
#include "ScratchBook.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
    ON_WM_CREATE()
END_MESSAGE_MAP()

static UINT indicators[] =
{
    ID_SEPARATOR,           // индикатор строки состояния
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
};

// Конструктор/деструктор класса CMainFrame

CMainFrame::CMainFrame()
{
    // TODO: добавьте сюда код инициализации элементов
}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD | WS_VISIBLE
| CBRS_TOP | CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC)
|| !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
    {
        TRACE0("Failed to create toolbar\n");
        return -1;  // не удалось создать
                    // панель инструментов
    }
}
```

```

        if (!m_wndStatusBar.Create(this) ||
            !m_wndStatusBar.SetIndicators(indicators,
                sizeof(indicators)/sizeof(UINT)))
        {
            TRACE0("Failed to create status bar\n");
            return -1;        // не удалось создать строку состояния
        }
        // TODO: Удалите три следующие строки, если не хотите,
        // чтобы панель инструментов была паркующей
        m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
        EnableDocking(CBRS_ALIGN_ANY);
        DockControlBar(&m_wndToolBar);

        return 0;
    }

    BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
    {
        if( !CFrameWnd::PreCreateWindow(cs) )
            return FALSE;
        // TODO: Модифицируйте стили или классы окна здесь,
        // добавляя или изменяя поля структуры cs

        return TRUE;
    }

    // Диагностика класса CMainFrame

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
    CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
    CFrameWnd::Dump(dc);
}

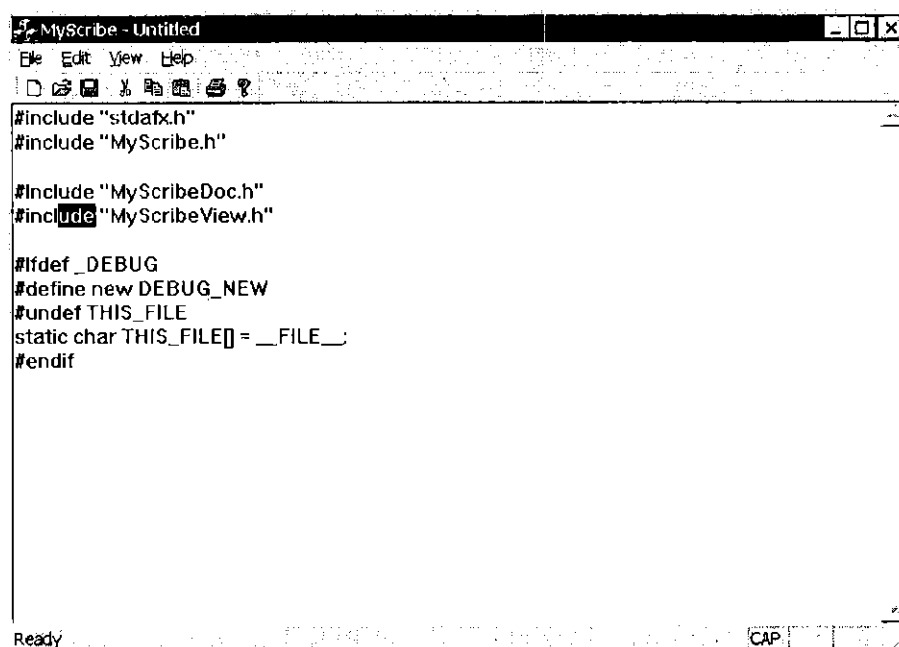
#endif // _DEBUG

// Обработчики сообщений класса CMainFrame

```

## **Простой текстовый редактор**

Простой текстовый редактор можно создать, порождая класс представления от MFC-класса `CEditView`, а не от `CView`, что позволяет вводить и редактировать *текст* внутри окна представления. Самостоятельно писать код, реализующий эти функции, не придется. В этом параграфе будет описано создание с помощью мастера Application Wizard программы MyScribe, окно которой приведено на следующей иллюстрации. Класс представления, порожденный от `CEditView`, создает экземпляр текстового редактора внутри окна представления. Программа позволяет вводить и редактировать текст в окне представления и отображает полосы прокрутки.



1. Используя редактор меню Visual Studio.NET, мы добавим пункты меню, позволяющие выполнять отдельные команды редактора.
2. Затем с помощью редактора ресурсов добавляются сочетания клавиш и значок программы.
3. Далее формируется меню программы, которое содержит команды печати текста, отмены последних действий во время редактирования, вырезания, копирования, вставки, выборки текста, поиска и замены. Версии программы MyScribe, рассмотренные в последующих главах, содержат в меню команды сохранения и загрузки текстовых файлов с диска и возможности одновременной обработки нескольких документов. При щелчке правой кнопкой мыши, когда указатель мыши находится внутри окна представления, программа отображает всплывающее (контекстное) меню, содержащее большую часть команд меню Edit.

Помимо рассмотренных в этой книге классов, библиотека MFC содержит следующие специальные классы представлений: CCtrlView, CDaoRecordView, CFormView, CHtmlView, CHtmlEditView, CListView, COleDBRecordView, CRecordView, CRichEditView, CScrollView и CTreeView. Все они являются прямыми или косвенными потомками класса CView. Полную информацию о каждом из этих классов можно найти справочной системе. Некоторые классы представлений специального назначения, реализующие соответствующие компоненты, описаны в следующих главах:

- класс CScrollView, формирующий прокручиваемое окно представления, рассматривается в гл. 13;
- класс CFormView, создающий окно для отображения элементов управления описан в гл. 16.

## ***MyScribe – проектирование программы***

Чтобы сгенерировать исходные файлы программы MyScribe, воспользуйтесь мастером Application Wizard подобно тому, как это делалось в предыдущих примерах.

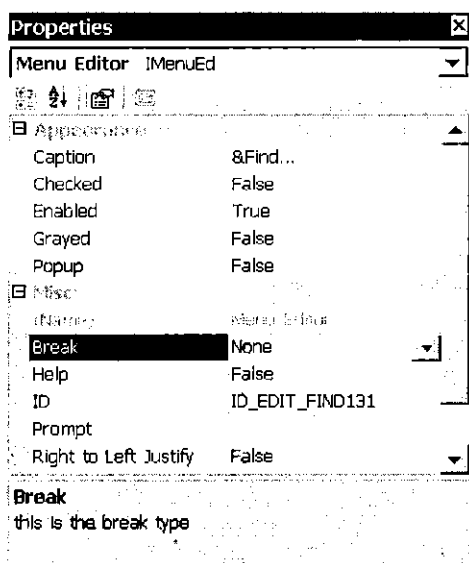
1. В диалоговом окне New Project введите имя MyScribe в поле Name, а в поле Location – путь к каталогу проекта.



2. На вкладках мастера Application Wizard выберите установки, которые выбирались при создании двух предыдущих программ.
3. На вкладке мастера Generated Classes вместо задания всех значений по умолчанию необходимо выбрать имя класса CMyScribeView из списка, находящегося в верхней части диалогового окна, и в списке базовых классов выбрать CEditView.
4. После создания исходного текста откройте вкладку Resource View и просмотрите ресурсы программы.

Далее следует изменить меню программы путем удаления неиспользуемых команд и добавления новых. Для этого выполните двойной щелчок на идентификаторе IDR\_MAINFRAME в разделе Menu. В открывшемся окне редактора меню необходимо выполнить следующие операции:

1. Удалите все пункты в меню File, кроме команды Exit, разделительного элемента над ней и команд печати.
2. Чтобы добавить новый пункт в меню Edit, выполните двойной щелчок на пустом прямоугольнике в нижней части меню Edit. Откроется диалоговое окно Properties. Введите идентификатор ID\_EDIT\_SELECT\_ALL в поле ID. Кроме того, введите строку Select &All в поле Caption. В меню Edit появится новая команда.
  - Символ & в поле Caption подчеркивает следующий за ним символ при отображении команды в меню. Когда меню открыто, для выбора команды пользователю достаточно ввести только один подчеркнутый в имени команды символ.



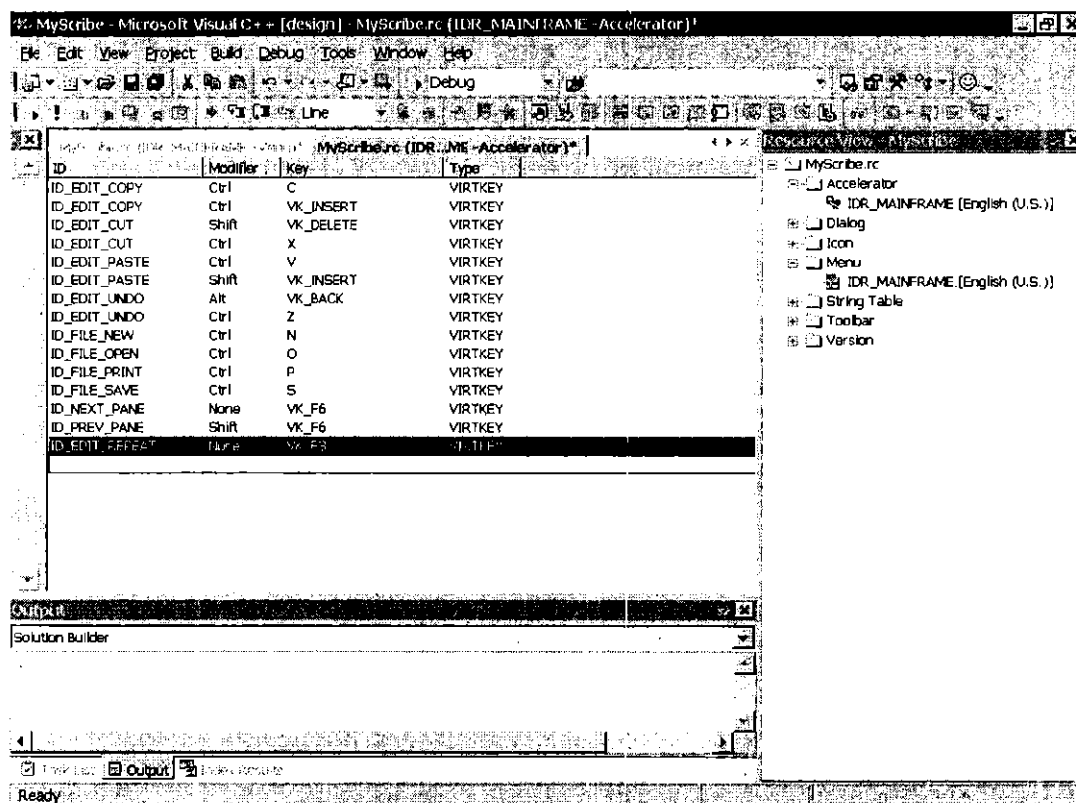
3. Перетащите новую команду (Select All) из нижней части меню Edit в подходящую позицию с помощью мыши.
4. Добавьте четыре новых команды в нижнюю часть меню Edit (под командой Paste), используя знакомые вам по п.2 приемы. Ниже перечислены надписи и идентификаторы для каждой из этих команд. После добавления команды Select All следует добавить разделительный элемент. Чтобы вставить разделитель, выберите команду Separator в контекстном меню, вызываемом правым щелчком на пустой области в меню.
  - Для надписи Select &All используется идентификатор ID\_EDIT\_SELECT\_ALL.

- Для надписи &Find... используется идентификатор ID\_EDIT\_FIND.
  - Для надписи Find &Next\<F3 используется идентификатор ID\_EDIT\_REPEAT. Символы \< задают вывод в строке символа табуляции, в результате чего следующая за ним часть строки выравнивается по правому краю меню (это удобно, если в элементе меню кроме названия команды необходимо отобразить и сочетание клавиш).
  - Для надписи &Replace... используется идентификатор ID\_EDIT\_REPLACE.
5. Закройте окно редактора меню, щелкнув на Close или выполнив двойной щелчок на его системном меню.

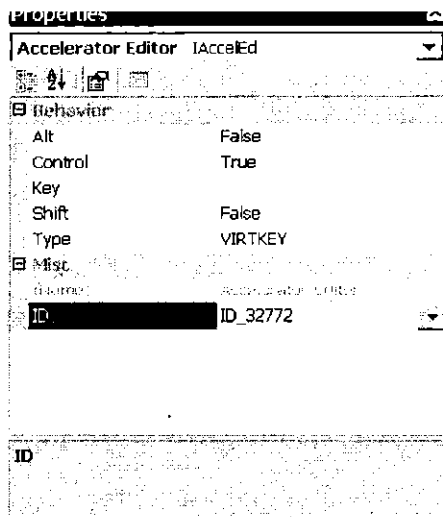
Добавляя команды меню, следует использовать стандартизованные (для библиотеки MFC) идентификаторы команд, приведенные выше, чтобы карта сообщений, определенная в классе CEditView, направляла каждое командное сообщение соответствующему обработчику. Для каждой команды, добавленной в меню Edit, в нижней части диалогового окна Properties отображается подсказка. Если программа во время работы отображает строку состояния, то подсказка для каждой выделенной команды меню выводится в этой строке.

В надписи к команде Find Next назначается клавиша для быстрого выполнения команды. Такие клавиши или их сочетания называются *горячими клавишами* и определяются редактором горячих клавиш Visual Studio следующим образом:

1. Дважды щелкните в Resource View в разделе Accelerator на идентификаторе IDR\_MAINFRAME.
- Откроется окно редактора горячих клавиш.



2. Дважды щелкните на пустом поле в нижней части списка для добавления новых клавиш. Откроется диалоговое окно Properties.



3. В поле ID введите заданный для команды Find Next меню Edit идентификатор ID\_EDIT\_REPEAT.
4. В поле Key укажите VK\_F3 (это соответствует клавише F3).
5. Закройте окно редактора горячих клавиш.

Разработка собственного значка программы MyScribe выполняется как обычно в окне графического редактора. Чтобы отредактировать значок IDR\_MAINFRAME, следуйте инструкциям для программы ScratchBook в параграфе “Ресурсы программы ScratchBook”.

Завершив редактирование ресурсов сохраните все изменения, выбрав в меню File команду Save All или щелкнув на панели инструментов Standard на кнопке Save All. Теперь можно построить проект MyScribe, запустить программу и поэкспериментировать с ней. Все команды меню полностью реализованы. Большинство этих команд обрабатываются средствами класса CEditView. Библиотека MFC отображает контекстное меню, появляющееся при щелчке правой кнопкой мыши внутри окна представления. Если ввести текст в окно и затем завершить работу программы MyScribe, она выводит стандартное окно сохранения текста в файле на диске. Это единственный способ сохранения файла в первой версии программы MyScribe. Полный набор функций ввода-вывода будет добавлен в версию, описанную в гл. 12.

## Тексты программы MyScribe

Исходные тексты программы MyScribe приведены в листингах (10.9—10.16). Они должны соответствовать текстам, полученным в результате выполнения описанных выше процедур генерации и настройки.

### Листинг 10.9

```
// MyScribe.h : главный заголовочный файл приложения MyScribe
//
#pragma once

#ifdef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif
```

```

#include "resource.h"          // основные символы

// CMyScribeApp:
// Смотри реализацию этого класса в файле MyScribe.cpp
//

class CMyScribeApp : public CWinApp
{
public:
    CMyScribeApp();

    // Переопределения
public:
    virtual BOOL InitInstance();

    // Реализация
    afx_msg void OnAppAbout();
    DECLARE_MESSAGE_MAP()
};

```

---

### Листинг 10.10

```

// MyScribe.cpp : Определяет поведение классов приложения.
//

#include "stdafx.h"
#include "MyScribe.h"
#include "MainFrm.h"

#include "MyScribeDoc.h"
#include "MyScribeView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// CMyScribeApp

BEGIN_MESSAGE_MAP(CMyScribeApp, CWinApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
    // Стандартные команды работы с файлами документов
    ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
    // Стандартные команды печати
    ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()

// Конструктор класса CMyScribeApp

CMyScribeApp::CMyScribeApp()
{
    // TODO: добавьте сюда код конструктора
    // Поместите весь существенный код инициализации
    // в функцию InitInstance
}

```

```

// Единственный объект класса CMyScribeApp
CMyScribeApp theApp;

// Инициализация CMyScribeApp
BOOL CMyScribeApp::InitInstance()
{
    CWinApp::InitInstance();

    // Стандартная инициализация
    // Если вы не используете какие-то из предоставленных
    // возможностей и хотите уменьшить размер конечного
    // исполняемого модуля, удалите из последующего кода
    // отдельные команды инициализации элементов, которые
    // вам не нужны
    // Измените ключ, под которым ваши установки
    // хранятся в реестре
    // TODO: Измените параметр на что-нибудь подходящее,
    // например, на имя вашей компании или организации
    SetRegistryKey(_T("Local AppWizard-Generated Applications"));
    LoadStdProfileSettings(4); // Загрузка стандартных установок
                                // из INI-файла (включая MRU)
    // Регистрация шаблона документа приложения. Шаблоны
    // документов служат связью между документами, окнами
    // документов и окнами приложений
    CSingleDocTemplate* pDocTemplate;
    pDocTemplate = new CSingleDocTemplate(
        IDR_MAINFRAME,
        RUNTIME_CLASS(CMyScribeDoc),
        RUNTIME_CLASS(CMainFrame),           // главное окно
                                                // SDI-приложения
        RUNTIME_CLASS(CMyScribeView));
    AddDocTemplate(pDocTemplate);
    // Просмотр командной строки для обнаружения стандартных
    // команд оболочки, DDE, открытия файлов
    CCommandLineInfo cmdInfo;
    ParseCommandLine(cmdInfo);
    // Выполнение команд, указанных в командной строке.
    // Вернет FALSE, если приложение было запущено с
    // /RegServer, /Register, /Unregserver или /Unregister.
    if (!ProcessShellCommand(cmdInfo))
        return FALSE;
    // Показ и обновление единственного
    // проинициализированного окна
    m_pMainWnd->ShowWindow(SW_SHOW);
    m_pMainWnd->UpdateWindow();

    return TRUE;
}

// CAboutDlg диалог, используемый в App About
class CAboutDlg : public CDialog
{
public:
    CAboutDlg();

```

```

// Данные для диалога
enum { IDD = IDD_ABOUTBOX };

protected:
    virtual void DoDataExchange(CDataExchange* pDX);
    // Поддержка DDX/DDV

// Реализация
protected:
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAaboutDlg::IDD)
{
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CAaboutDlg, CDialog)
END_MESSAGE_MAP()

// Команда приложения на вывод диалога
void CMyScribeApp::OnAppAbout()
{
    CAboutDlg aboutDlg;
    aboutDlg.DoModal();
}

// Обработчики сообщений класса CMyScribeApp

```

---

### Листинг 10.11

```

// MyScribeDoc.h : интерфейс класса CMyScribeDoc
//

#pragma once

class CMyScribeDoc : public CDocument
{
protected: // используется только для сериализации
    CMyScribeDoc();
    DECLARE_DYNCREATE(CMyScribeDoc)

// Атрибуты
public:

// Операции
public:

// Переопределения
public:
    virtual BOOL OnNewDocument();
    virtual void Serialize(CArchive& ar);

```

```

// Реализация
public:
    virtual ~CMyScribeDoc();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Сгенерированные функции карты сообщений
protected:
    DECLARE_MESSAGE_MAP()
};

```

---

### Листинг 10.12

```

// MyScribeDoc.cpp : реализация класса CMyScribeDoc
//

#include "stdafx.h"
#include "MyScribe.h"

#include "MyScribeDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// CMyScribeDoc

IMPLEMENT_DYNCREATE(CMyScribeDoc, CDocument)

BEGIN_MESSAGE_MAP(CMyScribeDoc, CDocument)
END_MESSAGE_MAP()

// Конструктор/деструктор класса CMyScribeDoc

CMyScribeDoc::CMyScribeDoc()
{
    // TODO: добавьте сюда одноразовый код конструктора
}

CMyScribeDoc::~CMyScribeDoc()
{
}

BOOL CMyScribeDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;

    ((CEditView*)m_viewList.GetHead())->SetWindowText(NULL);
}

```

```

        // TODO: добавьте сюда код повторной инициализации
        // (SDI-документы будут многократно использовать этот
        // документ)

        return TRUE;
    }

    // Сериализация CMyScribeDoc

void CMyScribeDoc::Serialize(CArchive& ar)
{
    // CEditView содержит элемент управления,
    // выполняющий сериализацию
    ((CEditView*)m_viewList.GetHead())->SerializeRaw(ar);
}

// Диагностика класса CMyScribeDoc

#ifdef _DEBUG
void CMyScribeDoc::AssertValid() const
{
    CDocument::AssertValid();
}

void CMyScribeDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}
#endif //_DEBUG

// Команды класса CMyScribeDoc

```

---

### Листинг 10.13

```

// MyScribeView.h : интерфейс класса CMyScribeView
//

#pragma once

class CMyScribeView : public CEditView
{
protected: // используется только для сериализации
    CMyScribeView();
    DECLARE_DYNCREATE(CMyScribeView)

// Атрибуты
public:
    CMyScribeDoc* GetDocument() const;

// Операции
public:

// Переопределения
public:
virtual BOOL PreCreateWindow(CREATESTRUCT& cs);

```



```

protected:
    virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
    virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
    virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);

// Реализация
public:
    virtual ~CMyScribeView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Сгенерированные функции карты сообщений
protected:
    DECLARE_MESSAGE_MAP()
};

#ifdef _DEBUG // отладочная версия в файле MyScribeView.cpp
inline CMYscribeDoc* CMYscribeView::GetDocument() const
    { return (CMYscribeDoc*)m_pDocument; }
#endif

```

---

#### Листинг 10.14

```

// MyScribeView.cpp : реализация класса CMYscribeView
//

#include "stdafx.h"
#include "MyScribe.h"

#include "MyScribeDoc.h"
#include "MyScribeView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// CMYscribeView

IMPLEMENT_DYNCREATE(CMYscribeView, CEditView)

BEGIN_MESSAGE_MAP(CMYscribeView, CEditView)
    // Стандартные команды печати
    ON_COMMAND(ID_FILE_PRINT, CEditView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_DIRECT, CEditView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_PREVIEW,
        CEditView::OnFilePrintPreview)
END_MESSAGE_MAP()

// Конструктор/деструктор класса CMYscribeView

```

```

CMyScribeView::CMyScribeView()
{
    // TODO: добавьте сюда код конструктора
}

CMyScribeView::~CMyScribeView()
{
}

BOOL CMyScribeView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Измените класс или стиль окна здесь,
    // добавляя или изменяя поля структуры cs

    BOOL bPreCreated = CEditView::PreCreateWindow(cs);
    cs.style &= ~(ES_AUTOHSCROLL|WS_HSCROLL);
    // Разрешить перенос слов

    return bPreCreated;
}

// Печать в CMyScribeView

BOOL CMyScribeView::OnPreparePrinting(CPrintInfo* pInfo)
{
    // Подготовка печати в CEditView по умолчанию
    return CEditView::OnPreparePrinting(pInfo);
}

void CMyScribeView::OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo)
{
    // Начало печати в CEditView по умолчанию
    CEditView::OnBeginPrinting(pDC, pInfo);
}

void CMyScribeView::OnEndPrinting(CDC* pDC, CPrintInfo* pInfo)
{
    // Окончание печати в CEditView по умолчанию
    CEditView::OnEndPrinting(pDC, pInfo);
}

// Диагностика класса CMyScribeView

#ifdef _DEBUG
void CMyScribeView::AssertValid() const
{
    CEditView::AssertValid();
}

void CMyScribeView::Dump(CDumpContext& dc) const
{
    CEditView::Dump(dc);
}

```

```

CMyScribeDoc* CMyScribeView::GetDocument() const
// рабочая (не отладочная) версия встроена
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CMyScribeDoc)));
    return (CMyScribeDoc*)m_pDocument;
}
#endif // _DEBUG

// Обработчики сообщений класса CMyScribeView

```

---

### Листинг 10.15

```

// MainFrm.h : интерфейс класса CMainFrame
//

#pragma once
class CMainFrame : public CFrameWnd
{
protected: // используется только для сериализации
    CMainFrame();
    DECLARE_DYNCREATE(CMainFrame)

// Атрибуты
public:

// Операции
public:

// Переопределения
public:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);

// Реализация
public:
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected: // Встроенные элементы управления
    CStatusBar m_wndStatusBar;
    CToolBar m_wndToolBar;

// Сгенерированные функции карты сообщений
protected:
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    DECLARE_MESSAGE_MAP()
};

```

---

### Листинг 10.16

```

// MainFrm.cpp : реализация класса CMainFrame

#include "stdafx.h"
#include "MyScribe.h"

```

```

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
    ON_WM_CREATE()
END_MESSAGE_MAP()

static UINT indicators[] =
{
    ID_SEPARATOR,           // индикатор строки состояния
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
};

// Конструктор/деструктор класса CMainFrame

CMainFrame::CMainFrame()
{
    // TODO: добавьте сюда код инициализации элементов
}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD | WS_VISIBLE
| CBRS_TOP | CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC)
|| !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
    {
        TRACE0("Failed to create toolbar\n");
        return -1;      // не удалось создать
                        // панель инструментов
    }

    if (!m_wndStatusBar.Create(this) ||
        !m_wndStatusBar.SetIndicators(indicators,
            sizeof(indicators)/sizeof(UINT)))
    {
        TRACE0("Failed to create status bar\n");
        return -1;      // не удалось создать строку состояния
    }
}

```

```

        // TODO: Удалите три следующие строки, если не хотите, чтобы
        // панель инструментов была паркующей
        m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
        EnableDocking(CBRS_ALIGN_ANY);
        DockControlBar(&m_wndToolBar);

        return 0;
    }

    BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
    {
        if( !CFrameWnd::PreCreateWindow(cs) )
            return FALSE;
        // TODO: Измените стиль или класс окна здесь,
        // изменяя и добавляя поля структуры cs

        return TRUE;
    }

    // Диагностика класса CMainFrame

#ifdef _DEBUG
    void CMainFrame::AssertValid() const
    {
        CFrameWnd::AssertValid();
    }

    void CMainFrame::Dump(CDumpContext& dc) const
    {
        CFrameWnd::Dump(dc);
    }

#endif //_DEBUG

    // Обработчики сообщений класса CMainFrame

```

## Резюме

Мы рассмотрели технику реализации класса представления в MFC-программах. В примере программы графического редактора ScratchBook класс представления порожден от класса общего назначения CView. Для обработки и отображения информации, выводимой в окне представления, добавлен оригинальный текст. В программе текстового редактора MyScribe класс представления порожден от специального класса CEditView, реализующего полный набор функций текстового редактирования, без добавления фрагментов программы в класс представления.

- *Сообщения о событиях* передаются в MFC-программе объекту класса. Можно задать функцион-член класса, получающую управление при каждой передаче определенного сообщения.
- *Обработчик сообщения.* Каждый тип сообщения имеет идентификатор. Например, при нажатии пользователем левой кнопки мыши, когда указатель мыши находится внутри окна представления, передается идентификатор WM\_LBUTTONDOWN. Если на это событие нужно ответить, можно использовать вкладку Properties, чтобы определить обработчик сообщений WM\_LBUTTONDOWN как функцию класса представления, а затем добавить в программу оригинальный код. Если для определенного типа сообщения обработчик не задан, то оно обрабатывается стандартным образом.

- *Окно представления* настраивается добавлением кода в виртуальную функцию `PreCreateWindow` класса представления, которая переопределяет функцию, заданную в классе `CView`, и вызывается непосредственно перед созданием окна. В полях структуры `CREATESTRUCT`, передаваемой в функцию `PreCreateWindow` можно задать любые свойства этого окна.
- *Ресурсы Windows*. Для генерируемой программы мастер `Application Wizard` определяет набор стандартных ресурсов (меню, сочетания клавиш, значки и панели). Чтобы внести в них изменения или добавить новые ресурсы, используйте редактор ресурсов.

# Глава 11

## Данные в документе

---

- Сохранение данных и перерисовка окна
- Модификация меню
- Удаление данных
- Текст программы ScratchBook

В MFC предусмотрен специальный класс (*класс документа*), который отвечает:

- за хранение данных;
- за их загрузку из файлов на диске;
- за обработку команд меню, непосредственно воздействующих на данные документа.

Этот класс содержит функции, позволяющие другим классам (в частности, классу представления) получать или изменять данные таким образом, чтобы они были доступны для просмотра и редактирования. В данной главе программа ScratchBook, созданная в предыдущей главе, будет дополнена основными возможностями класса документа:

- Сначала в нее добавятся переменные, сохраняющие координаты созданной линии.
- Затем класс представления изменится таким образом, чтобы не только рисовать линии, но и сохранять каждую из них внутри объекта класса документа.
- После этого реализуется функция OnDraw класса представления, восстанавливающая линии при перерисовке окна с использованием данных, сохраненных объектом документа. В предыдущей версии ScratchBook координаты в объекте документа не сохранялись, поэтому при перерисовке окна линии удалялись.
- Наконец, мы добавим в меню Edit команды Undo и Remove All. Обе эти команды реализуются классом документа, так как они непосредственно воздействуют на сохраняемые данные. Команда Undo удаляет последнюю линию, а Remove All – все линии на рисунке. Программный код ввода-вывода файлов, позволяющий сохранять рисунки в файлах на диске и затем их загружать, будет добавлен к классу документа в гл. 12.

Для получения набора исходных файлов в этой главе мастер Application Wizard не используется. Вместо него применяются созданные в предыдущей главе файлы программы ScratchBook, к которым добавляются новые компоненты. Если вы хотите сохранить предыдущую версию программы ScratchBook (см. гл. 10), скопируйте все файлы программы в новый каталог проекта и внесите описанные в этой главе изменения в копии файлов.

### Сохранение данных и перерисовка окна

---

Дополним программу ScratchBook кодом сохранения графических данных. После открытия проекта необходимо определить класс для сохранения данных о каждой созданной линии. В начало файла заголовков ScratchBookDoc.h перед определением класса CScratchBookDoc добавьте выделенное полужирным шрифтом определение CLine. Переменные m\_X1 и m\_Y1 класса CLine сохраняют координаты одного конца прямой линии, m\_X2 и m\_Y2 – другого. Класс CLine содержит

также функцию Draw для рисования линии. Данный класс порождается от CObject (скоро мы увидим, почему).

```
class CLine : public CObject
{
protected:
    int m_X1, m_Y1, m_X2, m_Y2;

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);
};
```

В класс CScratchBookDoc добавим требуемые члены, вводя в начало определения класса операторы, выделенные полужирным шрифтом. Новая переменная m\_LineArray – это экземпляр шаблона MFC-класса CTypedPtrArray (см. описание шаблонов классов в гл. 7.) Класс CTypedPtrArray генерирует семейство классов, каждый из которых является производным от класса, заданного в первом параметре шаблона (CObArray или CPtrArray). Каждый из этих классов предназначен для хранения переменных класса, описанных вторым параметром шаблона. Таким образом, переменная m\_LineArray является объектом класса, сохраняющего указатели на объекты класса Cline и порожденного от класса CObArray.

```
class CScratchBookDoc : public CDocument
{
protected:
    CTypedPtrArray<CObArray, CLine*> m_LineArray;

public:
    void AddLine (int X1, int Y1, int X2, int Y2);
    CLine *GetLine (int Index);
    int GetNumLines ();

protected: // используется только для сериализации
    CScratchBookDoc();
    DECLARE_DYNCREATE(CScratchBookDoc)
```

Для сохранения групп переменных или объектов использован класс CObArray (один из классов коллекций общего назначения в библиотеке MFC). Экземпляр класса CObArray 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 CObArray программа ScratchBook использует шаблон CTypedPtrArray, спроектированный специально для хранения объектов класса CLine. Это позволяет компилятору выполнять более жесткий 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. Включите его в стандартный файл заголовков StdAfx.h (как показано



ниже), чтобы сделать доступным для любого заголовочного или исходного файла в проекте ScratchBook. Файл заголовков StdAfx.h включают в начало всех исходных файлов проекта. Кроме того, он *предварительно компилируется*, поэтому, как и другие включенные в него файлы заголовков, не компилируется заново при каждом изменении исходного файла, что сокращает общее время компиляции. Информацию о предварительной компиляции файлов заголовков можно найти в справочной системе.

```
#include <afxdtctl.h>           // MFC-поддержка для Internet
                                // Explorer 4 Common Controls
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>             // MFC-поддержка для Windows
                                // Common Controls
#endif // _AFX_NO_AFXCMN_SUPPORT
#include <afxtempl.h>
```

Для хранения указателя на каждый объект класса CLine, сохраняющий информацию о линии, в ScratchBook используется переменная m\_LineArray. Функции AddLine, GetLine и GetNumLines предоставляют доступ к информации о линиях, хранящейся в массиве m\_LineArray. Переменная m\_LineArray является защищенной, поэтому другие классы не имеют к ней прямого доступа.

Введите определение функции CLine::Draw в конце файла реализации документа ScratchBookDoc.cpp. Функция Draw вызывает две функции класса CDC – MoveTo и LineTo (см. гл. 10), чтобы создать линию по координатам, сохраненным в текущем объекте. Кроме того, добавьте определения функций AddLine, GetLine и GetNumLines класса CScratchBookDoc.

```
// Команды класса CScratchBookDoc

void CLine::Draw (CDC *PDC)
{
    PDC->MoveTo (m_X1, m_Y1);
    PDC->LineTo (m_X2, m_Y2);
}

void CScratchBookDoc::AddLine (int X1, int Y1, int X2, int Y2)
{
    CLine *PLine = new CLine (X1, Y1, X2, Y2);
    m_LineArray.Add (PLine);
}

CLine *CScratchBookDoc::GetLine (int Index)
{
    if (Index < 0 || Index > m_LineArray.GetUpperBound ())
        return 0;
    return m_LineArray.GetAt (Index);
}

int CScratchBookDoc::GetNumLines ()
{
    return m_LineArray.GetSize ();
}
```

Функция AddLine создает новый объект класса CLine и вызывает функцию Add класса CObArray, чтобы добавить указатель объекта в коллекцию указателей класса CLine, сохраненных в массиве m\_LineArray. Указатели, сохраненные в массиве m\_LineArray, индексируются. Первый добавленный указатель имеет индекс 0, второй – 1 и т.д. Функция GetLine возвращает указатель, индекс которого задан в переданном ей параметре. Для этого она сначала 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 получаемый в результате вызова функции `GetAt` класса `CTypedPtrArray` указатель класса `CLine`.

Последняя из добавленных функций – `GetNumLines` – возвращает количество указателей на класс `CLine`, сохраненных в переменной `m_LineArray`. Для этого вызывается функция `GetSize` класса `CObArray`. Функции `AddLine`, `GetLine` и `GetNumLines` вызываются функциями-членами класса представления (см. ниже).

После рисования линии (см. гл. 10), когда пользователь отпускает левую кнопку мыши, функция `OnLButtonUp` класса представления, определенная в файле `ScratchBookView.cpp`, завершает операцию перемещения указателя и окончательно отображает линию. Добавьте в эту функцию вызовы функций `GetDocument` и `AddLine`, чтобы сохранить нарисованную линию.

```
void CScratchBookView::OnLButtonUp(UINT nFlags, CPoint point)
{
    // TODO: Вставьте сюда собственный код обработчика
    // или вызов стандартного
    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);

        CScratchBookDoc* PDoc = GetDocument();
        PDoc -> AddLine (m_PointOrigin.x,
                        m_PointOrigin.y, point.x, point.y);
    }

    CView::OnLButtonUp(nFlags, point);
}
```

Обратите внимание: кроме отображения данных документа, класс представления должен вызывать функции для обновления отредактированных данных. Теперь программа постоянно хранит данные, позволяющие восстановить линию, а класс представления может использовать их при перерисовке окна.

Для перерисовки окна система удаляет его содержимое, а затем вызывает функцию `OnDraw` класса представления. В действительности система удаляет только ту часть окна представления, которую нужно перерисовать, например, закрытую другим окном. В гл. 13 вы узнаете, как повысить эффективность функции `OnDraw` за счет перерисовки только требуемых частей. В минимальную версию функции `OnDraw`, генерируемую мастером `Application Wizard`, необходимо добавить собственный код для перерисовки окна. Для этого добавьте строки, выделенные полужирным шрифтом, в файл `ScratchBookView.cpp` в функцию `CScratchBookView::OnDraw`. В добавленном коде вызывается функция `CScratchBookDoc::GetNumLines`, позволяющая определить количество линий, сохраненных объектом документа. В этом фрагменте программы для каждой линии сначала вызывается

функция `CScratchBookDoc::GetLine`, которая получает указатель на соответствующий объект класса `CLine`, а затем этот указатель используется для рисования линии с помощью `CLine::Draw`.

```
// Прорисовка CScratchBookView

void CScratchBookView::OnDraw(CDC* pDC)
{
    CScratchBookDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    // TODO: вставьте сюда код прорисовки собственных данных

    int Index = pDoc->GetNumLines();
    while (Index-->0)
        pDoc->GetLine (Index)->Draw(pDC);
}
```

## Модификация меню

В результате внесенных в программу изменений графические данные хранятся внутри объекта документа. Поэтому целесообразно реализовать в программе некоторые команды меню `Edit`, чтобы можно было эти данные изменять. Добавим команды `Remove All` для удаления всех линий и `Undo` для удаления последней нарисованной линии. Чтобы добавить команды меню `Edit` в `ScratchBook`, откройте в вкладку `Resource View` для отображения списка ресурсов программы. Затем, чтобы открыть редактор меню, выполните двойной щелчок в разделе `Menu` на идентификаторе `IDR_MAINFRAME`.

В редакторе меню щелкните правой кнопкой мыши в пустом поле на правом крае строки меню и из контекстного меню выберите команду `Properties`. В поле `Caption` открывшегося окна введите `&Edit`. После этого в строке меню появится меню `Edit`. При создании разворачивающегося меню идентификатор не вводится. Идентификаторы присваиваются только командам меню. С помощью мыши перетяните меню `Edit` влево таким образом, чтобы оно разместилось между меню `File` и `View`.

На пустом прямоугольнике внутри меню `Edit` (под заголовком) щелкните правой кнопкой мыши, чтобы снова открыть окно `Properties` для определения новых команд меню. В поле `ID` введите `ID_EDIT_UNDO`, а в `Caption` – `&Undo\&Ctrl+Z`. Для команды `Undo` не нужно определять комбинацию клавиш `Ctrl+Z`, так как мастер `Application Wizard` определил ее при первичной генерации кода исходной программы. Кроме того, мастер назначил для команды `Undo` комбинацию клавиш `Alt+Backspace`, обычную для ранних графических приложений. Теперь в меню `Edit` появится команда `Undo`. Щелкните правой кнопкой мыши на пустом поле внизу меню `Edit` (под командой `Undo`) и в окне `Properties` отметьте опцию `Separator`. Под командой `Undo` будет вставлен разделитель. Выполните двойной щелчок на пустом поле внизу меню `Edit`, затем введите в поле `ID` значение `ID_EDIT_CLEAR_ALL`, а в `Caption` – `&Remove All`. В меню добавится команда `Remove All`. Теперь меню `Edit` представлено в окончательном виде.

Закройте окно редактора меню и сохраните результаты, выбрав команду `Save All` в меню `File` или щелкнув на кнопке `Save All` панели инструментов `Standard`. При проектировании меню с помощью редактора можно создавать каскадные меню, перемещать элементы между меню и подменю и задавать пунктам некоторые начальные установки (например, можно сделать опцию выбранной по умолчанию или отметить команду меню как недоступную). Информация об использовании редактора меню для реализации всех особенностей меню содержится в справочной системе.

## Реализация команды Remove All

Для определения обработчика сообщения, получающего управление при выборе команды Remove All:

1. Откройте вкладку Resource View, вызовите редактор меню и обратите внимание на идентификаторы, присвоенные Visual Studio командам Undo и Remove All в меню Edit (они будут числовыми).
2. Перейдите на вкладку Class View. Откройте контекстное меню для класса CScratchBookDoc щелчком правой кнопкой мыши на этом классе.
3. Откройте окно Properties. В списке обработчиков сообщений выберите сообщение с идентификатором команды Remove All и добавьте собственный обработчик для пункта COMMAND в правом столбце списка (этот пункт указывает на обработчик вызова команды меню, а второй пункт – UPDATE\_COMMAND\_UI – на обработчик инициализации этой команды). Каким образом сообщения, сгенерированные объектами пользовательского интерфейса, передаются через основные классы программы, пояснялось в параграфе “Обработчики сообщений” гл. 10.
4. Добавьте в созданную функцию выделенный в листинге полужирным шрифтом код. Вызов ранее определенной функции DeleteContents приводит к удалению содержимого документа. Далее функция UpdateAllViews класса CDocument удаляет текущее содержимое окна представления. В действительности (см. гл. 13) функция UpdateAllViews выполняет и другие действия.

```
void CScratchBookDoc::On57633()
{
    // TODO: Добавьте сюда собственный код обработчика
    DeleteContents ();
    UpdateAllViews (0);
}
```

5. Определите обработчик сообщения UPDATE\_COMMAND\_UI, посылаемого при первом открытии меню, содержащего команду Remove 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 Remove All доступной, если документ содержит одну или более линий, и недоступной, если документ их не содержит. Для определения обработчика выполните описанные в п. 1...4 действия, однако в списке выберите идентификатор UPDATE\_COMMAND\_UI. Visual Studio сгенерирует соответствующую функцию.
6. Добавьте в эту функцию выделенный полужирным шрифтом код. Функции 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 Remove All, если документ содержит хотя бы одну линию. В противном случае команда блокируется (она отображается затененной серым цветом) и пользователь не может ее выбрать. Таким образом, если документ не содержит линий, эту функцию нельзя вызвать.

```
void CScratchBookDoc::OnUpdate57633(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика

    pCmdUI->Enable (m_LineArray.GetSize ());
}
```

### Примечание

Обратите внимание на имя функции обработчика, которое в явном виде не содержит имени обрабатываемой команды меню. Приведенное имя генерируется Visual Studio. Узнать, какое имя соответствует той или иной команде, можно из вкладки Resource View и редактора меню.

Класс CCmdUI предоставляет четыре функции, которые можно использовать для инициализации команд:

- **Функция Enable.** Чтобы сделать команду доступной, в функцию Enable передается значение TRUE. Значение FALSE блокирует команду.

```
virtual void Enable (BOOL bOn = TRUE);
```

- **Функция SetCheck.** В функцию SetCheck можно передать значение 1, чтобы сделать пункт меню выбранным, или значение 0, чтобы отменить выбор. Обычно команда меню, представляющая какую-либо функциональную возможность программы (опцию, флажок), отмечается, если данная возможность активизирована (флажок установлен).

```
virtual void SetCheck (int nCheck = 1);
```

- **Функция SetRadio.** Чтобы отметить пункт меню (позицию переключателя из группы взаимоисключающих позиций) с помощью специального маркера (кружка), можно передать значение TRUE функции SetRadio. Значение FALSE удаляет маркер.

```
virtual void SetRadio (BOOL bOn = TRUE);
```

- **Функция SetText.** Чтобы изменить надпись команды меню, можно вызвать функцию SetText, задав параметр lpszText (указатель на новую строку текста). Например, если предыдущее действие состояло в удалении текста, то можно вызвать функцию SetText для замены команды Undo на Undo Delete.

```
virtual void SetText (LPCTSTR lpszText);
```

Можно определить обработчики сообщений UPDATE\_COMMAND\_UI и для других объектов пользовательского интерфейса, например для кнопок панели инструментов (см. гл. 14). Для обновления этих объектов можно использовать те же четыре функции класса CCmdUI. Особенности работы каждой из этих функций зависят от типа обрабатываемого объекта.

## Реализация команды Undo

Необходимо начать с постройки функции, получающей управление, когда пользователь *выбирает* команду Undo. Процедура проектирования не отличается от приведенной выше для команды Remove All. В списке сообщений выберите идентификатор, присвоенный Visual Studio команде Undo. Visual Studio создаст функцию с соответствующим именем. Добавьте в эту функцию выделенный полужирным код. При многократном выборе команды Undo обработчик этой команды продолжает удалять линии до тех пор, пока не останется ни одной. 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 оператор delete. И, наконец, вызывается функция UpdateAllViews, которая удаляет окно представления и вызывает функцию CScratchBookView::OnDraw. После этого обработчик OnDraw перерисовывает линии, оставшиеся в документе.

```
void CScratchBookDoc::On57643()
{
    // TODO: Добавьте сюда собственный код обработчика
    int Index = m_LineArray.GetUpperBound();
    if (Index > -1)
    {
        delete m_LineArray.GetAt (Index);
        m_LineArray.RemoveAt (Index);
    }
}
```

```

    }
    UpdateAllViews (0);
}

```

Функция инициализации команды меню Undo строится в рамках процедуры, аналогичной приведенной в предыдущем параграфе для команды Remove All. Добавьте в созданную функцию выделенный полужирным код. Эта функция делает команду Undo доступной при наличии хотя бы одной линии для удаления.

```

void CScratchBookDoc::OnUpdate57643(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->Enable (m_LineArray.GetSize ());
}

```

Для команды Undo была задана комбинация клавиш Ctrl+Z. Когда пользователь нажимает Ctrl+Z, вызывается обработчик команды Undo. Если функция OnUpdate делает команду меню доступной, вызывается обработчик сообщения, так что нажатие горячих клавиш обрабатывается так же, как и выбор команды меню. Однако если функция OnUpdate блокирует команду меню, система *не* вызывает обработчик сообщения. Таким образом, если в документе нет ни одной линии, комбинация клавиш также блокируется, и функция не будет вызываться.

## Удаление данных

Если в меню File выбрать команду New, то функция OnFileNew класса CWinApp MFC вызывает виртуальную функцию CDocument::DeleteContents для удаления содержимого текущего документа перед инициализацией нового. В последующих версиях программы ScratchBook 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 общепринятым и эффективным способом настройки MFC. Чтобы сгенерировать объявление и оболочку функции DeleteContents, воспользуйтесь вкладкой Class View.

1. Выберите в меню View команду Class View.
2. На вкладке Class View щелкните правой кнопкой на классе CScratchBookDoc и в контекстном меню выберите команду Properties.
3. В окне Properties выберите элемент Events (события). В списке, появившемся на экране, выберите пункт DeleteContents, а в соседнем столбце выберите собственный обработчик для данного события (это единственный пункт в списке).
4. Добавьте в функцию DeleteContents, сгенерированную Visual Studio в файле ScratchBook-Doc.cpp, строки, выделенные полужирным шрифтом в приведенном ниже листинге. 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 (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.

```

void CScratchBookDoc::DeleteContents()
{
    // TODO: Добавьте сюда свой код или вызов базового
    // класса
    int Index = m_LineArray.GetSize ();
    while (Index--)
        delete m_LineArray.GetAt (Index);
    m_LineArray.RemoveAll ();

    CDocument::DeleteContents();
}

```

Библиотека MFC после вызова функции `DeleteContents` удаляет (косвенным образом) окно представления и вызывает функцию `OnDraw`. Однако функция `OnDraw` не отображает линии, потому что они были удалены из класса документа. В результате мы получаем удаление данных документа командой `New` и выполнение перед созданием нового рисунка очистки окна представления.

## Текст программы *ScratchBook*

---

В приведенных ниже листингах (11.1—11.8) приведены исходные тексты версии программы *ScratchBook*, описанной в этой главе. Файлы содержат код, сгенерированный мастером `Application Wizard`, и все модификации и дополнения, сделанные в этой и предыдущей главах.

### Листинг 11.1

```

// ScratchBook.h : главный заголовочный файл приложения ScratchBook
//
#pragma once

#ifdef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h" // основные символы

// CScratchBookApp:
// Смотрите реализацию этого класса в файле ScratchBook.cpp
//

class CScratchBookApp : public CWinApp
{
public:
    CScratchBookApp();

    // Переопределения
public:
    virtual BOOL InitInstance();

    // Реализация
    afx_msg void OnAppAbout();
    DECLARE_MESSAGE_MAP()
};

```

---

## Листинг 11.2

```
// ScratchBook.cpp : Определяет поведение классов приложения
//

#include "stdafx.h"
#include "ScratchBook.h"
#include "MainFrm.h"

#include "ScratchBookDoc.h"
#include "ScratchBookView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// CScratchBookApp

BEGIN_MESSAGE_MAP(CScratchBookApp, CWinApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
    // Стандартные команды работы с файлами документов
    ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
END_MESSAGE_MAP()

// Конструктор CScratchBookApp

CScratchBookApp::CScratchBookApp()
{
    // TODO: добавьте сюда код конструктора
    // Поместите весь существенный код инициализации
    // в функцию InitInstance
}

// Единственный объект класса CScratchBookApp

CScratchBookApp theApp;

// Инициализация CScratchBookApp

BOOL CScratchBookApp::InitInstance()
{
    CWinApp::InitInstance();

    // Стандартная инициализация.
    // Если вы не используете какие-то из предоставленных
    // возможностей и хотите уменьшить размер конечного
    // исполняемого модуля, удалите из последующего кода
    // отдельные команды инициализации элементов, которые
    // вам не нужны.
    // Измените ключ, под которым ваши установки хранятся в реестре
```



```

// TODO: Измените эту строку параметра на что-нибудь подходящее,
// например, на имя вашей компании или организации
SetRegistryKey(_T("Local AppWizard-Generated Applications"));
LoadStdProfileSettings(4); // Загрузка стандартных установок
                             // из INI-файла (включая MRU)
// Регистрация шаблона документа приложения. Шаблоны
// документов служат связью между документами, окнами
// документов и окнами приложений
CSingleDocTemplate* pDocTemplate;
pDocTemplate = new CSingleDocTemplate(
    IDR_MAINFRAME,
    RUNTIME_CLASS(CScratchBookDoc),
    RUNTIME_CLASS(CMainFrame), // главное окно
                                // SDI-приложения
    RUNTIME_CLASS(CScratchBookView));
AddDocTemplate(pDocTemplate);
// Просмотр командной строки для обнаружения стандартных
// команд оболочки, DDE, открытия файлов
CCommandLineInfo cmdInfo;
ParseCommandLine(cmdInfo);
// Выполнение команд, указанных в командной строке.
// Вернет FALSE, если приложение было запущено с
// /RegServer, /Register, /Unregserver или /Unregister.
if (!ProcessShellCommand(cmdInfo))
    return FALSE;
// Показ и обновление единственного проинициализированного окна
m_pMainWnd->ShowWindow(SW_SHOW);
m_pMainWnd->UpdateWindow();

return TRUE;
}

// CAboutDlg диалог, используемый в App About

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

// Данные для диалога
    enum { IDD = IDD_ABOUTBOX };

protected:
    virtual void DoDataExchange(CDataExchange* pDX);
    // Поддержка DDX/DDV

// Реализация
protected:
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
}

```

```

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
END_MESSAGE_MAP()

// Команда приложения для выполнения диалога
void CScratchBookApp::OnAppAbout()
{
    CAboutDlg aboutDlg;
    aboutDlg.DoModal();
}

// Обработчики сообщений класса CScratchBookApp

```

---

### Листинг 11.3

```

// ScratchBookDoc.h : интерфейс класса CScratchBookDoc
//

#pragma once

class CLine : public CObject
{
protected:
    int m_X1, m_Y1, m_X2, m_Y2;

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);
};

class CScratchBookDoc : public CDocument
{
protected:
    CTypedPtrArray<CObArray, CLine*> m_LineArray;

public:
    void AddLine (int X1, int Y1, int X2, int Y2);
    CLine *GetLine (int Index);
    int GetNumLines ();

protected: // используется только для сериализации
    CScratchBookDoc();
    DECLARE_DYNCREATE(CScratchBookDoc)

// Атрибуты

```

```

public:

// Операции
public:

// Переопределения
    public:
        virtual BOOL OnNewDocument();
        virtual void Serialize(CArchive& ar);

// Реализация
public:
    virtual ~CScratchBookDoc();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Сгенерированные функции карты сообщений
protected:
    DECLARE_MESSAGE_MAP()
public:
    virtual void DeleteContents();
    afx_msg void On57633();
    afx_msg void OnUpdate57633(CCmdUI *pCmdUI);
    afx_msg void On57643();
    afx_msg void OnUpdate57643(CCmdUI *pCmdUI);
};

```

---

#### Листинг 11.4

```

// ScratchBookDoc.cpp : реализация класса CScratchBookDoc
//

#include "stdafx.h"
#include "ScratchBook.h"

#include "ScratchBookDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// CScratchBookDoc

IMPLEMENT_DYNCREATE(CScratchBookDoc, CDocument)

BEGIN_MESSAGE_MAP(CScratchBookDoc, CDocument)
    ON_COMMAND(57633, On57633)
    ON_UPDATE_COMMAND_UI(57633, OnUpdate57633)
    ON_COMMAND(57643, On57643)
    ON_UPDATE_COMMAND_UI(57643, OnUpdate57643)
END_MESSAGE_MAP()

```

```

// Конструктор/деструктор класса CScratchBookDoc

CScratchBookDoc::CScratchBookDoc()
{
    // TODO: добавьте сюда код одноразового конструктора
}

CScratchBookDoc::~CScratchBookDoc()
{
}

BOOL CScratchBookDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;

    // TODO: добавьте сюда код повторной инициализации
    // (SDI-документы будут использовать этот документ
    // многократно)

    return TRUE;
}

// Сериализация CScratchBookDoc

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

// Диагностика класса CScratchBookDoc

#ifdef _DEBUG
void CScratchBookDoc::AssertValid() const
{
    CDocument::AssertValid();
}

void CScratchBookDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}
#endif // _DEBUG

// Команды класса CScratchBookDoc

void CLine::Draw (CDC *PDC)

```

```

{
    PDC->MoveTo (m_X1, m_Y1);
    PDC->LineTo (m_X2, m_Y2);
}

void CScratchBookDoc::AddLine (int X1, int Y1, int X2, int Y2)
{
    CLine *PLine = new CLine (X1, Y1, X2, Y2);
    m_LineArray.Add (PLine);
}

CLine *CScratchBookDoc::GetLine (int Index)
{
    if (Index < 0 || Index > m_LineArray.GetUpperBound ())
        return 0;
    return m_LineArray.GetAt (Index);
}

int CScratchBookDoc::GetNumLines ()
{
    return m_LineArray.GetSize ();
}

void CScratchBookDoc::DeleteContents()
{
    // TODO: Добавьте сюда свой код или вызов базового
    // класса
    int Index = m_LineArray.GetSize ();
    while (Index-->0)
        delete m_LineArray.GetAt (Index);
    m_LineArray.RemoveAll ();

    CDocument::DeleteContents();
}

void CScratchBookDoc::On57633()
{
    // TODO: Добавьте сюда собственный код обработчика
    DeleteContents ();
    UpdateAllViews (0);
}

void CScratchBookDoc::OnUpdate57633(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика

    pCmdUI->Enable (m_LineArray.GetSize ());
}

void CScratchBookDoc::On57643()
{
    // TODO: Добавьте сюда собственный код обработчика
    int Index = m_LineArray.GetUpperBound ();
    if (Index > -1)
    {
        delete m_LineArray.GetAt (Index);
    }
}

```

```

        m_LineArray.RemoveAt (Index);
    }
    UpdateAllViews (0);
}

void CScratchBookDoc::OnUpdate57643(CCmdUI *pCmdUI)
{
    // TODC: Добавьте сюда собственный код обработчика
    pCmdUI->Enable (m_LineArray.GetSize ());
}

```

---

### Листинг 11.5

```

// ScratchBookView.h : интерфейс класса CScratchBookView
//

#pragma once

class CScratchBookView : public CView
{
protected:
    CString m_ClassName;
    int m_Dragging;
    HCURSOR m_HCross;
    CPoint m_PointOld;
    CPoint m_PointOrigin;

protected: // используется только для сериализации
    CScratchBookView();
    DECLARE_DYNCREATE(CScratchBookView)

// Атрибуты
public:
    CScratchBookDoc* GetDocument() const;

// Операции
public:

// Переопределения
public:
    virtual void OnDraw(CDC* pDC);
    // переопределена для прорисовки этого вида
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
protected:

// Реализация
public:
    virtual ~CScratchBookView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

```

```

// Сгенерированные функции карты сообщений
protected:
    DECLARE_MESSAGE_MAP()
public:
    afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
    afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
    afx_msg void OnMouseMove(UINT nFlags, CPoint point);
};

#ifdef _DEBUG // отладочная версия в файле ScratchBookView.cpp
inline CScratchBookDoc* CScratchBookView::GetDocument() const
{ return (CScratchBookDoc*)m_pDocument; }
#endif

```

---

## Листинг 11.6

```

// ScratchBookView.cpp : реализация класса CScratchBookView
//

#include "stdafx.h"
#include "ScratchBook.h"

#include "ScratchBookDoc.h"
#include "ScratchBookView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// CScratchBookView

IMPLEMENT_DYNCREATE(CScratchBookView, CView)

BEGIN_MESSAGE_MAP(CScratchBookView, CView)
    ON_WM_LBUTTONDOWN()
    ON_WM_LBUTTONUP()
    ON_WM_MOUSEMOVE()
END_MESSAGE_MAP()

// Конструктор/деструктор класса CScratchBookView

CScratchBookView::CScratchBookView()
{
    // TODO: добавьте сюда собственный код конструктора

    m_Dragging = 0;
    m_HCross = AfxGetApp()->LoadStandardCursor (IDC_CROSS);
}

CScratchBookView::~CScratchBookView()
{
}

```

```

BOOL CScratchBookView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Модифицируйте класс или стили окна,
    // добавляя и изменяя поля структуры cs

    m_ClassName = AfxRegisterWndClass
        (CS_HREDRAW | CS_VREDRAW,    // стили окна
        0,                          // без указателя
        (HBRUSH)::GetStockObject (WHITE_BRUSH),
        0);                          // задать белый фон
    cs.lpszClass = m_ClassName;
    // без значка

    return CView::PreCreateWindow(cs);
}

// Прорисовка CScratchBookView

void CScratchBookView::OnDraw(CDC* pDC)
{
    CScratchBookDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    // TODO: вставьте сюда код прорисовки собственных данных

    int Index = pDoc->GetNumLines ();
    while (Index--)
        pDoc->GetLine (Index)->Draw(pDC);
}

// Диагностика класса CScratchBookView

#ifdef _DEBUG
void CScratchBookView::AssertValid() const
{
    CView::AssertValid();
}

void CScratchBookView::Dump(CDumpContext& dc) const
{
    CView::Dump(dc);
}

CScratchBookDoc* CScratchBookView::GetDocument() const
// не отладочная версия встроена
{
    ASSERT(m_pDocument->IsKindOf (RUNTIME_CLASS (CScratchBookDoc)));
    return (CScratchBookDoc*)m_pDocument;
}
#endif // _DEBUG

// Обработчики сообщений класса CScratchBookView

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

```



```

{
    // TODO: Вставьте сюда собственный код обработчика
    // или вызов стандартного

    m_PointOrigin = point;
    m_PointOld = point;
    SetCapture ();
    m_Dragging = 1;
    RECT Rect;
    GetClientRect (&Rect);
    ClientToScreen (&Rect);
    ::ClipCursor (&Rect);

    CView::OnLButtonDown(nFlags, point);
}

void CScratchBookView::OnLButtonUp(UINT nFlags, CPoint point)
{
    // TODO: Вставьте сюда собственный код обработчика
    // или вызов стандартного
    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);

        CScratchBookDoc* PDoc = GetDocument();
        PDoc -> AddLine (m_PointOrigin.x,
            m_PointOrigin.y, point.x, point.y);

    }

    CView::OnLButtonUp(nFlags, point);
}

void CScratchBookView::OnMouseMove(UINT nFlags, CPoint point)
{
    // TODO: Вставьте сюда собственный код обработчика
    // или вызов стандартного

    ::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);
}

```

---

### Листинг 11.7

```

// MainFrm.h : интерфейс класса CMainFrame
//

#pragma once
class CMainFrame : public CFrameWnd
{

protected: // используется только для сериализации
    CMainFrame();
    DECLARE_DYNCREATE(CMainFrame)

// Атрибуты
public:

// Операции
public:

// Переопределения
public:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);

// Реализация
public:
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected: // встроенные элементы управления
    CStatusBar m_wndStatusBar;
    CToolBar m_wndToolBar;

// Сгенерированные функции карты сообщений
protected:
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    DECLARE_MESSAGE_MAP()
};

```

---

### Листинг 11.8

```

// MainFrm.cpp : реализация класса CMainFrame
//

#include "stdafx.h"
#include "ScratchBook.h"

```

```

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
    ON_WM_CREATE()
END_MESSAGE_MAP()

static UINT indicators[] =
{
    ID_SEPARATOR,           // индикатор строки состояния
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
};

// Конструктор/деструктор класса CMainFrame

CMainFrame::CMainFrame()
{
    // TODO: добавьте сюда код инициализации элементов
}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD |
WS_VISIBLE | CBRS_TOP | CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY |
CBRS_SIZE_DYNAMIC) || !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
    {
        TRACE0("Failed to create toolbar\n");
        return -1;      // не удалось создать панель инструментов
    }

    if (!m_wndStatusBar.Create(this) ||
        !m_wndStatusBar.SetIndicators(indicators,
            sizeof(indicators)/sizeof(UINT)))
    {
        TRACE0("Failed to create status bar\n");
        return -1;      // не удалось создать строку состояния
    }
}

```

```

        // TODO: Удалите три следующие строки, если не хотите,
        // чтобы панель инструментов была паркующей
        m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
        EnableDocking(CBRS_ALIGN_ANY);
        DockControlBar(&m_wndToolBar);

        return 0;
    }

    BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
    {
        if( !CFrameWnd::PreCreateWindow(cs) )
            return FALSE;
        // TODO: Модифицируйте стили или классы окна здесь,
        // добавляя или изменяя поля структуры cs

        return TRUE;
    }

    // Диагностика класса CMainFrame

#ifdef _DEBUG
    void CMainFrame::AssertValid() const
    {
        CFrameWnd::AssertValid();
    }

    void CMainFrame::Dump(CDumpContext& dc) const
    {
        CFrameWnd::Dump(dc);
    }

#endif // _DEBUG

    // Обработчики сообщений класса CMainFrame

```

## Резюме

---

Мы рассмотрели технику решения (в том числе и с помощью команд меню) задач класса документа, имеющих отношение к сохранению данных, и обеспечение доступа к данным с помощью функций-членов класса.

- *Хранение данных.* Если документ состоит из дискретных данных, наподобие графических фигур, необходимо определить класс для их хранения, отображения и выполнения других операций (например, класс `CLine` в программе `ScratchBook`). Класс документа может быть удобным хранилищем группы переменных или объектов внутри объекта, который является экземпляром одного из классов коллекций библиотеки MFC. Например, экземпляр MFC-класса `CObArray` может хранить в структуре данных, подобной массиву, коллекцию указателей на объекты (которые должны принадлежать классу, порожденному прямо или косвенно от класса `CObject`). Используя шаблон MFC-класса `CTypedPtrArray`, можно легко получить класс `CObArray`, предназначенный для работы с объектами специального класса. Примером такого объекта в программе `ScratchBook` может быть `m_LineArray`. Этот объект хранит набор указателей класса `CLine`.

- *Функции обработки данных.* Класс представления должен изменять или добавлять данные в процессе редактирования. Примерами функций класса представления являются `AddLine`, `GetLine` и `GetNumLines`. Класс документа должен определять функции, позволяющие классу представления получать или модифицировать данные документа. Например, функция `OnDraw` должна получать данные для того, чтобы повторно выводить их в окне представления.
- *Команды меню.* Команды меню, которые непосредственно изменяют данные документа, например, команды `Undo` и `Remove All` меню `Edit`, должны обрабатываться классом документа. Чтобы добавить в класс документа обработчик сообщения для команды меню, можно использовать вкладку `Class View` и окно `Properties`.
- *Обработчики сообщений.* Можно определять обработчики для двух типов сообщений команд меню: `COMMAND_UPDATE_UI`, передаваемого непосредственно перед отображением ниспадающего меню, содержащего команду, и `COMMAND`, передаваемого при выборе команды. Обработчик сообщения `COMMAND_UPDATE_UI` инициализирует пункт меню, используя функции передаваемого ему объекта `CcmdUI`. Если обработчик делает пункт меню недоступным, то связанная с ним комбинация клавиш также блокируется. Обработчик командного (`COMMAND`) сообщения выполняет команду меню.

# Глава 12

## Ввод-вывод

---

- Ввод-вывод в программе ScratchBook
- Ввод-вывод в программе MyScribe
- Альтернативные способы ввода-вывода

В этой главе рассматриваются приемы обмена данными между документом и дисковыми файлами. Для демонстрации базовых методов ввода-вывода в приложения ScratchBook и MyScribe добавлен код, реализующий стандартные команды меню File (New, Open..., Save и Save As...). Вы узнаете, как реализовать технологию drag-and-drop, позволяющую открывать файл, перетаскивая его значок из окна папки или Windows Explorer в окно приложения.

### *Ввод-вывод в программе ScratchBook*

---

Добавим в программу ScratchBook команды Open..., Save и Save As..., а также код, необходимый для их реализации. Предполагается, что исходные файлы программы содержат все модификации (см. гл. 10 и 11), сделанные ранее.

#### *Модификация меню File*

Открыв в Visual Studio проект ScratchBook, перейдите на вкладку Resource View для отображения списка ресурсов программы. Чтобы изменить меню программы, откройте редактор меню, выполнив двойной щелчок на идентификаторе IDR\_MAINFRAME. В редакторе меню откройте меню File. После команды New в меню File необходимо (используя методику, описанную в предыдущих главах) добавить команды:

- Open... с идентификатором ID\_FILE\_OPEN и надписью &Open...\tCtrl+O.
- Save с идентификатором ID\_FILE\_SAVE и надписью &Save\tCtrl+S.
- Save As... с идентификатором ID\_FILE\_SAVE\_AS и надписью Save &As...
- Разграничитель.
- Recent File с идентификатором ID\_FILE\_MRU\_FILE1 и надписью Recent File. 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 Application Wizard устанавливает максимальное количество последних использованных файлов равным 4. MFC хранит их имена в файле инициализации программы (ScratchBook.ini) в каталоге Windows, поэтому, когда пользователь выходит из программы и перезапускает ее, список этих файлов сохраняется.

Для этих команд “вручную” задавать комбинации клавиш не нужно, так как мастер Application Wizard уже определил их при создании приложения. В гл. 10 сгенерированное меню содержало все перечисленные выше команды, но они были удалены, так как в первых двух версиях программы ScratchBook не использовались. Закройте окно редактора меню.

Далее необходимо изменить строковый (string) ресурс программы для определения стандартного расширения файлов, отображаемых в диалоговых окнах Open и Save As. Для этого откройте редактор

строк Developer Studio, выполнив двойной щелчок на элементе String Table графа ResourceView. Первая строка в окне редактора имеет идентификатор IDR\_MAINFRAME. Она создана мастером Application Wizard и содержит информацию, относящуюся к программе ScratchBook. Чтобы модифицировать строку, откройте диалоговое окно Properties. Измените содержимое поля Caption как показано ниже. При редактировании строки *не* нажимайте клавишу Enter. Когда текст достигнет правого края текстового поля, его остаток будет перенесен на следующую строку автоматически.

**"ScratchBook\n\nScratchBook\nScratchBook Files  
(\* .drw) \n .drw\nScratchBook.Document\nScratchBook Document"**

- Строка "ScratchBook Files (\*.drw)" задает элемент, отображаемый в списке Files of type (или Save as type) диалогового окна Open (или Save As) и определяющий *стандартное расширение файлов программы*.
- Строка "(.drw)" задает стандартное расширение файлов. Если в процессе выполнения программы ScratchBook расширение файла при открытии или сохранении *не указано*, то в диалоговых окнах Open и Save As отобразится список всех файлов со стандартными расширениями. Стандартное расширение файла будет добавлено к его имени и в диалоговом окне Save As.

Инструкции, приведенные выше, относятся только к заданию стандартного расширения файлов в *существующей* программе. Соответствующие установки можно также выполнить и при создании приложения мастером Application Wizard – в его окне есть вкладка Document Template Strings, где можно ввести требуемые данные.

## Реализация команд

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 в меню File с последними использованными файлами.

- Команду New обрабатывает функция OnFileNew класса CWinApp (от которого порожден класс приложения ScratchBook). Эта функция вызывает виртуальную функцию DeleteContents (см. гл. 11) для удаления текущего содержимого документа, а затем инициализирует новый документ.
- Команду Open... обрабатывается функцией OnFileOpen класса CWinApp. Эта функция 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 (CScratchBookDoc::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.
- Команду Save обрабатывает функция OnFileSave класса CDocument (от которого порожден класс документа программы ScratchBook), 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 CScratchBookDoc::Serialize для выполнения собственно операции записи. Они также сохраняют полный путь к файлу и отображают его имя в строке заголовка. Если файл еще не был сохранен, то в поле File name диалогового окна Save As отобразится имя файла по умолчанию, которое создается добавлением к имени "Untitled" стандартного расширения файла (.drw для программы ScratchBook).

Следует помнить, что длина имени и расширения файла в Windows NT/2000/XP не ограничена восемью и тремя символами соответственно. 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 Stdlib.h: **MAX\_PATH**, **MAX\_DRIVE**, **MAX\_DIR**, **MAX\_FNAME** и **MAX\_EXT**. В Windows 98, например, имя файла может содержать до 255 символов.

## Чтение/запись данных

Мастер Application Wizard, создавая программу ScratchBook, определяет в файле ScratchBookDoc.cpp для функции `Serialize` только минимальную реализацию. 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 для этого файла набор функций для чтения и/или записи данных. Класс, объекты которого могут быть сериализованы, должен прямо или косвенно порождаться от MFC-класса `CObject`. При выполнении записи (т.е. если выбрана команда `Save` или `Save As...`) функция `IsStoring` класса `CArchive` возвращает значение `TRUE`, а при выполнении чтения (т.е. если выбрана команда `Open...` или команда вызова последнего использованного файла из меню `File`) функция `IsStoring` возвращает значение `FALSE`.

Класс документа в программе ScratchBook 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 `CScratchBookDoc::Serialize` можно дописать, просто добавив два обращения к функции `CObArray::Serialize`: одно для чтения и одно для записи.

```
// Сериализация CScratchBookDoc

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

Для каждого объекта класса `CLine` при *записи* данных в файл функция `CObArray::Serialize` записывает в файл информацию о классе объекта и вызывает функцию `Serialize` объекта, записывающую в файл данные объекта. При *чтении* данных из файла функция `CObArray::Serialize` читает информацию класса из файла, динамически создает объект соответствующего класса (`CLine`) и сохраняет указатель на объект. Затем она вызывает функцию `Serialize` объекта для чтения во вновь созданный объект данных из файла.

Чтобы для класса `CLine` обеспечить поддержку сериализации, включите в его определение два макроса (`DECLARE_SERIAL` и `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 с помощью функции `CObArray::Serialize` обеспечивают запись в файл (или чтение из файла) информации о классе объекта.

Добавьте макрокоманду `DECLARE_SERIAL` и конструктор по умолчанию в определение класса `CLine` в файле `ScratchBookDoc.h`, как показано ниже. Имя класса передается макросу `DECLARE_SERIAL` как параметр.

```
class CLine : public CObject
{
protected:
    int m_X1, m_Y1, m_X2, m_Y2;
    CLine ()
    {}
    DECLARE_SERIAL (CLine)

public:
```

В файле `ScratchBookDoc.cpp` макрокоманда `IMPLEMENT_SERIAL` добавляется сразу перед определением функции `CLine::Draw`. Первый параметр, переданный макросу `IMPLEMENT_SERIAL`, – это имя самого класса, а второй – имя базового класса. Третий параметр – это *номер версии*, идентифицирующий определенную версию программы (нельзя задавать номер версии -1 (минус один)). 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 `ScratchBook` задан номер 1. В более поздних версиях он увеличивается при каждом изменении формата данных.

```
// Команды класса CScratchBookDoc

IMPLEMENT_SERIAL (CLine, CObject, 1)

void CLine::Draw (CDC *PDC)
```

Далее для сериализации класса `Cline` необходимо добавить в класс `CLine` функцию `Serialize`, вызываемую в методе `CObArray::Serialize` для чтения или записи данных каждой строки. Добавьте объявление функции `Serialize` в файле `ScratchBookDoc.h` в раздел `public` определения класса `CLine`.

```
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 (CArchive &ar);
};
```

В файл реализации `ScratchBookDoc.cpp` после определения `CLine::Draw` добавьте определение функции `Serialize`. 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 используются для чтения и записи данных различных типов и определяются классом `CArchive`.

```

void CLine::Draw (CDC *PDC)
{
    PDC->MoveTo (m_X1, m_Y1);
    PDC->LineTo (m_X2, m_Y2);
}

void CLine::Serialize (CArchive &ar)
{
    if (ar.IsStoring())
        ar << m_X1 << m_Y1 << m_X2 << m_Y2;
    else
        ar >> m_X1 >> m_Y1 >> m_X2 >> m_Y2;
}

```

Объект класса, хранящий 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 предоставляемые классом `CArchive` перегруженные операторы `<<` и `>>`.

## Регистрация drw-файлов

Следует добавить в системный реестр Windows информацию, позволяющую открыть файл программы `ScratchBook` (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 `ScratchBook.cpp`. Эти функции регистрируют в реестре Windows связь между стандартным расширением файла программы `ScratchBook` (`.drw`) и самой программой (такая связь остается в реестре до тех пор, пока она не будет изменена явным образом). Практически эта связь проявляется в том, что:

- объект, представляющий любой файл с `drw`-расширением, отображает значок программы `ScratchBook`;
- двойной щелчок на файле с `drw`-расширением запускает программу `ScratchBook`, если она еще не запущена, и открывает файл в этой программе.

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 (она вводится в передаваемый в шаблон строковый ресурс с идентификатором `IDR_MAINFRAME`) была доступна объекту приложения.

```

AddDocTemplate(pDocTemplate);

EnableShellOpen ();
RegisterShellFileTypes ();

// Просмотр командной строки для обнаружения стандартных
// команд оболочки, DDE, открытия файлов
CCommandLineInfo cmdInfo;

```

## Открытие файлов

Технология “drag-and-drop” позволяет открывать файл, перемещая значок файла из папки Windows (а также из окна программы Explorer или любого другого окна, поддерживающего эту технологию) в окно программы. Для поддержки операции перетаскивания в программе ScratchBook вызовите функцию `CWnd::DragAcceptFiles` для объекта главного окна. Поместите это обращение внутри функции `InitInstance` класса приложения в файле `ScratchBook.cpp` после вызова `UpdateWindow`. 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 создается главное окно и присваивается значение переменной `m_pMainWnd`.

```
m_pMainWnd->ShowWindow(SW_SHOW);
m_pMainWnd->UpdateWindow();

m_pMainWnd->DragAcceptFiles ();

return TRUE;
```

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

## Учет изменений в данных

Чтобы иметь сведения о том, содержит ли документ не сохраненные данные, класс `CDocument` поддерживает *флаг изменений*. MFC проверяет этот флаг перед вызовом функции `DeleteContents` класса документа, используемой 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 не сохраненные данные), то выводится соответствующее сообщение. Класс `CDocument` устанавливает значение флага равным `FALSE`, когда документ открыт и сохранен. Для установки флага равным `TRUE` при каждом изменении данных документа вызывается функция `CDocument::SetModifiedFlag`. Флаг изменений можно установить в `TRUE`, вызывая эту функцию без аргументов (параметр функции `SetModifiedFlag` имеет значение по умолчанию `TRUE`). Чтобы установить флаг изменений в `FALSE`, необходимо явно передать это значение (обычно эту задачу выполняет класс `CDocument`). Добавьте в функцию `AddLine` и в обработчики команд `Remove All` и `Undo` в файле `ScratchBookDoc.cpp` вызов функции `SetModifiedFlag`.

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

void CScratchBookDoc::On57633()
{
    // TODO: Добавьте сюда собственный код обработчика
    DeleteContents ();
    UpdateAllViews (0);
    SetModifiedFlag ();
}
```

```

void CScratchBookDoc::On57643()
{
    // TODO: Добавьте сюда собственный код обработчика
    int Index = m_LineArray.GetUpperBound ();
    if (Index > -1)
    {
        delete m_LineArray.GetAt (Index);
        m_LineArray.RemoveAt (Index);
    }
    UpdateAllViews (0);
    SetModifiedFlag ();
}

```

## Текст программы ScratchBook

Ниже, в листингах (12.1 – 12.8) приведены модифицированные в этой главе тексты программы ScratchBook.

---

### Листинг 12.1

```

// ScratchBook.h : главный заголовочный файл приложения ScratchBook
//
#pragma once

#ifdef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"          // основные символы

// CScratchBookApp:
// Смотрите реализацию этого класса в файле ScratchBook.cpp
//

class CScratchBookApp : public CWinApp
{
public:
    CScratchBookApp();

    // Переопределения
public:
    virtual BOOL InitInstance();

    // Реализация
    afx_msg void OnAppAbout();
    DECLARE_MESSAGE_MAP()
};

```

---

### Листинг 12.2

```

// ScratchBook.cpp : Определяет поведение классов приложения
//

#include "stdafx.h"
#include "ScratchBook.h"

```

```

#include "MainFrm.h"

#include "ScratchBookDoc.h"
#include "ScratchBookView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// CScratchBookApp

BEGIN_MESSAGE_MAP(CScratchBookApp, CWinApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
    // Стандартные команды работы с файлами документов
    ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
END_MESSAGE_MAP()

// Конструктор CScratchBookApp

CScratchBookApp::CScratchBookApp()
{
    // TODO: добавьте сюда код конструктора
    // Поместите весь существенный код инициализации
    // в функцию InitInstance
}

// Единственный объект класса CScratchBookApp

CScratchBookApp theApp;

// Инициализация CScratchBookApp

BOOL CScratchBookApp::InitInstance()
{
    CWinApp::InitInstance();

    // Стандартная инициализация.
    // Если вы не используете какие-то из предоставленных
    // возможностей и хотите уменьшить размер конечного
    // исполняемого модуля, удалите из последующего кода
    // отдельные команды инициализации элементов, которые
    // вам не нужны.
    // Измените ключ, под которым ваши установки
    // хранятся в реестре.
    // TODO: Измените строку параметра на что-нибудь подходящее,
    // например, на имя вашей компании или организации
    SetRegistryKey(_T("Local AppWizard-Generated Applications"));
    LoadStdProfileSettings(4); // Загрузка стандартных установок
                               // из INI-файла (включая MRU)
    // Регистрация шаблона документа приложения. Шаблоны
    // документов служат связью между документами, окнами
    // документов и окнами приложений
    CSingleDocTemplate* pDocTemplate;

```

```

pDocTemplate = new CSingleDocTemplate(
    IDR_MAINFRAME,
    RUNTIME_CLASS(CScratchBookDoc),
    RUNTIME_CLASS(CMainFrame),           // главное окно
                                         // SDI-приложения
    RUNTIME_CLASS(CScratchBookView));
AddDocTemplate(pDocTemplate);

EnableShellOpen ();
RegisterShellFileTypes ();

// Просмотр командной строки для обнаружения стандартных
// команд оболочки, DDE, открытия файлов
CCommandLineInfo cmdInfo;
ParseCommandLine(cmdInfo);
// Выполнение команд, указанных в командной строке.
// Вернет FALSE, если приложение было запущено с
// /RegServer, /Register, /Unregserver или /Unregister.
if (!ProcessShellCommand(cmdInfo))
    return FALSE;
// Показ и обновление единственного
// проинициализированного окна
m_pMainWnd->ShowWindow(SW_SHOW);
m_pMainWnd->UpdateWindow();

m_pMainWnd->DragAcceptFiles ();

return TRUE;
}

// CAboutDlg диалог, используемый в App About

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

// Данные для диалога
    enum { IDD = IDD_ABOUTBOX };

protected:
    virtual void DoDataExchange(CDataExchange* pDX);
    // Поддержка DDX/DDV

// Реализация
protected:
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
}

```

```

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
END_MESSAGE_MAP()

// Команда приложения для выполнения диалога
void CScratchBookApp::OnAppAbout()
{
    CAboutDlg aboutDlg;
    aboutDlg.DoModal();
}

// Обработчики сообщений класса CScratchBookApp

```

---

### Листинг 12.3

```

// ScratchBookDoc.h : интерфейс класса CScratchBookDoc
//

#pragma once

class CLine : public CObject
{
protected:
    int m_X1, m_Y1, m_X2, m_Y2;
    CLine ()
    {}
    DECLARE_SERIAL (CLine)

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 (CArchive &ar);
};

class CScratchBookDoc : public CDocument
{
protected:
    CTypedPtrArray<CObArray, CLine*> m_LineArray;

public:
    void AddLine (int X1, int Y1, int X2, int Y2);
    CLine *GetLine (int Index);
    int GetNumLines ();

protected: // используется только для сериализации
    CScratchBookDoc();
    DECLARE_DYNCREATE(CScratchBookDoc)

// Атрибуты
public:

```

```

// Операции
public:

// Переопределения
public:
    virtual BOOL OnNewDocument();
    virtual void Serialize(CArchive& ar);

// Реализация
public:
    virtual ~CScratchBookDoc();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Сгенерированные функции карты сообщений
protected:
    DECLARE_MESSAGE_MAP()
public:
    virtual void DeleteContents();
    afx_msg void On57633();
    afx_msg void OnUpdate57633(CCmdUI *pCmdUI);
    afx_msg void On57643();
    afx_msg void OnUpdate57643(CCmdUI *pCmdUI);
};

```

---

## Листинг 12.4

```

// ScratchBookDoc.cpp : реализация класса CScratchBookDoc
//

#include "stdafx.h"
#include "ScratchBook.h"

#include "ScratchBookDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// CScratchBookDoc

IMPLEMENT_DYNCREATE(CScratchBookDoc, CDocument)

BEGIN_MESSAGE_MAP(CScratchBookDoc, CDocument)
    ON_COMMAND(57633, On57633)
    ON_UPDATE_COMMAND_UI(57633, OnUpdate57633)
    ON_COMMAND(57643, On57643)
    ON_UPDATE_COMMAND_UI(57643, OnUpdate57643)
END_MESSAGE_MAP()

```



```

// Конструктор/деструктор класса CScratchBookDoc

CScratchBookDoc::CScratchBookDoc()
{
    // TODO: добавьте сюда код одnorазового конструктора
}

CScratchBookDoc::~CScratchBookDoc()
{
}

BOOL CScratchBookDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;

    // TODO: добавьте сюда код повторной инициализации
    // (SDI-документы будут использовать этот документ
    // многократно)

    return TRUE;
}

// Сериализация CScratchBookDoc

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

// Диагностика класса CScratchBookDoc

#ifdef _DEBUG
void CScratchBookDoc::AssertValid() const
{
    CDocument::AssertValid();
}

void CScratchBookDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}
#endif // _DEBUG

```

```

// Команды класса CScratchBookDoc

IMPLEMENT_SERIAL (CLine, CObject, 1)

void CLine::Draw (CDC *PDC)
{
    PDC->MoveTo (m_X1, m_Y1);
    PDC->LineTo (m_X2, m_Y2);
}

void CLine::Serialize (CArchive &ar)
{
    if (ar.IsStoring())
        ar << m_X1 << m_Y1 << m_X2 << m_Y2;
    else
        ar >> m_X1 >> m_Y1 >> m_X2 >> m_Y2;
}

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

CLine *CScratchBookDoc::GetLine (int Index)
{
    if (Index < 0 || Index > m_LineArray.GetUpperBound ())
        return 0;
    return m_LineArray.GetAt (Index);
}

int CScratchBookDoc::GetNumLines ()
{
    return m_LineArray.GetSize ();
}

void CScratchBookDoc::DeleteContents()
{
    // TODO: Добавьте сюда собственный код или вызов базового
    // класса
    int Index = m_LineArray.GetSize ();
    while (Index--)
        delete m_LineArray.GetAt (Index);
    m_LineArray.RemoveAll ();

    CDocument::DeleteContents();
}

void CScratchBookDoc::On57633()
{
    // TODO: Добавьте сюда собственный код обработчика
    DeleteContents ();
    UpdateAllViews (0);
    SetModifiedFlag ( );
}

```

```

void CScratchBookDoc::OnUpdate57633(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика

    pCmdUI->Enable (m_LineArray.GetSize ());
}

void CScratchBookDoc::On57643()
{
    // TODO: Добавьте сюда собственный код обработчика
    int Index = m_LineArray.GetUpperBound ();
    if (Index > -1)
    {
        delete m_LineArray.GetAt (Index);
        m_LineArray.RemoveAt (Index);
    }
    UpdateAllViews (0);
    SetModifiedFlag ();
}

void CScratchBookDoc::OnUpdate57643(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->Enable (m_LineArray.GetSize ());
}

```

---

## Листинг 12.5

```

// ScratchBookView.h : интерфейс класса CScratchBookView
//

#pragma once

class CScratchBookView : public CView
{
protected:
    CString m_ClassName;
    int m_Dragging;
    HCURSOR m_HCross;
    CPoint m_PointOld;
    CPoint m_PointOrigin;

protected: // используется только для сериализации
    CScratchBookView();
    DECLARE_DYNCREATE(CScratchBookView)

// Атрибуты
public:
    CScratchBookDoc* GetDocument() const;

// Операции
public:

// Переопределения

```

```

    public:
        virtual void OnDraw(CDC* pDC);
        // переопределена для прорисовки этого вида
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    protected:

    // Реализация
    public:
        virtual ~CScratchBookView();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

    protected:

    // Сгенерированные функции карты сообщений
    protected:
        DECLARE_MESSAGE_MAP()
    public:
        afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
        afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
        afx_msg void OnMouseMove(UINT nFlags, CPoint point);
};

#ifdef _DEBUG // отладочная версия в файле ScratchBookView.cpp
inline CScratchBookDoc* CScratchBookView::GetDocument() const
{ return (CScratchBookDoc*)m_pDocument; }
#endif

```

---

## Листинг 12.6

```

// ScratchBookView.cpp : реализация класса CScratchBookView
//

#include "stdafx.h"
#include "ScratchBook.h"

#include "ScratchBookDoc.h"
#include "ScratchBookView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// CScratchBookView

IMPLEMENT_DYNCREATE(CScratchBookView, CView)

BEGIN_MESSAGE_MAP(CScratchBookView, CView)
    ON_WM_LBUTTONDOWN()
    ON_WM_LBUTTONUP()
    ON_WM_MOUSEMOVE()
END_MESSAGE_MAP()

```

```

// Конструктор/деструктор класса CScratchBookView

CScratchBookView::CScratchBookView()
{
    // TODO: добавьте сюда собственный код конструктора

    m_Dragging = 0;
    m_HCross = AfxGetApp()->LoadStandardCursor (IDC_CROSS);
}

CScratchBookView::~CScratchBookView()
{
}

BOOL CScratchBookView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Модифицируйте класс или стили окна,
    // добавляя и изменяя поля структуры cs

    m_ClassName = AfxRegisterWndClass
        (CS_HREDRAW | CS_VREDRAW,    // стили окна
        0,                          // без указателя
        (HBRUSH)::GetStockObject (WHITE_BRUSH),
                                     // задать белый фон
        0);                          // без значка
    cs.lpszClass = m_ClassName;

    return CView::PreCreateWindow(cs);
}

// Прорисовка CScratchBookView

void CScratchBookView::OnDraw(CDC* pDC)
{
    CScratchBookDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    // TODO: вставьте сюда код прорисовки собственных данных

    int Index = pDoc->GetNumLines ();
    while (Index--)
        pDoc->GetLine (Index)->Draw(pDC);
}

// Диагностика класса CScratchBookView

#ifdef _DEBUG
void CScratchBookView::AssertValid() const
{
    CView::AssertValid();
}

void CScratchBookView::Dump(CDumpContext& dc) const
{
    CView::Dump(dc);
}

```

```

CScratchBookDoc* CScratchBookView::GetDocument() const
// не отладочная версия встроена
{
    ASSERT(m_pDocument->
        IsKindOf(RUNTIME_CLASS(CScratchBookDoc)));
    return (CScratchBookDoc*)m_pDocument;
}
#endif // _DEBUG

// Обработчики сообщений класса CScratchBookView

void CScratchBookView::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: Вставьте сюда собственный код обработчика
    // или вызов стандартного

    m_PointOrigin = point;
    m_PointOld = point;
    SetCapture ();
    m_Dragging = 1;
    RECT Rect;
    GetClientRect (&Rect);
    ClientToScreen (&Rect);
    ::ClipCursor (&Rect);

    CView::OnLButtonDown(nFlags, point);
}

void CScratchBookView::OnLButtonUp(UINT nFlags, CPoint point)
{
    // TODO: Вставьте сюда собственный код обработчика
    // или вызов стандартного
    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);

        CScratchBookDoc* PDoc = GetDocument();
        PDoc -> AddLine (m_PointOrigin.x,
            m_PointOrigin.y, point.x, point.y);
    }

    CView::OnLButtonUp(nFlags, point);
}

void CScratchBookView::OnMouseMove(UINT nFlags, CPoint point)
{

```

```

// TODO: Вставьте сюда собственный код обработчика
// или вызов стандартного

::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);
}

```

---

### Листинг 12.7

```

// MainFrm.h : интерфейс класса CMainFrame
//

#pragma once
class CMainFrame : public CFrameWnd
{
protected: // используется только для сериализации
    CMainFrame();
    DECLARE_DYNCREATE(CMainFrame)

// Атрибуты
public:

// Операции
public:

// Переопределения
public:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);

// Реализация
public:
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected: // встроенные элементы управления
    CStatusBar m_wndStatusBar;
    CToolBar m_wndToolBar;

// Сгенерированные функции карты сообщений
protected:

```

```

    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    DECLARE_MESSAGE_MAP()
};

```

---

## Листинг 12.8

```

// MainFrm.cpp : реализация класса CMainFrame
//

#include "stdafx.h"
#include "ScratchBook.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
    ON_WM_CREATE()
END_MESSAGE_MAP()

static UINT indicators[] =
{
    ID_SEPARATOR,           // индикатор строки состояния
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
};

// Конструктор/деструктор класса CMainFrame

CMainFrame::CMainFrame()
{
    // TODO: добавьте сюда код инициализации элементов
}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD |
        WS_VISIBLE | CBRS_TOP | CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY |
        CBRS_SIZE_DYNAMIC) || !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
    {

```



```

        TRACE0("Failed to create toolbar\n");
        return -1;          // не удалось создать панель
                             // инструментов
    }

    if (!m_wndStatusBar.Create(this) ||
        !m_wndStatusBar.SetIndicators(indicators,
        sizeof(indicators)/sizeof(UINT)))
    {
        TRACE0("Failed to create status bar\n");
        return -1;          // не удалось создать строку состояния
    }
    // TODO: Удалите три следующие строки, если не хотите,
    // чтобы панель инструментов была паркующей
    m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
    EnableDocking(CBRS_ALIGN_ANY);
    DockControlBar(&m_wndToolBar);

    return 0;
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    if( !CFrameWnd::PreCreateWindow(cs) )
        return FALSE;
    // TODO: Модифицируйте стили или классы окна здесь,
    // добавляя или изменяя поля структуры cs

    return TRUE;
}

// Диагностика класса CMainFrame

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
    CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
    CFrameWnd::Dump(dc);
}

#endif // _DEBUG

// Обработчики сообщений класса CMainFrame

```

## ***Ввод-вывод в программе MyScribe***

Добавим в программу MyScribe команды New, Open..., Save, Save As..., а также код, необходимый для их реализации. Предполагается, что исходные файлы программы содержат все модификации (см. гл. 10), сделанные ранее.

## Модификация меню

Открыв в Visual Studio проект MyScribe, перейдите на вкладку Resource View для отображения списка ресурсов программы. Чтобы изменить меню программы, откройте редактор меню, выполнив двойной щелчок на идентификаторе IDR\_MAINFRAME. В редакторе меню откройте меню File. Перед командами печати в меню File необходимо (используя методику, описанную в предыдущих главах) добавить команды:

- New с идентификатором ID\_FILE\_NEW и надписью &New\тCtrl+N.
- Open... с идентификатором ID\_FILE\_OPEN и надписью &Open...\тCtrl+O.
- Save с идентификатором ID\_FILE\_SAVE и надписью &Save\тCtrl+S.
- Save As... с идентификатором ID\_FILE\_SAVE\_AS и надписью Save &As...  
Следом за командами печати добавьте разделитель и команду:
- Recent File с идентификатором ID\_FILE\_MRU\_FILE1 и надписью Recent File.

Чтобы задать стандартное расширение файлов .txt, измените строковый ресурс с идентификатором IDR\_MAINFRAME, по методике, описанной для программы ScratchBook (параграф “Модификация меню File” в текущей главе). Строка в поле Caption должна иметь вид:

```
"MyScribe\п\пMyScribe\пMyScribe Files  
(* .txt)\п.txt\пMyScribe.Document\пMyScribe Document"
```

## Реализация команд

Необходимо переопределить виртуальную функцию DeleteContents в виде функции класса документа CMyscribeDoc. MFC вызывает эту функцию перед созданием нового документа, открытием существующего или окончанием работы программы для удаления данных документа. Чтобы создать объявление функции и ее минимальное определение, используйте вкладку Class View и окно Properties (см. параграф “Удаление данных” гл. 11). К определению функции DeleteContents, сгенерированной мастером в файле MyScribeDoc.cpp, добавьте фрагмент, выделенный полужирным шрифтом. Класс CEditView присваивает окну представления класс EDIT. Окно, принадлежащее этому классу, называется *текстовым элементом управления* (или *редактируемым полем*). При его поддержке не только сохраняется отображаемый текст, но и обеспечивается большой набор средств редактирования. Так как класс представления порожден от класса CEditView, текст документа сохраняется в самом окне представления.

Чтобы его удалить, функция DeleteContents сначала вызывает функции GetFirstViewPosition и GetNextView класса CDocument для получения указателя на класс представления документа. Как будет описано в гл. 13, документ может иметь более одного присоединенного к нему представления. Эти две функции позволяют обращаться ко *всем* присоединенным представлениям. Функция GetFirstViewPosition получает индекс первого объекта представления (*единственного* в программе MyScribe). Затем функция GetNextView возвращает указатель на этот объект и заменяет параметр Pos индексом следующего представления в списке или присваивает параметру Pos значение 0, если другие представления отсутствуют (как в данном случае). Функция GetNextView может изменять значение *ссылочного* параметра Pos.

```
// Команды класса CMyscribeDoc  
  
void CMyscribeDoc::DeleteContents()  
{  
    // TODO: Добавьте сюда собственный код или вызов  
    // базового класса
```

```

POSITION Pos = GetFirstViewPosition();
CEditView *PCeditView = (CEditView *)GetNextView (Pos);
if (PCeditView)
    PCeditView->SetWindowText ("");

CDocument::DeleteContents();
}

```

Если для объекта представления функция `GetNextView` возвращает ненулевой адрес, то функция `DeleteContents` использует его для вызова функции `CWnd::SetWindowText`, чтобы записать в окно пустую строку, тем самым удаляя текст документа. Если же `GetNextView` возвращает нулевое значение, то `DeleteContents` не удаляет текст документа. При завершении программы MFC удаляет объект представления, а затем вызывает функцию `DeleteContents`. В этом случае функция `GetNextView` возвращает нуль, потому что объект представления удален, и `DeleteContents` удалять текст не нужно.

Для хранения текста программа использует буфер. Размер буфера ограничивается приблизительно 50 Кбайтами. При попытке открыть файл большего размера класс `CEditView` выводит предупреждение и не читает файл. При вводе текста большого объема прием символов прекращается, причем никакое предупреждение не выдается.

Для использования возможностей технологии “drag-and-drop”, включите в функцию `InitInstance` в файле `MyScribe.cpp` вызов функции `DragAcceptFiles`.

```

// Показ и обновление единственного
// проинициализированного окна
m_pMainWnd->ShowWindow(SW_SHOW);
m_pMainWnd->UpdateWindow();
m_pMainWnd->DragAcceptFiles();

```

```
return TRUE;
```

Устанавливать флаг изменений в программе `MyScribe` (как в программе `ScratchBook`) не нужно, так как это автоматически делает класс `CEditView`. В программе `MyScribe` не нужно изменять функцию `Serialize` класса документа, поскольку при построении класса представления из класса `CEditView` мастер `Application Wizard` автоматически добавляет требуемый код в `Serialize`. Этот код вызывает функцию `SerializeRaw` класса `CEditView`, которая читает и записывает данные, отображаемые окном представления в виде чистого текста. Информация о классе или версии в файле не сохраняется.

## Текст программы MyScribe

Ниже, в листингах (12.9—12.16) приведены модифицированные в этой главе тексты программы `MyScribe`.

---

### Листинг 12.9

```

// MyScribe.h : главный заголовочный файл приложения MyScribe
//
#pragma once

#ifdef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"          // основные символы

```

```

// CMyScribeApp:
// Смотри реализацию этого класса в файле MyScribe.cpp
//

class CMyScribeApp : public CWinApp
{
public:
    CMyScribeApp();

    // Переопределения
public:
    virtual BOOL InitInstance();

    // Реализация
    afx_msg void OnAppAbout();
    DECLARE_MESSAGE_MAP()
};

```

---

## Листинг 12.10

```

// MyScribe.cpp : Определяет поведение классов приложения.
//

#include "stdafx.h"
#include "MyScribe.h"
#include "MainFrm.h"

#include "MyScribeDoc.h"
#include "MyScribeView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// CMyScribeApp

BEGIN_MESSAGE_MAP(CMyScribeApp, CWinApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
    // Стандартные команды работы с файлами документов
    ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
    // Стандартные команды печати
    ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()

// Конструктор класса CMyScribeApp

CMyScribeApp::CMyScribeApp()
{
    // TODO: добавьте сюда код конструктора
    // Поместите весь существенный код инициализации
    // в функцию InitInstance
}

```

```

// Единственный объект класса CMyScribeApp

CMyScribeApp theApp;

// Инициализация CMyScribeApp

BOOL CMyScribeApp::InitInstance()
{
    CWinApp::InitInstance();

    // Стандартная инициализация.
    // Если вы не используете какие-то из предоставленных
    // возможностей и хотите уменьшить размер конечного
    // исполняемого модуля, удалите из последующего кода
    // отдельные команды инициализации элементов, которые
    // вам не нужны.
    // Измените ключ, под которым ваши установки хранятся
    // в реестре.
    // TODO: Измените строку параметра на что-нибудь подходящее,
    // например, на имя вашей компании или организации
    SetRegistryKey(_T("Local AppWizard-Generated Applications"));
    LoadStdProfileSettings(4); // Загрузка стандартных установок
                                // из INI-файла (включая MRU)

    // Регистрация шаблона документа приложения. Шаблоны
    // документов служат связью между документами, окнами
    // документов и окнами приложений
    CSingleDocTemplate* pDocTemplate;
    pDocTemplate = new CSingleDocTemplate(
        IDR_MAINFRAME,
        RUNTIME_CLASS(CMyScribeDoc),
        RUNTIME_CLASS(CMainFrame),           // главное окно
                                           // SDI-приложения
        RUNTIME_CLASS(CMyScribeView));
    AddDocTemplate(pDocTemplate);
    // Просмотр командной строки для обнаружения стандартных
    // команд оболочки, DDE, открытия файлов
    CCommandLineInfo cmdInfo;
    ParseCommandLine(cmdInfo);
    // Выполнение команд, указанных в командной строке.
    // Вернет FALSE, если приложение было запущено с
    // /RegServer, /Register, /Unregserver или /Unregister.
    if (!ProcessShellCommand(cmdInfo))
        return FALSE;
    // Показ и обновление единственного
    // проинициализированного окна
    m_pMainWnd->ShowWindow(SW_SHOW);
    m_pMainWnd->UpdateWindow();
    m_pMainWnd->DragAcceptFiles();

    return TRUE;
}

// CAboutDlg диалог, используемый в App About

class CAboutDlg : public CDialog

```

```

{
public:
    CAboutDlg();

    // Данные для диалога
    enum { IDD = IDD_ABOUTBOX };

protected:
    virtual void DoDataExchange(CDataExchange* pDX);
    // Поддержка DDX/DDV

    // Реализация
protected:
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
END_MESSAGE_MAP()

// Команда приложения на выполнение диалога
void CMyScribeApp::OnAppAbout()
{
    CAboutDlg aboutDlg;
    aboutDlg.DoModal();
}

// Обработчики сообщений класса CMyScribeApp

```

---

### Листинг 12.11

```

// MyScribeDoc.h : интерфейс класса CMyScribeDoc
//

#pragma once

class CMyScribeDoc : public CDocument
{
protected: // используется только для сериализации
    CMyScribeDoc();
    DECLARE_DYNCREATE(CMyScribeDoc)

    // Атрибуты
public:

    // Операции
public:

```

```

// Переопределения
public:
    virtual BOOL OnNewDocument();
    virtual void Serialize(CArchive& ar);

// Реализация
public:
    virtual ~CMyScribeDoc();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Сгенерированные функции карты сообщений
protected:
    DECLARE_MESSAGE_MAP()
public:
    virtual void DeleteContents();
};

```

---

### Листинг 12.12

```

// MyScribeDoc.cpp : реализация класса CMyScribeDoc
//

#include "stdafx.h"
#include "MyScribe.h"

#include "MyScribeDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// CMyScribeDoc

IMPLEMENT_DYNCREATE(CMyScribeDoc, CDocument)

BEGIN_MESSAGE_MAP(CMyScribeDoc, CDocument)
END_MESSAGE_MAP()

// Конструктор/деструктор класса CMyScribeDoc

CMyScribeDoc::CMyScribeDoc()
{
    // TODO: добавьте сюда одноразовый код конструктора
}

CMyScribeDoc::~CMyScribeDoc()
{
}

```

```

BOOL CMyScribeDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;

    ((CEditView*)m_viewList.GetHead())->SetWindowText(NULL);

    // TODO: добавьте сюда код повторной инициализации
    // (SDI-документы будут многократно использовать этот
    // документ)

    return TRUE;
}

// Сериализация CMyScribeDoc

void CMyScribeDoc::Serialize(CArchive& ar)
{
    // CEditView содержит элемент управления,
    // выполняющий сериализацию
    ((CEditView*)m_viewList.GetHead())->SerializeRaw(ar);
}

// Диагностика класса CMyScribeDoc

#ifdef _DEBUG
void CMyScribeDoc::AssertValid() const
{
    CDocument::AssertValid();
}

void CMyScribeDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}
#endif // _DEBUG

// Команды класса CMyScribeDoc

void CMyScribeDoc::DeleteContents()
{
    // TODO: Добавьте сюда собственный код или
    // вызов базового класса

    POSITION Pos = GetFirstViewPosition();
    CEditView *PCEditView = (CEditView *)GetNextView (Pos);
    if (PCEditView)
        PCEditView->SetWindowText ("");

    CDocument::DeleteContents();
}

```



---

**Листинг 12.13**

```
// MyScribeView.h : интерфейс класса CMyscribeView
//

#pragma once

class CMyscribeView : public CEditView
{
protected: // используется только для сериализации
    CMyscribeView();
    DECLARE_DYNCREATE(CMyscribeView)

// Атрибуты
public:
    CMyscribeDoc* GetDocument() const;

// Операции
public:

// Переопределения
public:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
protected:
    virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
    virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
    virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);

// Реализация
public:
    virtual ~CMyscribeView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Сгенерированные функции карты сообщений
protected:
    DECLARE_MESSAGE_MAP()
};

#ifdef _DEBUG // отладочная версия в файле MyScribeView.cpp
inline CMyscribeDoc* CMyscribeView::GetDocument() const
{ return (CMyscribeDoc*)m_pDocument; }
#endif
```

---

**Листинг 12.14**

```
// MyScribeView.cpp : реализация класса CMyscribeView
//

#include "stdafx.h"
#include "MyScribe.h"
```

```

#include "MyScribeDoc.h"
#include "MyScribeView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// CMyscribeView

IMPLEMENT_DYNCREATE(CMyScribeView, CEditView)

BEGIN_MESSAGE_MAP(CMyScribeView, CEditView)
    // Стандартные команды печати
    ON_COMMAND(ID_FILE_PRINT, CEditView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_DIRECT, CEditView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_PREVIEW,
CEditView::OnFilePrintPreview)
END_MESSAGE_MAP()

// Конструктор/деструктор класса CMyscribeView

CMyscribeView::CMyscribeView()
{
    // TODO: добавьте сюда код конструктора
}

CMyscribeView::~CMyscribeView()
{
}

BOOL CMyscribeView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Измените класс или стиль окна здесь,
    // добавляя или изменяя поля структуры cs

    BOOL bPreCreated = CEditView::PreCreateWindow(cs);
    cs.style &= ~(ES_AUTOHSCROLL|WS_HSCROLL);
    // Разрешить перенос слов

    return bPreCreated;
}

// Печать в CMyscribeView

BOOL CMyscribeView::OnPreparePrinting(CPrintInfo* pInfo)
{
    // Подготовка печати в CEditView по умолчанию
    return CEditView::OnPreparePrinting(pInfo);
}

void CMyscribeView::OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo)
{
    // Начало печати в CEditView по умолчанию

```

```

    CEditView::OnBeginPrinting(pDC, pInfo);
}

void CMyScribeView::OnEndPrinting(CDC* pDC, CPrintInfo* pInfo)
{
    // Окончание печати в CEditView по умолчанию
    CEditView::OnEndPrinting(pDC, pInfo);
}

// Диагностика класса CMyScribeView

#ifdef _DEBUG
void CMyScribeView::AssertValid() const
{
    CEditView::AssertValid();
}

void CMyScribeView::Dump(CDumpContext& dc) const
{
    CEditView::Dump(dc);
}

CMyScribeDoc* CMyScribeView::GetDocument() const
// не отладочная версия встроена
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CMyScribeDoc)));
    return (CMyScribeDoc*)m_pDocument;
}
#endif // _DEBUG

// Обработчики сообщений класса CMyScribeView

```

---

### Листинг 12.15

```

// MainFrm.h : интерфейс класса CMainFrame
//

#pragma once
class CMainFrame : public CFrameWnd
{

protected: // используется только для сериализации
    CMainFrame();
    DECLARE_DYNCREATE(CMainFrame)

// Атрибуты
public:

// Операции
public:

// Переопределения
public:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);

// Реализация

```

```

public:
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected: // Встроенные элементы управления
    CStatusBar m_wndStatusBar;
    CToolBar m_wndToolBar;

    // Сгенерированные функции карты сообщений
protected:
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    DECLARE_MESSAGE_MAP()
};

```

---

## Листинг 12.16

```

// MainFrm.cpp : реализация класса CMainFrame
//

#include "stdafx.h"
#include "MyScribe.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
    ON_WM_CREATE()
END_MESSAGE_MAP()

static UINT indicators[] =
{
    ID_SEPARATOR,           // индикатор строки состояния
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
};

// Конструктор/деструктор класса CMainFrame

CMainFrame::CMainFrame()
{
    // TODO: добавьте сюда код инициализации элементов
}

```

```

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT,
        WS_CHILD | WS_VISIBLE | CBRS_TOP | CBRS_GRIPPER |
        CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC) ||
        !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
    {
        TRACE0("Failed to create toolbar\n");
        return -1; // не удалось создать панель инструментов
    }

    if (!m_wndStatusBar.Create(this) ||
        !m_wndStatusBar.SetIndicators(indicators,
        sizeof(indicators)/sizeof(UINT)))
    {
        TRACE0("Failed to create status bar\n");
        return -1; // не удалось создать строку состояния
    }
    // TODO: Удалите три следующие строки, если не хотите, чтобы
    // панель инструментов была паркующей
    m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
    EnableDocking(CBRS_ALIGN_ANY);
    DockControlBar(&m_wndToolBar);

    return 0;
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    if( !CFrameWnd::PreCreateWindow(cs) )
        return FALSE;
    // TODO: Измените стиль или класс окна здесь,
    // изменяя и добавляя поля структуры cs

    return TRUE;
}

// Диагностика класса CMainFrame

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
    CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
    CFrameWnd::Dump(dc);
}

```

```
#endif //_DEBUG

// Обработчики сообщений класса CMainFrame
```

## Альтернативные способы ввода-вывода

Мы рассмотрели чтение и запись переменных, относящихся к базовым типам данных, с помощью перегруженных операторов << и >> класса CArchive, а также чтение и запись объектов с помощью функции Serialize. Класс CArchive предоставляет также функции Read и Write для чтения или записи произвольных блоков данных, например, текстового буфера или структуры:

- Функция CArchive::Read располагает двумя параметрами (lpBuf – адрес памяти для приема данных, nMax – число читаемых байтов) и имеет следующий вид:

```
UINT Read (void* lpBuf, UINT nMax);
```

- Функция CArchive::Write располагает двумя параметрами (lpBuf – адрес записываемого блока данных, nMax – число записываемых байтов), записывает только необработанные байты из указанного источника. Она не добавляет информацию о классе и не форматирует данные:

```
void Write (const void* lpBuf, UINT nMax);
```

При сериализации класса CArchive документ последовательно считывается из файла на диске, данные сохраняются и обрабатываются внутри программы, а документ последовательно записывается обратно в файл на диске. Данные файла сохраняются в двоичном формате и вместе с фактическими данными документа содержат информацию о версии программы и классе. В качестве альтернативы MFC позволяет использовать низкоуровневые методы ввода-вывода, которые дают возможность перемещать файловый указатель на заданное место в файле, читать или записывать определенное число байтов и выполнять другие операции.

Использование альтернативных методов делает возможной реализацию функции ввода-вывода для файлов, не удовлетворяющих требованиям сериализации (например, позволяют прочитать или записать простые текстовые файлы либо часть большого файла, не сохраняя весь файл). Один из способов файлового ввода-вывода заключается в использовании MFC-класса CFile. Объект класса CFile присоединяется к указанному файлу, предоставляя обширный набор функций для выполнения универсальных операций двоичного ввода и вывода данных без буферизации. Создать собственный объект класса CFile можно в любой момент, когда понадобится выполнить операцию ввода-вывода. Кроме того, MFC присоединяет объект класса CFile к открытому файлу – объекту CArchive, передаваемому в функцию Serialize.

К этому объекту класса CFile можно обратиться, вызвав функцию CArchive::GetFile. (Можно вызывать функции Seek, Read, Write и др.) Ввод-вывод текстового файла выполняется с использованием класса CStdioFile, производного от CFile, а чтение и запись – с использованием класса CMemFile, также порожденного от класса CFile. Полная информация о классах CFile, CStdioFile и CMemFile содержится в справочной системе.

Операции над файлами можно выполнять, используя функции ввода-вывода Win32 API. Win32 API обеспечивает полный набор функций чтения, записи и управления файлами. Два дополнительных способа ввода-вывода сводятся к использованию потока или низкоуровневых функций ввода-вывода, предоставляемых библиотекой периода выполнения. Можно также использовать функции библиотеки iostream. Они рассмотрены в справочной системе.

## Резюме

---

На примере редактора графики и текстового редактора мы узнали, как добавить операции ввода-вывода в класс документа программы и реализовать их в рамках стандартных команд меню.

- *Обработчики сообщений.* Библиотека MFC предоставляет обработчик сообщений для команд New, Open..., Save, Save As... и списка последних открытых файлов меню File. Для определения обработчиков этих команд вкладка Class View и окно Properties не используются, но написать код поддержки необходимо. MFC-обработчики команд Open... и Save As... отображают стандартные диалоговые окна Open... и Save As... для ввода имени файла и пути к нему.
- *Стандартное расширение файла.* Чтобы определить стандартное расширение файлов, используемое диалоговыми окнами Open... и Save As..., можно отредактировать строковый ресурс с идентификатором IDR\_MAINFRAME в редакторе ресурсов Visual Studio. Другой способ задания стандартного расширения файла заключается в обращении при создании *новой* программы к вкладке Document Template Strings диалогового окна мастера Application Wizard. Программу можно изменить так, чтобы пользователь мог открыть файл, выполняя двойной щелчок на объекте файла со стандартным расширением (например, .drw) в папке Windows. Для этого поместите вызовы функций EnableShellOpen и RegisterShellFileTypes класса CWinApp внутри функции InitInstance *после* вызова метода AddDocTemplate.
- *Сериализация.* Обработчики команд New, Open..., Save, Save As... и списка последних открытых файлов меню File открывают файл, а затем вызывают функцию Serialize класса документа для чтения или записи данных. Функция Serialize также вызывается библиотекой MFC при перетаскивании значка файла в окно программы или открытии файла двойным щелчком в папке Windows. Мастер Application Wizard обычно определяет внутри класса документа простейшую функцию Serialize. В нее необходимо добавить код для чтения или записи данных, сохраненных в классе документа. Функции Serialize передается ссылка на объект класса CArchive, присоединенный к открытому файлу, предоставляющий функции для чтения или записи данных. Функция Serialize позволяет читать или записывать переменные базовых типов, используя перегруженные операторы << и >> применительно к объектам класса CArchive. Функция Serialize читает или записывает данные объекта, вызывая его собственную функцию Serialize. Если объект принадлежит к определенному классу, необходимо добавить функцию Serialize в тот класс, который читает или записывает данные объекта.
- *Текстовая сериализация.* Если класс представления порожден от класса CEditView, мастер Application Wizard автоматически добавляет требуемый код в функцию Serialize. В этом коде вызывается функция SerializeRaw класса CeditView для текстового ввода-вывода.
- *Удаление данных.* Перед инициализацией нового документа, открытием существующего или завершением работы программы MFC вызывает функцию DeleteContents класса документа. Для удаления данных документа необходимо переопределить эту функцию.
- *Учет изменений.* При каждом изменении данных программы класс документа должен вызывать функцию SetModifiedFlag класса CDocument. Вызов этой функции сообщает MFC, что документ изменился. Тогда MFC позволяет сохранить данные на диске перед их удалением из памяти. Флаг изменений устанавливается автоматически при изменении текста, если класс представления порожден от класса CEditView.
- *Drag-and-drop.* Поддержку технологии “drag-and-drop” можно реализовать, поместив вызов метода DragAcceptFiles объекта главного окна *после* вызова функции ProcessShellCommand в функции InitInstance.
- *Альтернативный ввод-вывод.* Читать или записывать произвольные блоки данных можно, вызывая функции Read или Write класса CArchive.

## Глава 13

# Управление окнами представления

---

- Прокрутка окна
- Разделение окна
- Перерисовка окна
- Текст программы ScratchBook

Прокрутка окна представления делает возможным просмотр и редактирование документа, размеры которого превышают размеры окна. Разделение позволяет создавать несколько окон представления одного документа и прокручивать его в каждом окне отдельно. Эти два средства добавляются к программе относительно просто (практически вся работа выполняется специальными MFC-классами). В этой главе показано, как добавить в MFC-программу возможности прокрутки и разделения окна представления. Средства прокрутки и разделения окон будут добавлены в программу ScratchBook – в исходные файлы приложения, созданные в предыдущей главе.

## Прокрутка окна

---

Созданная к данному моменту версия программы ScratchBook для рисунка, размеры которого превышают размеры окна представления, обеспечивает просмотр и редактирование только части рисунка, помещающейся внутри окна. Чтобы просматривать и редактировать любую часть рисунка, размеры которого больше размеров окна, следует добавить к окну представления вертикальные и горизонтальные *полосы прокрутки*, а также код, поддерживающий прокрутку. Следует заметить, что в программу текстового редактора MyScribe добавлять средства прокрутки нет необходимости, так как класс CEditView, от которого порождается класс представления данной программы, автоматически решает эту задачу.

Как и класс CEditView в программе MyScribe, специальный класс представления CScrollView порождается от класса общего назначения CView. Измените класс представления программы ScratchBook таким образом, чтобы он порождался от класса CScrollView, а не CView. При порождении класса представления от класса CScrollView полосы прокрутки автоматически добавляются в окно представления и предоставляется большая часть кода для поддержки операций прокрутки. Кроме этого необходимо добавить некоторый собственный код. Чтобы класс представления порождался от класса CScrollView, необходимо в исходных файлах ScratchBookView.h и ScratchBookView.cpp заменить все вхождения класса CView на CScrollView.

При генерации *новой* программы можно сгенерировать класс представления от класса CScrollView с помощью мастера Application Wizard. Для этого в диалоговом окне мастера на вкладке Generated Classes выберите имя класса представления в списке сверху, а затем в списке Base class выберите класс CScrollView.

## Логические и фактические координаты

Когда документ открывается впервые, левый верхний угол рисунка отображается в левом верхнем углу окна (как в предыдущей версии программы). Однако при прокрутке документа с помощью полосы прокрутки MFC корректирует значение атрибута, называемого *началом области просмотра*. Начало области просмотра определяет положение текста или графики относительно окна. Изначально



точка с координатами (0, 0) появляется в левом верхнем углу окна представления, а точка с координатами (150, 30) – на расстоянии 150 пикселей от левого края окна и 30 пикселей от его верхнего края.

Если прокрутить документ вниз на 20 пикселей, то MFC скорректирует начало области просмотра так, что точка с координатами (0, 0) не будет видна (все, что выводится за пределы окна, не отображается), а точка с координатами (150, 30) будет выведена на расстоянии 10 пикселей от верхней границы окна. После того как MFC скорректирует начало области просмотра, функция OnDraw перерисует линии в окне представления, задавая *те же* координаты для каждой линии. Из-за изменения начала области просмотра, линии будут выведены *с учетом прокрутки*. Привлекательность такой системы состоит в том, что координаты в области отображения пересчитываются автоматически и не нужно модифицировать функцию OnDraw для работы с полосами прокрутки.

При программировании отображения следует помнить об отличии между двумя системами координат:

- *Логические координаты.* Координаты, заданные при рисовании объекта, называют логическими координатами.
- *Координаты устройства (фактические).* Фактические координаты объекта внутри окна называют координатами устройства. Координаты устройства – это, к примеру, координаты курсора мыши относительно начала координат текущего окна представления. В вышеупомянутом примере координаты устройства после прокрутки окна представления – (150, 10).

Все координаты, передаваемые функциям рисования MFC (например, MoveTo и LineTo), – это логические координаты. Однако некоторые из функций программы (например, обработчики сообщений, непосредственно приходящих от мыши или других устройств позиционирования), используют в качестве рабочих параметров координаты устройства (указателя мыши). До момента добавления к программе средств прокрутки логические координаты совпадали с координатами устройства. Теперь же (после добавления средств прокрутки) необходимо внести изменения, позволяющие преобразовывать логические координаты в фактические и наоборот. Необходимо преобразовать координаты указателя мыши, переданные обработчику сообщения мыши, из фактических в логические так, чтобы линии рисунка отображались правильно. Для этого добавьте в функцию OnLButtonDown в файле ScratchBookView.cpp выделенный полужирным шрифтом код. Чтобы преобразовать координаты устройства в логические, необходимо использовать контекст устройства, который относится к данному окну. Объект контекста устройства управляет выводом информации в окно, сохраняет атрибуты режима рисования и предоставляет функции для отображения текста или графики.

```
void CScratchBookView::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: Вставьте сюда собственный код обработчика
    // или вызов стандартного
    CClientDC clientDC (this);
    OnPrepareDC (&clientDC);
    clientDC.DPtoLP (&point);

    // проверка нахождения курсора внутри области
    // окна представления
    CSize ScrollSize = GetTotalSize ();
    CRect ScrollRect (0, 0, ScrollSize.cx, ScrollSize.cy);
    if (!ScrollRect.PtInRect (point))
        return;
```

Первый добавленный оператор создает объект контекста, относящийся к окну представления. Второй – вызывает функцию OnPrepareDC класса CScrollView, корректирующую начало области просмотра (это один из атрибутов рисунка, сохраняемых в объекте контекста устройства) на основании текущей позиции прокрученного рисунка. Третий оператор вызывает функцию DPtoLP класса

CClientDC, преобразующую координаты курсора, сохраненные в параметре point, из фактических в логические для объекта контекста по отношению к новому (установленному) началу области просмотра. Поскольку координаты, хранящиеся в параметре point, преобразованы в логические, то их уже можно использовать для рисования линии.

При создании собственного объекта контекста устройства его необходимо передать функции OnPrepareDC для коррекции начала области просмотра. Объект контекста устройства, передаваемый функции OnDraw, уже имеет правильно установленное начало области просмотра, скорректированное для прокрученного рисунка.

Обработчики сообщений мыши OnMouseMove и OnLButtonUp следует изменить аналогичным образом. Эти функции создают объект контекста устройства; следовательно, необходимо добавить в них обращения к функциям OnPrepareDC и DPTOLP. В функцию OnMouseMove следует внести такие изменения (выделены полужирным шрифтом):

```
void CScratchBookView::OnMouseMove(UINT nFlags, CPoint point)
{
    // TODO: Вставьте сюда собственный код обработчика
    // или вызов стандартного

    CClientDC ClientDC (this);
    OnPrepareDC (&ClientDC);
    ClientDC.DPtoLP (&point);

    CSize ScrollSize = GetTotalSize();
}
```

В функцию OnLButtonUp следует внести такие изменения (выделены полужирным шрифтом):

```
void CScratchBookView::OnLButtonUp(UINT nFlags, CPoint point)
{
    // TODO: Вставьте сюда собственный код обработчика
    // или вызов стандартного
    if (m_Dragging)
    {
        m_Dragging = 0;
        ::ReleaseCapture ();
        ::ClipCursor (NULL);
        CClientDC ClientDC (this);
        OnPrepareDC (&ClientDC);
        ClientDC.DPtoLP (&point);

        ClientDC.SetROP2 (R2_NOT);
    }
}
```

## Границы рисунка в окне представления

MFC-класс CScrollView обеспечивает прокрутку рисунка при перемещении бегунка полосы прокрутки. Следовательно, библиотеке MFC должна быть передана информация о *размерах* рисунка. В этом параграфе в программу ScratchBook будут добавлены операторы, передающие MFC размеры рисунка, для чего следует переопределить виртуальную функцию OnInitialUpdate в классе представления программы. С помощью вкладки Class View и раздела Overrides окна Properties создайте начальную версию переопределенной функции. Выберите виртуальную функцию OnInitialUpdate в списке Messages. В окне редактора введите выделенный полужирным текст в ее определение в файле ScratchBookView.cpp:

```
void CScratchBookView::OnInitialUpdate()
{
}
```

```

CScrollView::OnInitialUpdate();

// TODO: Добавьте сюда собственный код или
// вызов базового класса

SIZE Size = {800, 600};
SetScrollSizes (MM_TEXT, Size);
}

```

Непосредственно перед *первым* отображением документа в окне представления MFC вызывает виртуальную функцию `OnInitialUpdate` класса представления. Функция `SetScrollSizes` класса `CScrollView` сообщает MFC размер рисунка. Горизонтальные и вертикальные размеры рисунка записываются в структуру `Size`, передаваемую во втором параметре. Полнофункциональная программа рисования обычно позволяет задавать размер каждого нового рисунка. Это особенно важно при печати, так как позволяет в процессе создания рисунка учитывать размер страницы, на которой он будет напечатан. Размер рисунка сохраняется классом документа, а при сохранении рисунка в файле его размеры сохраняются вместе с другими данными рисунка.

В текущей версии программы `ScratchBook` размер рисунка установлен постоянным: 800 пикселей в ширину и 600 – в высоту. Если приложение изменяет размер документа (например, при вводе или удалении данных), то вызывается функция `SetScrollSizes`. Лучше всего это сделать внутри функции `OnUpdate` класса представления, вызываемой при каждом изменении данных в документе (см. далее параграф “Перерисовка окна”). В параграфе “Средства прокрутки” гл. 18 рассматривается использование этой методики. Первый параметр, переданный функции `SetScrollSizes`, задает *режим отображения* – атрибут рисунка, сохраняемый объектом контекста устройства. Режим отображения определяет систему координат и единицы измерения, используемые для вывода текста и графики. Программа `ScratchBook` (как и другие программы в данной книге) использует режим отображения `MM_TEXT`, в котором все единицы заданы в пикселях, горизонтальные координаты увеличиваются слева вправо, а вертикальные – сверху вниз. Другие режимы отображения кратко описаны в гл. 19. Если из функции `OnInitialUpdate` вместо функции `SetScrollSizes` вызвать функцию `CScrollView::SetScaleToFitSize`, то текст и графика в окне представления *масштабируются* таким образом, чтобы весь документ помещался внутри текущего окна представления. При этом полосы прокрутки будут отсутствовать, поскольку такое масштабирование устраняет потребность в прокрутке.

У функции `SetScrollSizes` есть и другие параметры, позволяющие управлять шагом прокрутки. Если щелкнуть кнопкой мыши на полосе прокрутки (но не на кнопке со стрелкой или бегунке), то окно представления прокручивается на одну *страницу*. По умолчанию страница равна 1/10 размера документа в направлении прокрутки. Если щелкнуть на кнопке со стрелкой, то окно представления прокручивается на одну *строку*. По умолчанию строка равна 1/10 страницы. В третьем и четвертом параметрах функции `SetScrollSizes` задаются размеры страницы и строки.

Если окно представления больше рисунка и за пределами рисунка нарисована линия, то прокрутка работает некорректно. Для рисунков с заданным размером в программу необходимо добавить средства, предотвращающие рисование линий *вне* области рисунка. Сначала добавим в функцию `OnDraw` операторы, позволяющие при каждой перерисовке окна представления задавать внизу и справа ограничивающие линии, показывающие пользователю границы рисунка. Добавьте строки, выделенные полужирным шрифтом, в функцию `OnDraw` в файл `ScratchBookView.cpp`. В данном фрагменте программы функция `GetTotalSize` класса `CScrollView` возвращает объект класса `CSize` с размерами рисунка, заданными при вызове функции `SetScrollSizes`. Затем эти размеры используются для отображения справа и снизу линий, ограничивающих рисунок.

```

// Прорисовка CScratchBookView

void CScratchBookView::OnDraw(CDC* pDC)

```

```

{
    CScratchBookDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    // TODO: вставьте сюда код прорисовки собственных данных
    CSize ScrollSize = GetTotalSize();
    pDC->MoveTo (ScrollSize.cx, 0);
    pDC->LineTo (ScrollSize.cx, ScrollSize.cy);
    pDC->LineTo (0, ScrollSize.cy);

    int Index = pDoc->GetNumLines();

```

Операторы, предотвращающие размещение линий вне области рисунка, следует добавить в обработчик сообщений OnLButtonDown. Для этого перепишите функцию OnLButtonDown в файле ScratchBookView.cpp (новые или измененные строки выделены полужирным шрифтом). Так как классу представления передается сообщение мыши, то ее указатель должен находиться внутри окна представления; при этом он может находиться вне области рисунка. Чтобы удостовериться в том, что указатель находится в области рисунка, определяется объект ScrollRect, являющийся экземпляром MFC-класса CRect, и ему присваиваются размеры рисунка, возвращаемые функцией GetTotalSize. Затем вызывается функция CRect::PtInRect для объекта ScrollRect и ей передаются координаты указателя. Функция PtInRect возвращает значение TRUE, если координаты указателя попадают в прямоугольную область, сохраняемую в объекте ScrollRect. Если указатель находится за пределами рисунка, выполняется возврат из функции OnLButtonDown.

```

void CScratchBookView::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: Вставьте сюда собственный код обработчика
    // или вызов стандартного
    CClientDC ClientDC (this);
    OnPrepareDC (&ClientDC);
    ClientDC.DPtoLP (&point);

    // проверка нахождения курсора внутри области
    // окна представления
    CSize ScrollSize = GetTotalSize();
    CRect ScrollRect (0, 0, ScrollSize.cx, ScrollSize.cy);
    if (!ScrollRect.PtInRect (point))
        return;

    // сохранение позиции курсора, захват мыши и
    // установка флага перемещения
    m_PointOrigin = point;
    m_PointOld = point;
    SetCapture ();
    m_Dragging = 1;

    // ограничение перемещений курсора мыши
    ClientDC.LPtoDP (&ScrollRect);
    CRect ViewRect;
    GetClientRect (&ViewRect);
    CRect IntRect;
    IntRect.IntersectRect (&ScrollRect, &ViewRect);
    ClientToScreen (&IntRect);
    ::ClipCursor (&IntRect);

```

```

        CScrollView::OnLButtonDown(nFlags, point);
    }

```

В предыдущей версии программы ScratchBook указатель перемещался внутри окна представления как угодно. Теперь функция OnLButtonDown не только предотвращает *начало* рисования линии вне области рисунка, но и препятствует *выходу* линии за границы этой области. Следовательно, данная функция ограничивает перемещение указателя в окне представления областью рисунка. Иными словами, она ограничивает перемещение указателя зоной пересечения рисунка и окна представления. В добавленном фрагменте программы вызывается функция CDC::LPToDP для преобразования логических координат рисунка, сохраняемых в объекте ScrollRect, в координаты устройства. Затем определяется объект ViewRect класса CRect, которому присваиваются координаты окна представления, возвращаемые функцией GetClientRect. Далее определяется объект IntRect и вызывается функция CRect::IntersectRect для задания ему координат зоны пересечения области рисунка и окна представления.

Функция CWnd::ClientToScreen преобразовывает значение IntRect в координаты экрана и передает их в функцию ::ClipCursor, что позволяет ограничить перемещение указателя. Изменим программу так, чтобы форма указателя изменялась на крестообразную внутри области рисунка (это означает, что линию можно рисовать) и восстанавливалась в прежнем виде вне области рисунка (линия не может быть нарисована). Добавьте в определение класса CScratchBookView в файле ScratchBookView.h переменную m\_HArrow.

```

class CScratchBookView : public CScrollView
{
protected:
    CString m_ClassName;
    int m_Dragging;
    HCURSOR m_HCross;
    HCURSOR m_HArrow;
    CPoint m_PointOld;
    CPoint m_PointOrigin;

protected: // используется только для сериализации
    CScratchBookView();

```

А затем внутри конструктора класса CScratchBookView в файле ScratchBookView.cpp инициализируйте переменную m\_HArrow. Здесь переменной m\_HArrow присваивается дескриптор стандартного указателя мыши в форме стрелки.

```

// Конструктор/деструктор класса CScratchBookView

CScratchBookView::CScratchBookView()
{
    // TODO: добавьте сюда собственный код конструктора

    m_Dragging = 0;
    m_HCross = AfxGetApp()->LoadStandardCursor (IDC_CROSS);
    m_HArrow = AfxGetApp()->LoadStandardCursor (IDC_ARROW);
}
// Конструктор/деструктор класса CScratchBookView

CScratchBookView::CScratchBookView()
{
    // TODO: добавьте сюда собственный код конструктора

```

```

    m_Dragging = 0;
    m_HCross = AfxGetApp()->LoadStandardCursor (IDC_CROSS);
    m_HArrow = AfxGetApp()->LoadStandardCursor (IDC_ARROW);
}

```

Затем в функции OnMouseMove в файле ScratchBookView.cpp переместите объявление объекта ClientDC и вызовы методов OnPrepareDC и DPtoLP в начало функции и добавьте строки, отмеченные полужирным шрифтом. Вместо постоянного отображения крестообразного указателя обработчик OnMouseMove показывает его только внутри области рисунка ScrollRect. Когда указатель находится вне этой области, обработчик отображает указатель-стрелку.

```

void CScratchBookView::OnMouseMove(UINT nFlags, CPoint point)
{
    // TODO: Вставьте сюда собственный код обработчика
    // или вызов стандартного

    CClientDC ClientDC (this);
    OnPrepareDC (&ClientDC);
    ClientDC.DPtoLP (&point);

    CSize ScrollSize = GetTotalSize();
    CRect ScrollRect(0, 0, ScrollSize.cx, ScrollSize.cy);
    if (ScrollRect.PtInRect (point))
        ::SetCursor (m_HCross);
    else
        ::SetCursor (m_HArrow);

    if (m_Dragging)
    {
        ClientDC.SetROP2 (R2_NOT);
    }
}

```

## Разделение окна

Для решения некоторых задач редактирования удобным средством служит разделение окна представления на две отдельные части (два окна), называемых также *панелями*. Границу между двумя панелями называют *линией разбивки*. В этом параграфе к окну программы ScratchBook будет добавлена *вешка разбивки*, позволяющая выполнить разбивку окна. Оба окна представления отображают один рисунок, а прокручиваются независимо друг от друга, отображая различные его части. Чтобы разделить окно, выполните двойной щелчок на вешке разбивки (окно разделится на две равные панели) или перетащите линию разбивки в требуемую позицию. В этом параграфе для примера мы построим средства деления исходного окна по горизонтали. Когда окно разделено на два по горизонтали, вертикальная полоса прокрутки является для них общей, однако каждая панель имеет собственную горизонтальную полосу прокрутки. В горизонтальном направлении панели прокручиваются независимо друг от друга. Чтобы добавить в программу средства разделения, измените класс главного окна. Добавьте в файле MainFrm.h в начало определения класса CMainFrame объявление объекта m\_SplitterWnd.

```

#pragma once
class CMainFrame : public CFrameWnd
{
protected:
    CSplitterWnd m_SplitterWnd;

protected: // используется только для сериализации

```

```
CMainFrame();
DECLARE_DYNCREATE(CMainFrame)
```

Созданный объект `m_SplitterWnd` служит для создания и управления разделенным окном и представляет собой экземпляр MFC-класса `CSplitterWnd`, порожденного от класса `CWnd`.

Далее необходимо переопределить виртуальную функцию `OnCreateClient` как функцию-члена класса `CMainFrame`. После добавления функции (таким же способом, как и описанной выше функции `OnInitialUpdate`) измените сгенерированную функцию в файле `MainFrm.cpp`, удалив вызов функции `CFrameWnd::OnCreateClient` и добавив строки, выделенные полужирным шрифтом.

```
BOOL CMainFrame::OnCreateClient(LPCREATESTRUCT lpcs,
CCreateContext* pContext)
{
    // TODO: Добавьте сюда собственный код или вызов базового
    // класса
    return m_SplitterWnd.Create
    (this,          // родительское окно разделенного окна;
    1,             // максимальное число строк;
    2,             // максимальное число столбцов;
    CSize(20, 20), // минимальный размер окна представления;
    pContext);     // информация о контексте устройства
}
```

Заданная по умолчанию версия функции `OnCreateClient`, определенная в классе `CFrameWnd`, создает единственное окно представления, заполняющее область главного окна (при создании главного окна библиотека MFC вызывает эту виртуальную функцию). Переопределенная версия этой функции вызывает функцию `CSplitterWnd::Create` для объекта `m_SplitterWnd`, чтобы создать разделенное окно вместо окна представления. Разделенное окно является дочерним по отношению к главному окну приложения. В свою очередь, каждая панель – дочернее окно разделенного окна. Разделенное окно первоначально содержит одну панель. При выполнении двойного щелчка на вешке разбивки создается вторая панель. Первый параметр, переданный функции `Create`, указывает родительское окно для разделенного окна. Параметр `this` делает последнее дочерним окном по отношению к главному.

Максимальное число разделительных панелей по вертикали задается вторым параметром. Значение 1 показывает, что пользователь не может делить окно горизонтальной разделительной линией (следовательно, вешка горизонтальной разбивки в окне не появится). Третий параметр определяет максимальное число разделительных панелей по горизонтали. Значение 2 показывает, что можно делить окно на левую и правую панель. Четвертый параметр задает минимальные горизонтальный и вертикальный размеры панели. Пятый параметр содержит переданную в функцию `OnCreateClient` информацию о контексте устройства.

## Перерисовка окна

MFC автоматически вызывает функцию `CScratchBookView::OnDraw` для перерисовки панели при каждом изменении данных в окне (например, пользователь расширил окно или удалил перекрывающее окно). Каждая из панелей управляется отдельным объектом представления, т.е. отдельным экземпляром класса `CScratchBookView`. Если нарисовать линию в одной панели, то другую панель также понадобится перерисовать для того, чтобы линия появилась в *обоих* окнах (при условии, что вторая панель прокручена к области рисунка, содержащей линию). Однако MFC *не* вызывает функцию `OnDraw` для второго объекта представления автоматически. После прорисовки линии в одном окне

представления необходимо явно вызвать функцию `CDocument::UpdateAllViews` для объекта документа, чтобы MFC вызвала `OnDraw` для другого представления.

Добавьте выделенное полужирным шрифтом обращение к методу `UpdateAllViews` в функцию `OnLButtonUp` в файле `ScratchBookView.cpp`. Функция `UpdateAllViews` вынуждает программу вызывать функцию `OnDraw` для всех связанных с документом представлений, *кроме* указанного в первом параметре. В программе `ScratchBook` передача в качестве первого параметра значения `this` в функцию `UpdateAllViews` вызывает функцию `OnDraw` только для другого представления. Если линия нарисована в одной панели, то она появится и в другой. Текущее окно представления не нуждается в перерисовке, так как новая линия в нем уже отображается.

```
void CScratchBookView::OnLButtonUp(UINT nFlags, CPoint point)
{
    // TODO: Вставьте сюда собственный код обработчика
    // или вызов стандартного
    if (m_Dragging)
    {
        m_Dragging = 0;
        ::ReleaseCapture ();
        ::ClipCursor (NULL);
        CClientDC ClientDC (this);
        OnPrepareDC (&ClientDC);
        ClientDC.DPtoLP (&point);

        ClientDC.SetROP2 (R2_NOT);
        ClientDC.MoveTo (m_PointOrigin);
        ClientDC.LineTo (m_PointOld);
        ClientDC.SetROP2 (R2_COPYPEN);
        ClientDC.MoveTo (m_PointOrigin);
        ClientDC.LineTo (point);

        CScratchBookDoc* PDoc = GetDocument();
        PDoc -> AddLine (m_PointOrigin.x,
            m_PointOrigin.y, point.x, point.y);

        PDoc->UpdateAllViews (this, 0, PCLine);
    }

    CScrollView::OnLButtonUp(nFlags, point);
}
```

Чтобы перерисовать *все* окна представлений, следует в функцию `UpdateAllViews` передать в качестве первого параметра значение 0. В программе `ScratchBook` не может быть больше двух окон представления. При включении двух (горизонтальной и вертикальной) вешек разбивки программа с однодокументным (однооконным) интерфейсом может иметь до четырех представлений данного документа. Программа с многодокументным (многооконным) интерфейсом может иметь больше четырех представлений (см. гл. 17).

Изменим программу `ScratchBook` так, чтобы при рисовании линии в одной панели вторая перерисовывала только измененную видимую часть рисунка. Добавьте в определение класса `CLine` функцию `GetDimRect`, возвращающую размеры прямоугольника, который ограничивает нарисованную линию. Этот прямоугольник – часть окна, измененная при рисовании линии. Добавьте в файле `ScratchBookDoc.h` объявление функции в раздел `public` определения класса `CLine`.

```
class CLine : public CObject
{
```



```
protected:
    int m_X1, m_Y1, m_X2, m_Y2;
    CLine ()
    {}
    DECLARE_SERIAL (CLine)

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);
    CRect GetDimRect ();
    virtual void Serialize (CArchive &ar);
};
```

Добавьте приведенное ниже определение функции `GetDimRect` в конце файла `ScratchBookDoc.cpp`. Функция `GetDimRect` возвращает объект класса `CRect` с размерами ограничивающего прямоугольника. Макрокоманды `min` и `max` (определенные в файле `Windows.h`) используются для проверки координат его сторон. Координата левой стороны по горизонтали должна быть меньше правой, координата нижней стороны по вертикали – больше верхней. Прямоугольник интерпретируется как пустой, если хотя бы одно из этих условий не выполнено.

```
CRect CLine::GetDimRect ()
{
    return CRect
        (min (m_X1, m_X2), min (m_Y1, m_Y2),
         max (m_X1, m_X2) + 1, max (m_Y1, m_Y2) + 1);
}
```

Функцию `AddLine` класса документа следует изменить таким образом, чтобы она возвращала указатель на объект класса `CLine`, хранящий новую линию. Измените объявление функции `AddLine` в разделе `public` класса `CScratchBookDoc` в файле `ScratchBookDoc.h`

```
CLine *AddLine (int X1, int Y1, int X2, int Y2);
```

Определение функции `AddLine` в файле `ScratchBookDoc.cpp` измените так, как показано ниже:

```
CLine *CScratchBookDoc::AddLine (int X1, int Y1, int X2, int Y2)
{
    CLine *PLine = new CLine (X1, Y1, X2, Y2);
    m_LineArray.Add (PLine);
    SetModifiedFlag( );
    return PLine;
}
```

В функции `OnLButtonUp` файла `ScratchBookView.cpp` измените вызовы функций `AddLine` и `UpdateAllViews`. Добавленный код сохраняет указатель на объект класса `CLine`, возвращенный функцией `AddLine`, а затем передает его как третий параметр в функцию `UpdateAllViews`.

```
void CScratchBookView::OnLButtonUp(UINT nFlags, CPoint point)
{
    // TODO: Вставьте сюда собственный код обработчика
    // или вызов стандартного
```

```

if (m_Dragging)
{
m_Dragging = 0;
::ReleaseCapture ();
::ClipCursor (NULL);
CClientDC ClientDC (this);
OnPrepareDC (&ClientDC);
ClientDC.DPtoLP (&point);

ClientDC.SetROP2 (R2_NOT);
ClientDC.MoveTo (m_PointOrigin);
ClientDC.LineTo (m_PointOld);
ClientDC.SetROP2 (R2_COPYPEN);
ClientDC.MoveTo (m_PointOrigin);
ClientDC.LineTo (point);

CScratchBookDoc* PDoc = GetDocument();
CLine *PCLine;
PCLine = PDoc -> AddLine (m_PointOrigin.x,
    m_PointOrigin.y, point.x, point.y);

PDoc->UpdateAllViews (this, 0, PCLine);
}

CScrollView::OnLButtonUp(nFlags, point);
}

```

Для функция UpdateAllViews класса CDocument второй и третий параметры необязательны, они передают информацию об изменениях (называемую *рекомендацией*), которые нужно внести в документ. Функция UpdateAllViews вызывает виртуальную функцию OnUpdate для каждого объекта представления, передавая ей значения двух рекомендуемых параметров (lHint и pHint).

```

void UpdateAllViews (CView* pSender, LPARAM lHint = 0L,
    Object* pHint = NULL);

```

Функция OnUpdate класса CScrollView игнорирует эти значения и повторно перерисовывает *все окно представления*. Для увеличения эффективности перерисовки необходимо переопределить функцию OnUpdate. Для переопределения функции откройте вкладку Class View, выберите класс CScratchBookView, откройте окно Properties и в разделе этого окна Overrides выберите в левом столбце функцию OnUpdate, а в правом – добавление функции (единственный пункт в выпадающем списке). В переопределенную функцию введите следующие строки в файле ScratchBookView.cpp.

```

void CScratchBookView::OnUpdate(CView* pSender, LPARAM lHint,
    Object* pHint)
{
    // TODO: Добавьте сюда собственный код или вызов базового
    // класса
    if (pHint !=0)
    {
        CRect InvalidRect = ((CLine *)pHint)->GetDimRect ();
        CClientDC ClientDC (this);
        OnPrepareDC (&ClientDC);
        ClientDC.LPtoDP (&InvalidRect);
        InvalidateRect (&InvalidRect);
    }
}

```

```

else
    CScrollView::OnUpdate (pSender, lHint, pHint);
}

```

Теперь рекомендованная информация будет использоваться для перерисовки измененной области окна представления.

Параметр `pHint` содержит указатель на объект класса `CLine` в случае вызова функции `OnUpdate` функцией `UpdateAllViews`. Однако функция `OnUpdate` также вызывается заданной по умолчанию функцией `OnInitialUpdate`, которая получает управление непосредственно перед *первым* отображением рисунка в окне представления. В этом случае для параметра `pHint` задается значение 0. Поэтому вначале функция `OnUpdate` проверяет значение данного параметра. Если параметр `pHint` содержит указатель на объект класса `CLine` (т.е. отличен от нуля), функция `OnUpdate` вызывает функцию `CLine::GetDimRect` для получения прямоугольника, ограничивающего добавленную линию, и сохраняет его в объекте `InvalidRect` класса `CRect`. Затем эта функция создает объект контекста устройства `ClientDC` и вызывает функцию `CScrollView::OnPrepareDC`, чтобы скорректировать объект для текущей позиции прокрутки рисунка. После этого вызывается функция `CDC::LPtoDP` для преобразования координат в объекте `InvalidRect` из логических в фактические. В заключение объект `InvalidRect` передается в функцию `InvalidateRect` класса `CWnd`, что делает указанную прямоугольную область *недействительной* (т.е. помечает ее как требующую перерисовки) и вызывает функцию `OnDraw` класса представления. Область становится недействительной также вследствие ответа Windows на событие, требующее перерисовки некоторой части рисунка (например, верхнее перекрывающее окно передвинуто).

Передаваемые в функцию `InvalidateRect` координаты – это координаты устройства. Если координаты устройства, переданные в функцию `InvalidateRect`, не принадлежат области окна представления (например, одно окно представления в настоящее время не отображает часть рисунка, измененную в другом окне), часть окна представления не становится недействительной. *Недействительная часть* окна представления называется *областью обновления*. При вызове функции `InvalidateRect` недействительная прямоугольная область добавляется к текущей области обновления. Если в окне представления появляется область обновления ненулевого размера, то после обработки любых сообщений с более высоким приоритетом вызывается функция `OnDraw`, которая выполняет обновление (все видимое поле рисунка становится действительным).

Если значение параметра `pHint` равно 0 (т.е. он не содержит указатель на объект класса `CLine`), то функция `OnUpdate` вызывает свою стандартную реализацию, которая делает недействительным все окно представления и вызывает функцию `OnDraw`. Функция `OnDraw` обновляет весь рисунок, даже если только часть его попадает в недействительную область окна представления.

Перепишем функцию `OnDraw` так, чтобы перерисовывать только область, ставшую недействительной. Необходимо, чтобы функция `OnDraw` вызывала метод `CDC::GetClipBox` для получения размеров недействительной области и функцию `GetDimRect` для получения ограничивающего прямоугольника, после чего вызывала функцию `IntersectRect`, определяющую, попадает ли ограничивающий прямоугольник в область, признанную недействительной. Последняя функция возвращает значение `TRUE`, если два переданных прямоугольника имеют непустое пересечение, т.е. перекрываются. Функция `OnDraw` рисует линию, если ограничивающий линию прямоугольник лежит внутри недействительной области.

```

// Прорисовка CScratchBookView

void CScratchBookView::OnDraw(CDC* pDC)
{
    CScratchBookDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
}

```

```

// TODO: вставьте сюда код прорисовки собственных данных
CSize ScrollSize = GetTotalSize ();
pDC->MoveTo (ScrollSize.cx, 0);
pDC->LineTo (ScrollSize.cx, ScrollSize.cy);
pDC->LineTo (0, ScrollSize.cy);

CRect ClipRect;
CRect DimRect;
CRect IntRect;
CLine *PLine;
pDC->GetClipBox (&ClipRect);

int Index = pDoc->GetNumLines ();
while (Index--)
{
    PLine = pDoc->GetLine (Index);
    DimRect = PLine->GetDimRect ();
    if (IntRect.IntersectRect (DimRect, ClipRect))
        PLine->Draw(pDC);
}
}

```

Не попавшая в действительную область окна представления часть рисунка, при перерисовке отсекается, т.е. игнорируется. В программе ScratchBook можно просто вызвать функцию для перерисовки всех линий (на скорости рисования это не отразится). Однако полнофункциональная программа рисования или программа автоматизированного проектирования обычно строит намного более сложные рисунки, поэтому в подобных программах перерисовка только фигур, которые находятся в действительной области, позволяет сократить время на обновление окна.

Добавленный в этом разделе фрагмент программы позволяет повысить эффективность перерисовки и при рисовании новой линии, и при каком-либо внешнем событии. При рисовании линии в одном окне представления вызывается функция UpdateAllViews объекта документа, которой передается указатель на объект класса CLine, содержащий новую линию. Затем для другого представления вызывается функция OnUpdate, которой передается указатель на объект класса CLine. Функция OnUpdate делает действительной часть окна представления, ограничивающую новую линию, что приводит к вызову функции OnDraw. Последняя выводит *только* линии, попадающие в действительную область (т.е. выводит новую линию и перерисовывает все другие линии в этой области). Если окно представления нуждается в перерисовке из-за внешнего события (например, перемещения перекрывающего окна), то Windows также признает действительной *только* ту область окна, которая нуждается в перерисовке. Затем новая версия функции OnDraw перерисовывает только линии, попавшие в эту область.

## Текст программы ScratchBook

Ниже в листингах 13.1—13.8 приведены исходные тексты программы ScratchBook, рассмотренной в этой главе.

### Листинг 13.1

```

// ScratchBook.h : главный заголовочный файл приложения ScratchBook
//

#pragma once

```

```

#ifndef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"          // основные символы

// CScratchBookApp:
// Смотрите реализацию этого класса в файле ScratchBook.cpp
//

class CScratchBookApp : public CWinApp
{
public:
    CScratchBookApp();

    // Переопределения
public:
    virtual BOOL InitInstance();

    // Реализация
    afx_msg void OnAppAbout();
    DECLARE_MESSAGE_MAP()
};

```

---

## Листинг 13.2

```

// ScratchBook.cpp : Определяет поведение классов приложения
//

#include "stdafx.h"
#include "ScratchBook.h"
#include "MainFrm.h"

#include "ScratchBookDoc.h"
#include "ScratchBookView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// CScratchBookApp

BEGIN_MESSAGE_MAP(CScratchBookApp, CWinApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
    // Стандартные команды работы с файлами документов
    ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
END_MESSAGE_MAP()

// Конструктор CScratchBookApp

CScratchBookApp::CScratchBookApp()
{

```

```

        // TODO: добавьте сюда код конструктора.
        // Поместите весь существенный код инициализации
        // в функцию InitInstance
    }

    // Единственный объект класса CScratchBookApp

    CScratchBookApp theApp;

    // Инициализация CScratchBookApp

    BOOL CScratchBookApp::InitInstance()
    {
        CWinApp::InitInstance();

        // Стандартная инициализация.
        // Если вы не используете какие-то из предоставленных
        // возможностей и хотите уменьшить размер конечного
        // исполняемого модуля, удалите из последующего кода
        // отдельные команды инициализации элементов, которые
        // вам не нужны.
        // Измените ключ, под которым ваши установки хранятся в
        // реестре.
        // TODO: Измените строку параметра на что-нибудь подходящее,
        // например, на имя вашей компании или организации
        SetRegistryKey(_T("Local AppWizard-Generated Applications"));
        LoadStdProfileSettings(4); // Загрузка стандартных установок
                                   // из INI-файла (включая MRU)
        // Регистрация шаблона документа приложения. Шаблоны
        // документов служат связью между документами, окнами
        // документов и окнами приложений
        CSingleDocTemplate* pDocTemplate;
        pDocTemplate = new CSingleDocTemplate(
            IDR_MAINFRAME,
            RUNTIME_CLASS(CScratchBookDoc),
            RUNTIME_CLASS(CMainFrame),           // главное окно
                                                    // SDI-приложения
            RUNTIME_CLASS(CScratchBookView));
        AddDocTemplate(pDocTemplate);

        EnableShellOpen ();
        RegisterShellFileTypes ();

        // Просмотр командной строки для обнаружения стандартных
        // команд оболочки, DDE, открытия файлов
        CCommandLineInfo cmdInfo;
        ParseCommandLine(cmdInfo);
        // Выполнение команд, указанных в командной строке.
        // Вернет FALSE, если приложение было запущено с
        // /RegServer, /Register, /Unregserver или /Unregister.
        if (!ProcessShellCommand(cmdInfo))
            return FALSE;
        // Показ и обновление единственного
        // проинициализированного окна
        m_pMainWnd->ShowWindow(SW_SHOW);
        m_pMainWnd->UpdateWindow();
    }

```

```

        m_pMainWnd->DragAcceptFiles ();

        return TRUE;
    }

    // CAboutDlg диалог, используемый в App About

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

        // Данные для диалога
        enum { IDD = IDD_ABOUTBOX };

    protected:
        virtual void DoDataExchange(CDataExchange* pDX);
        // Поддержка DDX/DDV

        // Реализация
    protected:
        DECLARE_MESSAGE_MAP()
    };

    CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
    {
    }

    void CAboutDlg::DoDataExchange(CDataExchange* pDX)
    {
        CDialog::DoDataExchange(pDX);
    }

    BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    END_MESSAGE_MAP()

    // Команда приложения для выполнения диалога
    void CScratchBookApp::OnAppAbout()
    {
        CAboutDlg aboutDlg;
        aboutDlg.DoModal();
    }

    // Обработчики сообщений класса CScratchBookApp

```

---

### Листинг 13.3

```

// ScratchBookDoc.h : интерфейс класса CScratchBookDoc
//

#pragma once

class CLine : public CObject
{
protected:

```

```

        int m_X1, m_Y1, m_X2, m_Y2;
        CLine ()
        {}
        DECLARE_SERIAL (CLine)

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);
        CRect GetDimRect ();
        virtual void Serialize (CArchive &ar);
};

class CScratchBookDoc : public CDocument
{
protected:
        CTypedPtrArray<CObArray, CLine*> m_LineArray;

public:
        CLine *AddLine (int X1, int Y1, int X2, int Y2);
        CLine *GetLine (int Index);
        int GetNumLines ();

protected: // используется только для сериализации
        CScratchBookDoc();
        DECLARE_DYNCREATE(CScratchBookDoc)

// Атрибуты
public:

// Операции
public:

// Переопределения
        public:
        virtual BOOL OnNewDocument();
        virtual void Serialize(CArchive& ar);

// Реализация
public:
        virtual ~CScratchBookDoc();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif
protected:

// Сгенерированные функции карты сообщений
protected:
        DECLARE_MESSAGE_MAP()

```



```

public:
    virtual void DeleteContents();
    afx_msg void On57633();
    afx_msg void OnUpdate57633(CCmdUI *pCmdUI);
    afx_msg void On57643();
    afx_msg void OnUpdate57643(CCmdUI *pCmdUI);
};

```

---

#### Листинг 13.4

```

// ScratchBookDoc.cpp : реализация класса CScratchBookDoc
//

#include "stdafx.h"
#include "ScratchBook.h"

#include "ScratchBookDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// CScratchBookDoc

IMPLEMENT_DYNCREATE(CScratchBookDoc, CDocument)

BEGIN_MESSAGE_MAP(CScratchBookDoc, CDocument)
    ON_COMMAND(57633, On57633)
    ON_UPDATE_COMMAND_UI(57633, OnUpdate57633)
    ON_COMMAND(57643, On57643)
    ON_UPDATE_COMMAND_UI(57643, OnUpdate57643)
END_MESSAGE_MAP()

// Конструктор/деструктор класса CScratchBookDoc

CScratchBookDoc::CScratchBookDoc()
{
    // TODO: добавьте сюда код одnorазового конструктора
}

CScratchBookDoc::~CScratchBookDoc()
{
}

BOOL CScratchBookDoc::OnNewDocument()
{
    {
        if (!CDocument::OnNewDocument())
            return FALSE;

        // TODO: добавьте сюда код повторной инициализации
        // (SDI-документы будут использовать этот
        // документ многократно)

        return TRUE;
    }
}

```

```

// Сериализация CScratchBookDoc

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

// Диагностика класса CScratchBookDoc

#ifdef _DEBUG
void CScratchBookDoc::AssertValid() const
{
    CDocument::AssertValid();
}

void CScratchBookDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}
#endif //_DEBUG

// Команды класса CScratchBookDoc

IMPLEMENT_SERIAL (CLine, CObject, 1)

void CLine::Draw (CDC *PDC)
{
    PDC->MoveTo (m_X1, m_Y1);
    PDC->LineTo (m_X2, m_Y2);
}

void CLine::Serialize (CArchive &ar)
{
    if (ar.IsStoring())
        ar << m_X1 << m_Y1 << m_X2 << m_Y2;
    else
        ar >> m_X1 >>m_Y1 >> m_X2 >> m_Y2;
}

CLine *CScratchBookDoc::AddLine (int X1, int Y1, int X2, int Y2)
{
    CLine *PLine = new CLine (X1, Y1, X2, Y2);
    m_LineArray.Add (PLine);
    SetModifiedFlag( );
    return PLine;
}

```

```

CLine *CScratchBookDoc::GetLine (int Index)
{
    if (Index < 0 || Index > m_LineArray.GetUpperBound ())
        return 0;
    return m_LineArray.GetAt (Index);
}

int CScratchBookDoc::GetNumLines ()
{
    return m_LineArray.GetSize ();
}

void CScratchBookDoc::DeleteContents()
{
    // TODO: Добавьте сюда собственный код или
    // вызов базового класса
    int Index = m_LineArray.GetSize ();
    while (Index--)
        delete m_LineArray.GetAt (Index);
    m_LineArray.RemoveAll ();

    CDocument::DeleteContents();
}

void CScratchBookDoc::On57633()
{
    // TODO: Добавьте сюда собственный код обработчика
    DeleteContents ();
    UpdateAllViews (0);
    SetModifiedFlag ();
}

void CScratchBookDoc::OnUpdate57633(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->Enable (m_LineArray.GetSize ());
}

void CScratchBookDoc::On57643()
{
    // TODO: Добавьте сюда собственный код обработчика
    int Index = m_LineArray.GetUpperBound ();
    if (Index > -1)
    {
        delete m_LineArray.GetAt (Index);
        m_LineArray.RemoveAt (Index);
    }
    UpdateAllViews (0);
    SetModifiedFlag ();
}

void CScratchBookDoc::OnUpdate57643(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->Enable (m_LineArray.GetSize ());
}

```

```

CRect CLine::GetDimRect ()
{
    return CRect
        (min (m_X1, m_X2), min (m_Y1, m_Y2),
         max (m_X1, m_X2) + 1, max (m_Y1, m_Y2) + 1);
}

```

---

### Листинг 13.5

```

// ScratchBookView.h : интерфейс класса CScratchBookView
//

#pragma once

class CScratchBookView : public CScrollView
{
protected:
    CString m_ClassName;
    int m_Dragging;
    HCURSOR m_HCross;
    HCURSOR m_HArrow;
    CPoint m_PointOld;
    CPoint m_PointOrigin;

protected: // используется только для сериализации
    CScratchBookView();
    DECLARE_DYNCREATE(CScratchBookView)

// Атрибуты
public:
    CScratchBookDoc* GetDocument() const;

// Операции
public:

// Переопределения
    public:
        virtual void OnDraw(CDC* pDC);
        // переопределена для прорисовки этого вида
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
protected:

// Реализация
public:
    virtual ~CScratchBookView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Сгенерированные функции карты сообщений
protected:
    DECLARE_MESSAGE_MAP()

```

```

public:
    afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
    afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
    afx_msg void OnMouseMove(UINT nFlags, CPoint point);
    virtual void OnInitialUpdate();
protected:
    virtual void OnUpdate(CView* /*pSender*/, LPARAM /*lHint*/,
                        CObject* /*pHint*/);
};

#ifdef _DEBUG // отладочная версия в файле ScratchBookView.cpp
inline CScratchBookDoc* CScratchBookView::GetDocument() const
    { return (CScratchBookDoc*)m_pDocument; }
#endif

```

---

### Листинг 13.6

```

// ScratchBookView.cpp : реализация класса CScratchBookView
//

#include "stdafx.h"
#include "ScratchBook.h"

#include "ScratchBookDoc.h"
#include "ScratchBookView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// CScratchBookView

IMPLEMENT_DYNCREATE(CScratchBookView, CScrollView)

BEGIN_MESSAGE_MAP(CScratchBookView, CScrollView)
    ON_WM_LBUTTONDOWN()
    ON_WM_LBUTTONUP()
    /   ON_WM_MOUSEMOVE()
END_MESSAGE_MAP()

// Конструктор/деструктор класса CScratchBookView

CScratchBookView::CScratchBookView()
{
    // TODO: добавьте сюда собственный код конструктора

    m_Dragging = 0;
    m_HCross = AfxGetApp()->LoadStandardCursor (IDC_CROSS);
    m_HArrow = AfxGetApp()->LoadStandardCursor (IDC_ARROW);
}

CScratchBookView::~CScratchBookView()
{
}

```

```

BOOL CScratchBookView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Модифицируйте класс или стили окна,
    // добавляя и изменяя поля структуры cs

    m_ClassName = AfxRegisterWndClass
        (CS_HREDRAW | CS_VREDRAW, // стили окна
        0, // без указателя
        (HBRUSH)::GetStockObject (WHITE_BRUSH), // задать белый фон
        0); // без значка
    cs.lpszClass = m_ClassName;

    return CScrollView::PreCreateWindow(cs);
}

// Прорисовка CScratchBookView

void CScratchBookView::OnDraw(CDC* pDC)
{
    CScratchBookDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    // TODO: вставьте сюда код прорисовки собственных данных
    CSize ScrollSize = GetTotalSize ();
    pDC->MoveTo (ScrollSize.cx, 0);
    pDC->LineTo (ScrollSize.cx, ScrollSize.cy);
    pDC->LineTo (0, ScrollSize.cy);

    CRect ClipRect;
    CRect DimRect;
    CRect IntRect;
    CLine *PLine;
    pDC->GetClipBox (&ClipRect);

    int Index = pDoc->GetNumLines ();
    while (Index--)
    {
        PLine = pDoc->GetLine (Index);
        DimRect = PLine->GetDimRect ();
        if (IntRect.IntersectRect (DimRect, ClipRect))
            PLine->Draw(pDC);
    }
}

// Диагностика класса CScratchBookView

#ifdef _DEBUG
void CScratchBookView::AssertValid() const
{
    CScrollView::AssertValid();
}

void CScratchBookView::Dump(CDumpContext& dc) const
{
    CScrollView::Dump(dc);
}

```

```

CScratchBookDoc* CScratchBookView::GetDocument() const
// не отладочная версия встроена
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CScratchBookDoc)));
    return (CScratchBookDoc*)m_pDocument;
}
#endif // _DEBUG

// Обработчики сообщений класса CScratchBookView

void CScratchBookView::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: Вставьте сюда собственный код обработчика
    // или вызов стандартного
    CClientDC ClientDC (this);
    OnPrepareDC (&ClientDC);
    ClientDC.DPtoLP (&point);

    // проверка нахождения курсора внутри области
    // окна представления
    CSize ScrollSize = GetTotalSize ();
    CRect ScrollRect (0, 0, ScrollSize.cx, ScrollSize.cy);
    if (!ScrollRect.PtInRect (point))
        return;

    // сохранение позиции курсора, захват мыши и
    // установка флага перемещения
    m_PointOrigin = point;
    m_PointOld = point;
    SetCapture ();
    m_Dragging = 1;

    // ограничение перемещений курсора мыши
    ClientDC.LPtoDP (&ScrollRect);
    CRect ViewRect;
    GetClientRect (&ViewRect);
    CRect IntRect;
    IntRect.IntersectRect (&ScrollRect, &ViewRect);
    ClientToScreen (&IntRect);
    ::ClipCursor (&IntRect);

    CScrollView::OnLButtonDown(nFlags, point);
}

void CScratchBookView::OnLButtonUp(UINT nFlags, CPoint point)
{
    // TODO: Вставьте сюда собственный код обработчика
    // или вызов стандартного
    if (m_Dragging)
    {
        m_Dragging = 0;
        ::ReleaseCapture ();
        ::ClipCursor (NULL);
        CClientDC ClientDC (this);
        OnPrepareDC (&ClientDC);
    }
}

```

```

ClientDC.DPtoLP (&point);

ClientDC.SetROP2 (R2_NOT);
ClientDC.MoveTo (m_PointOrigin);
ClientDC.LineTo (m_PointOld);
ClientDC.SetROP2 (R2_COPYPEN);
ClientDC.MoveTo (m_PointOrigin);
ClientDC.LineTo (point);

CScratchBookDoc* PDoc = GetDocument();
CLine *PCLine;
PCLine = PDoc -> AddLine (m_PointOrigin.x,
                          m_PointOrigin.y, point.x, point.y);

PDoc->UpdateAllViews (this, 0, PCLine);
}

CScrollView::OnLButtonUp(nFlags, point);
}

void CScratchBookView::OnMouseMove(UINT nFlags, CPoint point)
{
    // TODO: Вставьте сюда собственный код обработчика
    // или вызов стандартного

    CClientDC ClientDC (this);
    OnPrepareDC (&ClientDC);
    ClientDC.DPtoLP (&point);

    CSize ScrollSize = GetTotalSize();
    CRect ScrollRect(0, 0, ScrollSize.cx, ScrollSize.cy);
    if (ScrollRect.PtInRect (point))
        ::SetCursor (m_HCross);
    else
        ::SetCursor (m_HArrow);

    if (m_Dragging)
    {
        ClientDC.SetROP2 (R2_NOT);
        ClientDC.MoveTo (m_PointOrigin);
        ClientDC.LineTo (m_PointOld);
        ClientDC.MoveTo (m_PointOrigin);
        ClientDC.LineTo (point);
        m_PointOld = point;
    }

    CScrollView::OnMouseMove(nFlags, point);
}

void CScratchBookView::OnInitialUpdate()
{
    CScrollView::OnInitialUpdate();

    // TODO: Добавьте сюда собственный код или
    // вызов базового класса

```



```

        SIZE Size = {800, 600};
        SetScrollSizes (MM_TEXT, Size);
    }

void CScratchBookView::OnUpdate(CView* pSender, LPARAM lHint,
CObject* pHint)
{
    // TODO: Добавьте сюда собственный код или
    // вызов базового класса
    if (pHint !=0)
    {
        CRect InvalidRect = ((CLine *)pHint)->GetDimRect ();
        CClientDC ClientDC (this);
        OnPrepareDC (&ClientDC);
        ClientDC.LPtoDP (&InvalidRect);
        InvalidateRect (&InvalidRect);
    }
    else
        CScrollView::OnUpdate (pSender, lHint, pHint);
}

```

---

### Листинг 13.7

```

// MainFrm.h : интерфейс класса CMainFrame
//

#pragma once
class CMainFrame : public CFrameWnd
{
protected:
    CSplitterWnd m_SplitterWnd;

protected: // используется только для сериализации
    CMainFrame();
    DECLARE_DYNCREATE(CMainFrame)

// Атрибуты
public:

// Операции
public:

// Переопределения
public:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);

// Реализация
public:
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif
protected: // встроенные элементы управления
    CStatusBar m_wndStatusBar;

```

```

        CToolBar    m_wndToolBar;

// Сгенерированные функции карты сообщений
protected:
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    DECLARE_MESSAGE_MAP()
    virtual BOOL OnCreateClient(LPCREATESTRUCT lpcs, CCreateContext*
pContext);
};

```

---

### Листинг 13.8

```

// MainFrm.cpp : реализация класса CMainFrame
//

#include "stdafx.h"
#include "ScratchBook.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
    ON_WM_CREATE()
END_MESSAGE_MAP()

static UINT indicators[] =
{
    ID_SEPARATOR,           // индикатор строки состояния
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
};

// Конструктор/деструктор класса CMainFrame

CMainFrame::CMainFrame()
{
    // TODO: добавьте сюда код инициализации элементов
}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;
}

```

```

        if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD |
            WS_VISIBLE | CBRS_TOP | CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY |
            CBRS_SIZE_DYNAMIC) || !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
        {
            TRACE0("Failed to create toolbar\n");
            return -1;    // не удалось создать панель
                        // инструментов
        }

        if (!m_wndStatusBar.Create(this) ||
            !m_wndStatusBar.SetIndicators(indicators,
            sizeof(indicators)/sizeof(UINT)))
        {
            TRACE0("Failed to create status bar\n");
            return -1;    // не удалось создать строку состояния
        }
        // TODO: Удалите три следующие строки, если не хотите,
        // чтобы панель инструментов была паркующей
        m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
        EnableDocking(CBRS_ALIGN_ANY);
        DockControlBar(&m_wndToolBar);

        return 0;
    }

    BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
    {
        if( !CFrameWnd::PreCreateWindow(cs) )
            return FALSE;
        // TODO: Модифицируйте стили или классы окна здесь,
        // добавляя или изменяя поля структуры cs

        return TRUE;
    }

    // Диагностика класса CMainFrame

#ifdef _DEBUG
    void CMainFrame::AssertValid() const
    {
        CFrameWnd::AssertValid();
    }

    void CMainFrame::Dump(CDumpContext& dc) const
    {
        CFrameWnd::Dump(dc);
    }

#endif // _DEBUG

    // Обработчики сообщений класса CMainFrame

    BOOL CMainFrame::OnCreateClient(LPCREATESTRUCT lpcs,
                                    CCreateContext* pContext)
    {
        // TODO: Добавьте сюда собственный код или вызов базового

```

```
// класса.
return m_SplitterWnd.Create
(
    this,                // родительское окно разделенного окна;
    1,                   // максимальное число строк;
    2,                   // максимальное число столбцов;
    CSize (20, 20),      // минимальный размер окна представления;
    pContext);           // информация о контексте устройства
}
```

## Резюме

На примере простого графического редактора ScratchBook мы рассмотрели принципы прокрутки и разделения окон представления.

- *Полосы прокрутки.* Чтобы снабдить окно представления горизонтальными и вертикальными полосами прокрутки, следует порождать класс представления не от MFC-класса CView, а от класса CScrollView. При этом MFC выполняет большую часть логических операций для прокрутки документа, отображаемого в окне представления. При прокрутке документа MFC корректирует *начало области просмотра*, которое в окне представления определяет положение текста или графики. Начало области просмотра для объекта контекста устройства, передаваемого в функцию OnDraw, уже скорректировано, вследствие чего текст или графика автоматически выводится в окне. Для поддержки прокрутки *не* нужно изменять функцию OnDraw. Для коррекции начала области просмотра прокрученного документа объект контекста устройства передается в функцию CScrollView::OnPrepareDC.
- *Размер документа.* Функция SetScrollSizes вызывается из переопределенной версии функции OnInitialUpdate, заданной как член класса представления, и позволяет определить размер прокручиваемого документа. Функция OnInitialUpdate вызывается непосредственно перед *первым* отображением рисунка (нового или уже существующего) в окне представления. Для документов фиксированного размера необходимо блокировать попытки добавления текста или графики вне границ документа.
- *Логические и фактические координаты.* Указанные при выводе объекта координаты, называются *логическими*. Фактические координаты объекта относительно окна называются *координатами устройства*. Если документ прокручен в окне, то логические координаты отличаются от фактических. Функция CDC::DPtoLP преобразовывает фактические координаты в логические, а функция CDC::LPtoDP осуществляет обратное преобразование. MFC иногда использует логические координаты (например, при вызове функции рисования), а иногда – координаты устройства (например, при передаче обработчику сообщения координат указателя мыши).
- *Разделение окна.* Для разделения окна программы на отдельные окна представления (или *панели*) используются горизонтальная и вертикальная линии разбивки. Для программной реализации средств разделения окон в класс главного окна добавляется переменная, являющаяся объектом MFC-класса CSplitterWnd и переопределяется виртуальная функция OnCreateClient, вызываемая при первичном создании главного окна. Для *разделения* окна следует из функции OnCreateClient вызвать функцию Create для объекта класса CSplitterWnd, который автоматически создает одно или несколько окон представления. При создании *новой* программы с помощью мастера Application Wizard можно добавить возможность разделения окна представления, выбрав соответствующие опции.
- *Прорисовка разделенного окна.* При редактировании данных в одной панели данные в другой панели должны быть модифицированы. Объект представления вызывает функцию Document::UpdateAllViews для объекта документа, передавая ей информацию в виде *рекомендаций*,

описывающих изменения. Функция `UpdateAllViews` вызывает виртуальную функцию `OnUpdate` для каждого объекта представления, передавая ей рекомендации. Чтобы сделать недействительной измененную область окна представления, переопределите виртуальную функцию, которая вызывает метод `CWnd::InvalidateRect`. Функция `OnUpdate` определяет размеры недействительной области на основании рекомендуемой информации. Функция `InvalidateRect` вызывает функцию `OnDraw` класса представления, которая, в свою очередь, вызывает функцию `GetClipBox`, чтобы получить размеры недействительной области и перерисовать только находящиеся в этой области текст и графику.

## Глава 14

### Панель инструментов и строка состояния

---

- Перемещаемая панель инструментов в программе ScratchBook
- Строка состояния в программе ScratchBook
- Текст программы ScratchBook

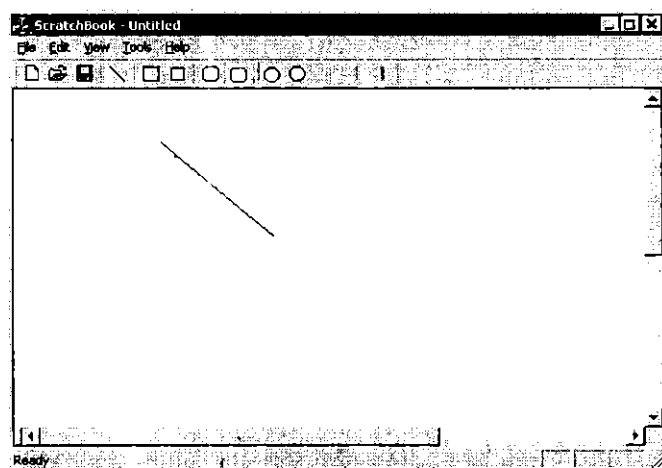
В этой главе рассматриваются сложные элементы интерфейса, поддерживаемые библиотекой MFC:

- *Перемещаемая панель инструментов* состоит из совокупности кнопок. Щелчок на любой из них приводит к немедленному выполнению команды или установке/сбросу опции. Панели инструментов можно “закреплять” (припарковывать) на любом краю окна программы или перемещать в окне.
- *Строка состояния* обычно отображается внизу главного окна и используется для вывода сообщений, отображения состояний клавиш или указания режимов программы (например, состояние клавиши Caps Lock или ход сохранения файла).
- *Диалоговая панель* напоминает панель инструментов, но основана на шаблоне диалогового окна, включает элементы управления, отличные от кнопок (например, списки).
- *Переключаемая панель* (rebar control) является контейнером для переупорядочиваемых панелей инструментов и других элементов управления.

Панели инструментов, строки состояния, диалоговые панели и переключаемые панели принадлежат общей категории элементов, называемых *панелями управления*. Библиотека MFC предоставляет отдельные классы, которые порождаются от класса `CControlBar`, для манипулирования панелями управления любого указанного типа. Соответствующая установка опций при генерации программы с помощью Application Wizard позволяет добавлять в состав приложений панель инструментов со стандартными командами работы с файлами и строку состояния с индикатором статуса программы и индикаторами состояний клавиатуры (см. гл. 9).

#### Перемещаемая панель инструментов в программе ScratchBook

---



При создании приложения, если выбраны соответствующие установки, будет автоматически создана стандартная панель управления, содержащая кнопки для работы с файлами. В стандартную панель инструментов, отредактированную в предыдущих главах (убраны некоторые кнопки), показанную на рисунке, добавлены десять кнопок для выбора инструментов рисования и толщины рисуемых линий. Первые семь предназначены для выбора инструментов рисования. Последние три – для задания толщины линии при рисовании. Одновременно можно выбирать по одной кнопке в каждой группе – одну в первых семи кнопках и одну в последних трех. Код для рисования фигур или использования линий различной толщины будет добавлен в программу только в гл. 19. В этой главе в программе выводятся простые линии независимо от выбранного инструмента. Мы добавим также команды меню, соответствующие кнопкам, чтобы можно было выбирать инструмент рисования или задавать толщину линии, используя команду меню, а не только кнопку панели инструментов.

## Модификация ресурсов

Откройте проект ScratchBook, а затем – вкладку Resource View, чтобы отобразить список ресурсов программы для его корректировки. Чтобы добавить кнопки в панель инструментов программы ScratchBook, откройте раздел Toolbar на вкладке Resource View. Двойным щелчком на значке IDR\_MAINFRAME в папке откройте диалоговое окно Properties. Visual Studio отобразит панель инструментов внутри окна редактора панелей инструментов. Это окно содержит три области:

- *Область панели инструментов*, размещенная вверху, отображает всю панель инструментов. Чтобы *выбрать* кнопку, щелкните на ней внутри этой области. Чтобы *удалить* кнопку, перетащите ее с помощью мыши за границы рисунка панели инструментов.
- *Область предварительного просмотра* (внизу слева) отображает выбранную кнопку в ее действительных размерах.
- *Область редактирования* (внизу справа) отображает увеличенное изображение кнопки. Значок (изображение) кнопки редактируется внутри этой области. Пока окно редактора панелей инструментов активно, Visual Studio отображает панели инструментов Graphics и Colors, используемые для создания или редактирования изображений кнопок.

Процедура конструирования кнопки панели инструментов не отличается сложностью. Создайте рисунок для первой кнопки в области редактирования, используя мышь и панели инструментов Graphics и Colors. Можно создать собственные изображения, но при этом не следует забывать о функциях, соответствующих кнопкам. Информация об использовании команд редактора панелей инструментов содержится в справочной системе. Щелкните правой кнопкой мыши на созданной кнопке в области панели инструментов и в контекстном меню выберите команду Properties. В открывшемся окне назначьте кнопке идентификатор ID\_TOOLS\_LINE. Вводить текст интерактивной справки для команды в поле Prompt не нужно, это можно будет сделать при определении команды меню с таким же идентификатором (ID\_TOOLS\_LINE). Сразу после начала редактирования первой кнопки справа от нее появляется новая пустая кнопка. Щелкните на ней внутри области панели инструментов, и она отобразится в двух других областях.

Для оставшихся кнопок повторите описанную процедуру. Необходимо создать десять кнопок, их идентификаторы перечислены ниже:

```
ID_TOOLS_LINE
ID_TOOLS_RECTANGLE
ID_TOOLS_FRECTANGLE
ID_TOOLS_RRECTANGLE
ID_TOOLS_FRRECTANGLE
ID_TOOLS_CIRCLE
ID_TOOLS_FCIRCLE
ID_LINE_SINGLE
```

ID\_LINE\_DOUBLE  
ID\_LINE\_TRIPLE

В панели инструментов следует создать промежутки после первой, третьей, пятой и седьмой кнопок. Чтобы создать такой промежуток, внутри области панели инструментов с помощью мыши перетащите вправо кнопку, следующую за создаваемым промежутком. Промежутки будут преобразованы в разделители при отображении панели инструментов. Крайняя справа кнопка предназначена для создания новых кнопок и не является частью панели инструментов.

## Модификация меню

Откройте окно редактора меню для меню IDR\_MAINFRAME, чтобы добавить команды, соответствующих кнопкам панели инструментов. Справа от существующего меню Edit добавьте меню Tools и поместите в него пункты, показанные в табл. 14.1. Для каждой команды меню в таблице приведена строка, добавляемая в поле Prompt диалогового окна Properties:

- часть строки *перед* символом перевода строки (\n) – это *интерактивная справка*;
- часть строки *после* символа перевода строки – *всплывающая подсказка*.

Табл. 14.1. Свойства пунктов меню Tools

Идентификатор (ID)	Надпись (Caption)	Интерактивная справка (Prompt)	Другие свойства
	&Tools		Popup (Ниспадающее меню)
ID_TOOLS_LINE	&Line	Select tool to draw straight lines\nLine (Выбор инструмента для рисования прямых линий)	
ID_TOOLS_RECTANGLE	&Rectangle	Select tool to draw open rectangles\nRectangle (Выбор инструмента для рисования не залитых прямоугольников)	
ID_TOOLS_FRECTANGLE	&Filled Rectangle	Select tool to draw filled rectangles\nFilled Rectangle (Выбор инструмента для рисования залитых прямоугольников)	
ID_TOOLS_RRECTANGLE	R&ounded Rectangle	Select tool to draw open rectangles with rounded corners\nRounded Rectangle (Выбор инструмента для рисования не залитых прямоугольников с закругленными углами)	
ID_TOOLS_FRRECTANGLE	F&illed Rounded Rectangle	Select tool to draw filled rectangles with rounded corners\nFilled Rounded Rectangle (Выбор инструмента для рисования залитых прямоугольников с закругленными углами)	
ID_TOOLS_CIRCLE	&Circle	Select tool to draw open circles or ellipses\nCircle (Выбор инструмента для рисования не заливой окружности)	
ID_TOOLS_FCIRCLE	Fill&ed Circle	Select tool to draw filled circles or ellipses\nFilled Circle (Выбор инструмента для рисования заливой окружности)	



Интерактивную справку или всплывающую подсказку команды меню можно модифицировать. Для этого откройте меню в редакторе меню, выполнив двойной щелчок на команде, и отредактируйте текст в поле Prompt. Чтобы изменить интерактивную справку или всплывающую подсказку для кнопки панели инструментов:

1. Откройте панель инструментов в редакторе панели инструментов.
2. Выберите кнопку (щелкнув на ней в области панели инструментов).
3. Выполните двойной щелчок в области редактирования.
4. Отредактируйте текст в поле Prompt.

Можно также отредактировать текст в редакторе строк (см. параграф “Добавление команд в меню File” гл. 12). Если ID\_MYCOMMAND – идентификатор кнопки панели инструментов или команды меню, то в редакторе строк нужно открыть ресурс String Table. Отредактируйте строку ID\_MYCOMMAND или добавьте строку с этим идентификатором, если он еще не существует.

Ниже последней команды меню добавьте подменю Lines с пунктами, описанными в табл. 14.2. Команды в этом меню позволяют выбирать толщину линии и соответствуют последним трем кнопкам панели инструментов.

Табл. 14.2. Свойства пунктов подменю Lines

Идентификатор (ID)	Надпись (Caption)	Интерактивная справка (Prompt)	Другие свойства
	Lines		Рорир (Ниспадающее меню)
ID_LINE_SINGLE	Single	Draw using single-width lines\nSingle-width Lines (Рисование линиями обычной толщины)	
ID_LINE_DOUBLE	Double	Draw using double-width lines\nDouble-width Lines (Рисование линиями двойной толщины)	
ID_LINE_TRIPLE	Triple	Draw using triple-width lines\nTriple-width Lines (Рисование линиями тройной толщины)	

Для сохранения результатов работы выберите в меню File команду Save All или щелкните в панели инструментов Standard на кнопке Save All.

## Модификация текста программы

Доработаем фрагмент программы ScratchBook, ответственный за создание панели инструментов. Переменная m\_wndToolBar определена в разделе protected класса CMainFrame в файле заголовков MainFrm.h. Переменная m\_wndToolBar – это объект класса CToolBar, являющегося MFC-классом управления панелями инструментов. Класс CToolBar порожден от класса CControlBar, как и классы для манипулирования панелями управления различных типов, рассматриваемые в этой главе.

```
protected: // встроенные элементы управления
    CStatusBar m_wndStatusBar;
    CToolBar m_wndToolBar;
```

Внутри класса главного окна обработчик сообщения WM\_CREATE, передаваемого в момент создания окна непосредственно перед тем, как оно становится видимым, создает панель инструментов. Его текст в файле Mainfrm.cpp приведен ниже.

```

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT,
        WS_CHILD | WS_VISIBLE | CBRS_TOP | CBRS_GRIPPER |
        CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC) ||
        !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
    {
        TRACE0("Failed to create toolbar\n");
        return -1;        // не удалось создать панель инструментов
    }
}

```

Используемая функция `CToolBar::CreateEx` создает панель инструментов и задает ее стили. Первый параметр `this` указывает, что главное окно является родительским для панели инструментов (дочернего окна). Второй параметр `CreateEx` задает стили панели инструментов. При передаче параметра `TBSTYLE_FLAT` создается панель инструментов с плоскими кнопками. Описание других стилей кнопок приведено в справочной системе. Третий параметр `CreateEx` задает общие стили для панели управления:

- константа `WS_CHILD` делает панель инструментов дочерним окном главного окна;
- константа `WS_VISIBLE` – делает окно (панель инструментов) видимым;
- константа `CBRS_TOP` приводит окно (панель инструментов) к первоначальному размещению;
- константа `CBRS_GRIPPER` служит для отображения маркера перемещения (широкой вертикальной полосы) на левом верхнем краю панели инструментов, который используется для перетаскивания панели инструментов (можно перетащить ее за одну из тонких разделительных линий, но пользоваться маркером удобнее);
- константа `CBRS_TOOLTIPS` активирует режим отображения всплывающих подсказок;
- константа `CBRS_FLYBY` обеспечивает отображение интерактивной справки в строке состояния, когда указатель мыши находится над кнопкой. Если стиль `CBRS_FLYBY` не задан, то программа отображает интерактивную справку при нажатой кнопке мыши, когда указатель находится на кнопке;
- константа `CBRS_SIZE_DYNAMIC` позволяет изменять форму панели инструментов и переупорядочивать ее кнопки, если панель является перемещаемой (но *не* припаркована);
- информация о других стилях, задаваемых в третьем параметре функции `CreateEx` (константы семейства `CBRS_`), содержится в справочной системе.

Ресурс панели инструментов, созданный с помощью редактора ресурсов загружается функцией `CToolBar::LoadToolBar`. Если одна из функций `CreateEx` или `LoadToolBar` возвращает значение 0, указывающее на ошибку, то функция `OnCreate` возвращает значение -1, также означающее, что произошла ошибка. Это приводит к завершению программы с удалением главного окна.

Последний приведенный фрагмент кода, также взятый из файла `Mainfrm.cpp`, позволяет панели инструментов перемещаться. Первое обращение к функции `EnableDocking` вызывает саму функцию, определенную в классе `CControlBar`, и разрешает прикрепление (парковку) панели инструментов. Второе обращение к функции `EnableDocking` вызывает эту функцию, определенную в классе `CFrameWnd`, и разрешает перемещение панели инструментов в главном окне. Передача значения `CBRS_ALIGN_ANY` в обоих вызовах позволяет пользователю прикрепить панель инструментов к *любому* краю окна. Функция `CFrameWnd::DockControlBar` возвращает панель инструментов в исходное положение (прикрепление к верхней границе рабочей области окна у левого края). Если

пренебречь любой из этих функций, то панель инструментов станет непереключаемой, зафиксированной в верхней части окна стандартной панелью.

```
// TODO: Удалите три следующие строки, если не хотите,  
// чтобы панель инструментов была переключаемой  
m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);  
EnableDocking(CBRS_ALIGN_ANY);  
DockControlBar(&m_wndToolBar);
```

При выполнении программы ScratchBook отображается панель инструментов и ее можно перемещать, однако кнопки при этом заблокированы (т.е. затенены серым цветом и не реагируют на щелчки мыши), потому что обработчики сообщений кнопок не определены. По этой же причине заблокированы соответствующие пункты меню. Панель инструментов можно скрывать или отображать командой **ToolBar** в меню **View**. Обработчики сообщений для этой команды предоставляются библиотекой MFC, поэтому она доступна в меню.

При создании перемещаемой панели инструментов для новой программы мастер Application Wizard определит объект класса `CToolBar` и добавит необходимые вызовы в функцию `OnCreate`. Для изменения стилей, указанных при обращении к функции `CToolBar::CreateEx` (т.е. для создания стандартной панели инструментов), измените параметры, передаваемые функции `EnableDocking`, или уберите обращения к функциям `EnableDocking` и `DockControlBar`.

## Обработчики сообщений кнопок и команд

Чтобы определить обработчики сообщений для кнопок панели инструментов и соответствующих команд меню в окне редактора панелей инструментов щелкните на кнопке создаваемой панели, ответственной за режим рисования линии, правой кнопкой мыши. Из открывшегося контекстного меню выберите команду **Add Event Handler...** (Добавить обработчик события). В появившемся окне мастера Event Handler Wizard выберите класс `CScratchBookApp` в поле **Class List**, чтобы класс приложения обрабатывал сообщения от кнопок панелей инструментов и соответствующих команд меню. Класс приложения выбран потому, что выбор текущего инструмента рисования и толщины линии воздействует не на определенный документ или представление, а на работу приложения в целом.

Затем в списке **Message type** выберите сообщение `COMMAND`, щелкните на кнопке **Add and Edit**, приняв заданное по умолчанию имя функции. Так как идентификатор кнопки является и идентификатором команды меню **Tools**, то создаваемая функция получает управление:

- либо после щелчка на кнопке;
- либо после выбора команды меню.

Для создания обработчика сообщения `UPDATE_COMMAND_UI` повторите описанную процедуру, но в поле **Message type** выберите сообщение с именем `UPDATE_COMMAND_UI`. Функция `OnUpdate` получает управление при открытии меню **Lines** для инициализации команды меню (см. гл. 11), а также через равные промежутки времени при простое системы для обновления кнопки.

Воспользуйтесь мастером Event Handler Wizard, чтобы сгенерировать обработчики сообщений для остальных кнопок панели инструментов и соответствующих им команд. В табл. 14.3 описаны все необходимые обработчики сообщений. Во всех случаях необходимо принимать стандартное имя функции, предлагаемое мастером.

Для некоторых обработчиков сообщений необходимо определить и инициализировать две переменных класса приложения:

- `m_CurrentWidth`, которая хранит текущую толщину линии (1, 2 или 3);
- `m_CurrentTool`, которая хранит идентификатор выбранной кнопки с обозначением инструмента рисования.

Табл. 14.3. Обработчики сообщений для кнопок панели инструментов программы ScratchBook

Идентификатор кнопки/команды меню	Идентификатор сообщения	Обработчик сообщения
ID_LINE_DOUBLE	COMMAND	OnLineDouble
	UPDATE_COMMAND_UI	OnUpdateLineDouble
ID_LINE_SINGLE	COMMAND	OnLineSingle
	UPDATE_COMMAND_UI	OnUpdateLineSingle
ID_LINE_TRIPLE	COMMAND	OnLineTriple
	UPDATE_COMMAND_UI	OnUpdateLineTriple
ID_TOOLS_CIRCLE	COMMAND	OnToolsCircle
	UPDATE_COMMAND_UI	OnUpdateToolsCircle
ID_TOOLS_FCIRCLE	COMMAND	OnToolsFCircle
	UPDATE_COMMAND_UI	OnUpdateToolsFCircle
ID_TOOLS_LINE	COMMAND	OnToolsLine
	UPDATE_COMMAND_UI	OnUpdateToolsLine
ID_TOOLS_RECTANGLE	COMMAND	OnToolsRectangle
	UPDATE_COMMAND_UI	OnUpdateToolsRectangle
ID_TOOLS_FRECTANGLE	COMMAND	OnToolsFRectangle
	UPDATE_COMMAND_UI	OnUpdateToolsFRectangle
ID_TOOLS_RRECTANGLE	COMMAND	OnToolsRRectangle
	UPDATE_COMMAND_UI	OnUpdateToolsRRectangle
ID_TOOLS_FRRECTANGLE	COMMAND	OnToolsFRRectangle
	UPDATE_COMMAND_UI	OnUpdateToolsFRRectangle

Добавьте следующие объявления в определение класса CScratchBookApp в файле ScratchBook.h.

```
// CScratchBookApp:
// Смотрите реализацию этого класса в файле ScratchBook.cpp
//
```

```
class CScratchBookApp : public CWinApp
{
public:
    int m_CurrentWidth;
    UINT m_CurrentTool;
```

Следующий код инициализации добавьте в конструктор класса CScratchBookApp в файле ScratchBook.cpp:

```
// Конструктор CScratchBookApp

CScratchBookApp::CScratchBookApp()
{
    // TODO: добавьте сюда код конструктора
    // Поместите весь существенный код инициализации
    // в функцию InitInstance

    m_CurrentWidth = 1;
    m_CurrentTool=ID_TOOLS_LINE;
}
```

В версии программы ScratchBook, описанной в гл. 19, используется переменная `m_CurrentTool`, определяющая, какой тип фигуры нужно генерировать, когда пользователь выполняет операцию рисования и переменная `m_CurrentWidth` для определения толщины линии, используемой при создании фигуры. Добавленный код инициализации заставляет программу выводить простые линии толщиной в один пиксель до тех пор, пока не будет выбрана другая толщина линии или другой инструмент рисования.

В определении обработчиков сообщений, сгенерированные мастером Event Handler Wizard в файле `ScratchBook.cpp`, добавим необходимый код:

```
// Обработчики сообщений класса CScratchBookApp

void CScratchBookApp::OnToolsLine()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentTool = ID_TOOLS_LINE;
}

void CScratchBookApp::OnToolsRectangle()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentTool = ID_TOOLS_RECTANGLE;
}

void CScratchBookApp::OnUpdateToolsRectangle(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck
        (m_CurrentTool == ID_TOOLS_RECTANGLE ? 1 : 0);
}

void CScratchBookApp::OnToolsFrextangle()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentTool = ID_TOOLS_FREXTANGLE;
}

void CScratchBookApp::OnUpdateToolsFrextangle(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck
        (m_CurrentTool == ID_TOOLS_FREXTANGLE ? 1 : 0);
}

void CScratchBookApp::OnToolsRrectangle()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentTool = ID_TOOLS_RRECTANGLE;
}

void CScratchBookApp::OnUpdateToolsRrectangle(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck
        (m_CurrentTool == ID_TOOLS_RRECTANGLE ? 1 : 0);
}
```

```

void CScratchBookApp::OnToolsFrrectangle()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentTool = ID_TOOLS_FRRECTANGLE;
}

void CScratchBookApp::OnUpdateToolsFrrectangle(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck
        (m_CurrentTool == ID_TOOLS_FRRECTANGLE ? 1 : 0);
}

void CScratchBookApp::OnToolsCircle()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentTool = ID_TOOLS_CIRCLE;
}

void CScratchBookApp::OnUpdateToolsCircle(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck
        (m_CurrentTool == ID_TOOLS_CIRCLE ? 1 : 0);
}

void CScratchBookApp::OnToolsFcircle()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentTool = ID_TOOLS_FCIRCLE;
}

void CScratchBookApp::OnUpdateToolsFcircle(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck
        (m_CurrentTool == ID_TOOLS_FCIRCLE ? 1 : 0);
}

void CScratchBookApp::OnLineSingle()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentWidth = 1;
}

void CScratchBookApp::OnUpdateLineSingle(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck(m_CurrentWidth == 1 ? 1 : 0);
}

void CScratchBookApp::OnLineDouble()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentWidth = 2;
}

```

```

void CScratchBookApp::OnUpdateLineDouble(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck(m_CurrentWidth == 2 ? 1 : 0);
}

void CScratchBookApp::OnLineTriple()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentWidth = 3;
}

void CScratchBookApp::OnUpdateLineTriple(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck(m_CurrentWidth == 3 ? 1 : 0);
}

void CScratchBookApp::OnUpdateToolsLine(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck(m_CurrentTool == ID_TOOLS_LINE ? 1 : 0);
}

```

Кратко рассмотрим использование функций класса CCmdUI для обновления команд меню и кнопок панели инструментов. Объект класса CCmdUI передается обработчику сообщения UPDATE\_COMMAND\_UI (т.е. функции OnUpdate...) для объектов интерфейса любого типа: команд меню, сочетаний клавиш, кнопок панели инструментов, полей строки состояния и элементов управления диалоговых окон. Функции Enable, SetCheck, SetRadio и SetText класса CCmdUI используются для обновления объектов пользовательского интерфейса (действие функции зависит от типа объекта) любого вида:

- *Функция Enable.* Если в функцию Enable передается значение False, кнопки панели инструментов или команды меню становятся недоступными. Кнопка или команда меню вернется в нормальное состояние, если в функцию Enable передать значение True.
- *Функция SetCheck.* Если передать значение 1 в функцию SetCheck, то выбирается кнопка панели инструментов или команда меню, а если значение 0, то отменяется выбор кнопки панели инструментов или с команды меню снимается метка выбора. Кроме того, если в функцию SetCheck передать значение 2, то кнопка панели инструментов перейдет в *неопределенное* состояние (или будет просто отмечена команда меню).
- *Функция SetRadio.* Если передать значение TRUE в функцию SetRadio, она отметит команду меню, используя метку выбора, или выберет кнопку на панели инструментов (как функция SetCheck). Если передать значение FALSE, то функция отменяет выбор кнопки или снимает метку с команды меню. Предполагается, что команда (кнопка) входит в группу позиций переключателя.
- *Функция SetText.* Эта функция используется, чтобы задать надпись для команды меню. На кнопку панели инструментов она не воздействует.

Работу приведенных обработчиков сообщений рассмотрим на примере OnLineDouble – обработчика, который получает управление всякий раз при щелчке на кнопке Double-Width Lines или выборе команды Double в меню Lines. В обоих случаях эта функция устанавливает значение переменной m\_CurrentWidth равным 2, после чего программа должна выбрать кнопку Double-Width

Lines и отметить команду меню Double, чтобы можно было увидеть, какая толщина линии выбрана. Эти действия выполняются функцией OnUpdateLineDouble, вызываемой при простое программы и открытии ниспадающего меню Lines. В обоих случаях этой функции передается указатель на объект класса CCmdUI, и она вызывает функцию SetCheck класса CCmdUI, передавая ей отличное от нуля значение (1, так как переменной m\_CurrentWidth присвоено значение 2). Если функция OnUpdateLineDouble вызывается во время простоя программы, то объект CCmdUI связан с кнопкой панели инструментов и при вызове функции SetCheck выбирается эта кнопка. Если OnUpdateLineDouble вызвана в ответ на открытие меню Lines, значит объект класса CCmdUI связан с командой меню и функция SetCheck отмечает эту команду. Таким образом, функция SetCheck выполняет определенное действие в зависимости от типа объекта пользовательского интерфейса.

Если текущая ширина линии равна 2 (как в случае, описанном в предыдущем абзаце), то функции OnUpdateLineSingle и OnUpdateLineTriple будучи вызванными при простое программы и открытии меню Lines, передадут значение 0 в функцию SetCheck, *отменяя выбор* “своих” кнопок (если какая-то из них не выбрана) или *убирая отметку* возле “своей” команды меню (если она отмечена). Таким образом, метка перемещается от пункта, выбранного ранее, к пункту, выбранному в настоящий момент.

Выбирая определенную толщину линии с помощью панели инструментов или меню Lines, пользователь вызывает одну из функций OnLineSingle, OnLineDouble или OnLineTriple, которые присваивают переменной m\_LineWidth соответствующее значение. Кроме того, OnUpdateLineDouble, OnUpdateLineSingle или OnUpdateLineTriple гарантируют, что в меню Lines необходимая команда отмечена и на панели инструментов соответствующая кнопка выбрана, а две другие – нет.

Аналогичным образом работают функции, обслуживающие семь кнопок, связанных с инструментами рисования, и соответствующие команды в меню Tools. Функция-обработчик:

- присваивает соответствующее значение переменной m\_CurrentTool;
- выбирает свою кнопку в панели инструментов;
- отмечает в меню Tools свою команду.

## Диалоговые и переключаемые панели

MFC поддерживает два дополнительных элемента пользовательского интерфейса, называемые диалоговой панелью и переключаемой панелью.

- *Диалоговая панель* (dialog bar) – это гибрид панели инструментов и диалогового окна. Диалоговая панель управляется MFC-классом CDialogBar, порожденным от CControlBar. В отличие от панели инструментов, управляемой классом CToolBar, диалоговая панель основана на шаблоне диалогового окна. Соответственно, панель можно разрабатывать в редакторе диалоговых окон Developer Studio и в нее можно вводить элементы управления любого типа, доступные для диалоговых окон. Редактор диалоговых окон описан в гл. 15. Диалоговая панель может размещаться вдоль любой из границ окна приложения. Как и при работе в диалоговом окне, клавиши Tab или Shift+Tab используются для перехода между элементами управления.
- *Переключаемая панель* (tab control) – это прямоугольный блок, обычно отображаемый у верхней границы окна, который может содержать одну или несколько *полос*. Каждая полоса состоит из маркера, текстовой метки, растрового рисунка, а также панели инструментов, элемента управления (например, поля со списком) или дочернего окна другого типа. Пользователь может реорганизовать полосы внутри переключаемой панели путем их перетаскивания. Переключаемая панель используется для размещения на относительно малой площади большого числа элементов управления, так как можно отображать отдельные полосы, частично скрывая неиспользуемые. Например, Internet Explorer отображает свою строку меню и панели инструментов внутри пере-



ключаемой панели. Переключаемую панель можно создать, используя MFC-классы `CReBar` или `CReBarCtrl`. В приложении можно создать как исходную панель инструментов (описанную выше), так и пустую диалоговую панель, которые будут помещены в одну переключаемую панель.

## Строка состояния в программе *ScratchBook*

Можно добавить в MFC-программу строку состояния, выполнив следующие действия:

1. Необходимо определить объект класса `CStatusBar` как член класса главного окна. За создание строки состояния отвечает объект класса `CStatusBar`, описанный в определении класса `CMainFrame` в файле `Mainfrm.h`.

```
protected: // встроенные элементы управления
    CStatusBar m_wndStatusBar;
    CToolBar m_wndToolBar;
```

2. Затем следует в файле `MainFrm.cpp` определить массив `indicators`, хранящий идентификаторы требуемых полей строки состояния.

```
static UINT indicators[] =
{
    ID_SEPARATOR,          // индикатор строки состояния
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
};
```

Идентификатор `ID_SEPARATOR` предназначен для создания пропуска. Так как этот идентификатор размещен первым в массиве, результирующая строка состояния будет иметь слева пустое поле. Первое поле строки состояния должно быть пустым, потому что MFC автоматически отображает внутри этого поля интерактивную справку. Три оставшихся идентификатора, присвоенные элементам массива `indicators`, определены с помощью MFC, которая предоставляет обработчики сообщений для полей с заданными идентификаторами. Эти обработчики отображают текущее состояние клавиш Caps Lock, Num Lock и Scroll Lock. Обратите внимание: второе, третье и четвертое поля выравниваются по правому краю строки состояния, а первое занимает в левой части строки все оставшееся место.

3. В заключение необходимо вызвать две функции (`Create` и `SetIndicators`) класса `CStatusBar` из функции `OnCreate` класса главного окна. В файле `MainFrm.cpp` функция `OnCreate` содержит код для создания строки состояния.

```
if (!m_wndStatusBar.Create(this) ||
    !m_wndStatusBar.SetIndicators(indicators,
    sizeof(indicators)/sizeof(UINT)))
{
    TRACE0("Failed to create status bar\n");
    return -1; // не удалось создать строку состояния
}
```

Код начинается вызовом функции `CStatusBar::Create`, создающей строку состояния. Массив `indicators`, передаваемый функции `CStatusBar::SetIndicators`, задает идентификатор каждого поля строки состояния. MFC предоставляет обработчики сообщений для отображения информации в строке состояния.

Если команды меню не выделены, и кнопки панели инструментов не нажаты, MFC отображает информацию *о простое* в первом поле строки состояния. Стандартное сообщение *о простое* – это

строка Ready. Если хотите задать другое сообщение о простое, то измените с помощью редактора строк строку с идентификатором AFX\_IDS\_IDLEMESSAGE.

Чтобы добавить пользовательское поле в строку состояния программы ScratchBook (сейчас она содержит только стандартные поля, поддерживаемые библиотекой MFC), потребуется включить его идентификатор в массив, передаваемый в функцию CStatusBar::SetIndicators. Кроме того, необходимо использовать редактор строк Visual Studio (см. параграф “Добавление команд в меню File” гл. 12), чтобы определить соответствующий строковый ресурс с таким же идентификатором, как и у поля. Если такая строка не определена, то программа завершится сообщением об ошибке. Когда MFC отображает в строке состояния пользовательское поле, она делает его достаточно широким, чтобы поместился соответствующий строковый ресурс. Однако строка состояния автоматически не отображается, так как вначале поле заблокировано. Чтобы сделать поле доступным для строкового ресурса, необходимо:

1. Сгенерировать для поля обработчик сообщения UPDATE\_COMMAND\_UI (т. е. функцию OnUpdate...).
2. Вызвать из этого обработчика функцию CCmdUI::Enable, опуская параметры или передавая значение TRUE. Тогда строковый ресурс станет видимым.

Чтобы отобразить другое содержимое в пользовательском поле, передайте нужную строку функции CCmdUI::SetText. Внимание: при задании новой отображаемой строки ширина поля *не* будет корректироваться. Если новая строка, переданная в SetText, длиннее строкового ресурса, то она будет усечена и видна не целиком, а только начальная часть.

## Текст программы ScratchBook

---

В приведенных ниже листингах (14.1—14.8) содержатся исходные тексты версии программы ScratchBook, содержащей код для создания панели инструментов и строки состояния. Обратите внимание: при каждом выделении команды меню или помещении указателя мыши над кнопкой панели инструментов MFC отображает интерактивную справку внутри первого поля новой строки состояния, а также статус клавиш Caps Lock, Num Lock и Scroll Lock во втором, третьем и четвертом полях. Кнопки панели инструментов и соответствующие им команды меню не работают, а лишь позволяют задать состояние выбранной кнопки (нажата/не нажата) и отметить команду меню. Инструменты для рисования и задания толщины линий реализованы в версии программы ScratchBook, описанной в гл. 19.

---

### Листинг 14.1.

```
// ScratchBook.h : главный заголовочный файл приложения ScratchBook
//

#pragma once

#ifndef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCN
#endif

#include "resource.h"           // основные символы

// CScratchBookApp:
// Смотрите реализацию этого класса в файле ScratchBook.cpp
//

class CScratchBookApp : public CWinApp
{
}
```

```

public:
    int m_CurrentWidth;
    UINT m_CurrentTool;

public:
    CScratchBookApp();

// Переопределения
public:
    virtual BOOL InitInstance();

// Реализация
    afx_msg void OnAppAbout();
    DECLARE_MESSAGE_MAP()
    afx_msg void OnToolsLine();
    afx_msg void OnToolsRectangle();
    afx_msg void OnUpdateToolsRectangle(CCmdUI *pCmdUI);
    afx_msg void OnToolsFrextangle();
    afx_msg void OnUpdateToolsFrextangle(CCmdUI *pCmdUI);
    afx_msg void OnToolsRRectangle();
    afx_msg void OnUpdateToolsRRectangle(CCmdUI *pCmdUI);
    afx_msg void OnToolsFrrectangle();
    afx_msg void OnUpdateToolsFrrectangle(CCmdUI *pCmdUI);
    afx_msg void OnToolsCircle();
    afx_msg void OnUpdateToolsCircle(CCmdUI *pCmdUI);
    afx_msg void OnToolsFcicle();
    afx_msg void OnUpdateToolsFcicle(CCmdUI *pCmdUI);
    afx_msg void OnLineSingle();
    afx_msg void OnUpdateLineSingle(CCmdUI *pCmdUI);
    afx_msg void OnLineDouble();
    afx_msg void OnUpdateLineDouble(CCmdUI *pCmdUI);
    afx_msg void OnLineTriple();
    afx_msg void OnUpdateLineTriple(CCmdUI *pCmdUI);
    afx_msg void OnUpdateToolsLine(CCmdUI *pCmdUI);
};

```

---

#### Листинг 14.2.

```

// ScratchBook.cpp : Определяет поведение классов приложения
//

#include "stdafx.h"
#include "ScratchBook.h"
#include "MainFrm.h"

#include "ScratchBookDoc.h"
#include "ScratchBookView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// CScratchBookApp

```

```

BEGIN_MESSAGE_MAP(CScratchBookApp, CWinApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
    // Стандартные команды работы с файлами документов
    ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
    ON_COMMAND(ID_TOOLS_LINE, OnToolsLine)
    ON_COMMAND(ID_TOOLS_RECTANGLE, OnToolsRectangle)
    ON_UPDATE_COMMAND_UI(ID_TOOLS_RECTANGLE, OnUpdateToolsRectangle)
    ON_COMMAND(ID_TOOLS_FRECTANGLE, OnToolsFrectangle)
    ON_UPDATE_COMMAND_UI(ID_TOOLS_FRECTANGLE,
        OnUpdateToolsFrectangle)
    ON_COMMAND(ID_TOOLS_RRECTANGLE, OnToolsRrectangle)
    ON_UPDATE_COMMAND_UI(ID_TOOLS_RRECTANGLE,
        OnUpdateToolsRrectangle)
    ON_COMMAND(ID_TOOLS_FRRECTANGLE, OnToolsFrrectangle)
    ON_UPDATE_COMMAND_UI(ID_TOOLS_FRRECTANGLE,
        OnUpdateToolsFrrectangle)
    ON_COMMAND(ID_TOOLS_CIRCLE, OnToolsCircle)
    ON_UPDATE_COMMAND_UI(ID_TOOLS_CIRCLE, OnUpdateToolsCircle)
    ON_COMMAND(ID_TOOLS_FCIRCLE, OnToolsFcircle)
    ON_UPDATE_COMMAND_UI(ID_TOOLS_FCIRCLE, OnUpdateToolsFcircle)
    ON_COMMAND(ID_LINE_SINGLE, OnLineSingle)
    ON_UPDATE_COMMAND_UI(ID_LINE_SINGLE, OnUpdateLineSingle)
    ON_COMMAND(ID_LINE_DOUBLE, OnLineDouble)
    ON_UPDATE_COMMAND_UI(ID_LINE_DOUBLE, OnUpdateLineDouble)
    ON_COMMAND(ID_LINE_TRIPLE, OnLineTriple)
    ON_UPDATE_COMMAND_UI(ID_LINE_TRIPLE, OnUpdateLineTriple)
    ON_UPDATE_COMMAND_UI(ID_TOOLS_LINE, OnUpdateToolsLine)
END_MESSAGE_MAP()

// Конструктор CScratchBookApp

CScratchBookApp::CScratchBookApp()
{
    // TODO: добавьте сюда код конструктора
    // Поместите весь существенный код инициализации
    // в функцию InitInstance

    m_CurrentWidth = 1;
    m_CurrentTool=ID_TOOLS_LINE;
}

// Единственный объект класса CScratchBookApp

CScratchBookApp theApp;

// Инициализация CScratchBookApp

BOOL CScratchBookApp::InitInstance()
{
    CWinApp::InitInstance();

    // Стандартная инициализация.
    // Если вы не используете какие-то из предоставленных
    // возможностей и хотите уменьшить размер конечного

```

```

// исполняемого модуля, удалите из последующего кода
// отдельные команды инициализации элементов, которые
// вам не нужны.
// Измените строку-параметр функции, под которым ваши установки
// хранятся в реестре.
// TODO: Измените эту строку на что-нибудь подходящее,
// например, на имя вашей компании или организации
SetRegistryKey(_T("Local AppWizard-Generated Applications"));
LoadStdProfileSettings(4); // Загрузка стандартных установок
                             // из INI-файла (включая MRU)
// Регистрация шаблона документа приложения. Шаблоны
// документов служат связью между документами, окнами
// документов и окнами приложений
CSingleDocTemplate* pDocTemplate;
pDocTemplate = new CSingleDocTemplate(
    IDR_MAINFRAME,
    RUNTIME_CLASS(CScratchBookDoc),
    RUNTIME_CLASS(CMainFrame), // главное окно
                                // SDI-приложения
    RUNTIME_CLASS(CScratchBookView));
AddDocTemplate(pDocTemplate);

EnableShellOpen ();
RegisterShellFileTypes ();

// Просмотр командной строки для обнаружения стандартных
// команд оболочки, DDE, открытия файлов
CCommandLineInfo cmdInfo;
ParseCommandLine(cmdInfo);
// Выполнение команд, указанных в командной строке.
// Вернет FALSE, если приложение было запущено с
// /RegServer, /Register, /Unregserver или /Unregister.
if (!ProcessShellCommand(cmdInfo))
    return FALSE;
// Показ и обновление единственного проинициализированного окна
m_pMainWnd->ShowWindow(SW_SHOW);
m_pMainWnd->UpdateWindow();

m_pMainWnd->DragAcceptFiles ();

return TRUE;
}

// CAboutDlg диалог, используемый в App About

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

// Данные для диалога
enum { IDD = IDD_ABOUTBOX };

protected:
    virtual void DoDataExchange(CDataExchange* pDX);
    // Поддержка DDX/DDV

```

```

// Реализация
protected:
    DECLARE_MESSAGE_MAP()
public:
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
END_MESSAGE_MAP()

// Команда приложения для выполнения диалога
void CScratchBookApp::OnAppAbout()
{
    CAboutDlg aboutDlg;
    aboutDlg.DoModal();
}

// Обработчики сообщений класса CScratchBookApp

void CScratchBookApp::OnToolsLine()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentTool = ID_TOOLS_LINE;
}

void CScratchBookApp::OnToolsRectangle()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentTool = ID_TOOLS_RECTANGLE;
}

void CScratchBookApp::OnUpdateToolsRectangle(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck
        (m_CurrentTool == ID_TOOLS_RECTANGLE ? 1 : 0);
}

void CScratchBookApp::OnToolsFrextangle()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentTool = ID_TOOLS_FREXTANGLE;
}

void CScratchBookApp::OnUpdateToolsFrextangle(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика

```

```

        pCmdUI->SetCheck
            (m_CurrentTool == ID_TOOLS_FRECTANGLE ? 1 : 0);
    }

void CScratchBookApp::OnToolsRrectangle()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentTool = ID_TOOLS_RRECTANGLE;
}

void CScratchBookApp::OnUpdateToolsRrectangle(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck
        (m_CurrentTool == ID_TOOLS_RRECTANGLE ? 1 : 0);
}

void CScratchBookApp::OnToolsFrrectangle()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentTool = ID_TOOLS_FRRECTANGLE;
}

void CScratchBookApp::OnUpdateToolsFrrectangle(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck
        (m_CurrentTool == ID_TOOLS_FRRECTANGLE ? 1 : 0);
}

void CScratchBookApp::OnToolsCircle()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentTool = ID_TOOLS_CIRCLE;
}

void CScratchBookApp::OnUpdateToolsCircle(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck
        (m_CurrentTool == ID_TOOLS_CIRCLE ? 1 : 0);
}

void CScratchBookApp::OnToolsFcircle()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentTool = ID_TOOLS_FCIRCLE;
}

void CScratchBookApp::OnUpdateToolsFcircle(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck
        (m_CurrentTool==ID_TOOLS_FCIRCLE ? 1 : 0);
}

```

```

void CScratchBookApp::OnLineSingle()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentWidth = 1;
}

void CScratchBookApp::OnUpdateLineSingle(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck(m_CurrentWidth == 1 ? 1 : 0);
}

void CScratchBookApp::OnLineDouble()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentWidth = 2;
}

void CScratchBookApp::OnUpdateLineDouble(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck(m_CurrentWidth == 2 ? 1 : 0);
}

void CScratchBookApp::OnLineTriple()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentWidth = 3;
}

void CScratchBookApp::OnUpdateLineTriple(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck(m_CurrentWidth == 3 ? 1 : 0);
}

void CScratchBookApp::OnUpdateToolsLine(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck(m_CurrentTool == ID_TOOLS_LINE ? 1 : 0);
}

```

---

### Листинг 14.3.

```

// ScratchBookDoc.h : интерфейс класса CScratchBookDoc
//

#pragma once

class CLine : public CObject
{
protected:
    int m_X1, m_Y1, m_X2, m_Y2;
    CLine ()
    {}
    DECLARE_SERIAL (CLine)

```



```

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);
    CRect GetDimRect ();
    virtual void Serialize (CArchive &ar);
};

class CScratchBookDoc : public CDocument
{
protected:
    CTypedPtrArray<CObArray, CLine*> m_LineArray;

public:
    CLine *AddLine (int X1, int Y1, int X2, int Y2);
    CLine *GetLine (int Index);
    int GetNumLines ();

protected: // используется только для сериализации
    CScratchBookDoc();
    DECLARE_DYNCREATE(CScratchBookDoc)

// Атрибуты
public:

// Операции
public:

// Переопределения
public:
    virtual BOOL OnNewDocument();
    virtual void Serialize(CArchive& ar);

// Реализация
public:
    virtual ~CScratchBookDoc();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Сгенерированные функции карты сообщений
protected:
    DECLARE_MESSAGE_MAP()
public:
    virtual void DeleteContents();
    afx_msg void On57633();
    afx_msg void OnUpdate57633(CCmdUI *pCmdUI);

```

```

        afx_msg void On57643();
        afx_msg void OnUpdate57643(CCmdUI *pCmdUI);
};

```

---

#### Листинг 14.4.

```

// ScratchBookDoc.cpp : реализация класса CScratchBookDoc
//

#include "stdafx.h"
#include "ScratchBook.h"

#include "ScratchBookDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// CScratchBookDoc

IMPLEMENT_DYNCREATE(CScratchBookDoc, CDocument)

BEGIN_MESSAGE_MAP(CScratchBookDoc, CDocument)
    ON_COMMAND(57633, On57633)
    ON_UPDATE_COMMAND_UI(57633, OnUpdate57633)
    ON_COMMAND(57643, On57643)
    ON_UPDATE_COMMAND_UI(57643, OnUpdate57643)
END_MESSAGE_MAP()

// Конструктор/деструктор класса CScratchBookDoc

CScratchBookDoc::CScratchBookDoc()
{
    // TODO: добавьте сюда код одноразового конструктора
}

CScratchBookDoc::~CScratchBookDoc()
{
}

BOOL CScratchBookDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;

    // TODO: добавьте сюда код повторной инициализации
    // (SDI-документы будут использовать этот документ
    // многократно)

    return TRUE;
}

```

```

// Сериализация CScratchBookDoc

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

// Диагностика класса CScratchBookDoc

#ifdef _DEBUG
void CScratchBookDoc::AssertValid() const
{
    CDocument::AssertValid();
}

void CScratchBookDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}
#endif //_DEBUG

// Команды класса CScratchBookDoc

IMPLEMENT_SERIAL (CLine, CObject, 1)

void CLine::Draw (CDC *PDC)
{
    PDC->MoveTo (m_X1, m_Y1);
    PDC->LineTo (m_X2, m_Y2);
}

void CLine::Serialize (CArchive &ar)
{
    if (ar.IsStoring())
        ar << m_X1 << m_Y1 << m_X2 << m_Y2;
    else
        ar >> m_X1 >>m_Y1 >> m_X2 >> m_Y2;
}

CLine *CScratchBookDoc::AddLine (int X1, int Y1, int X2, int Y2)
{
    CLine *PLine = new CLine (X1, Y1, X2, Y2);
    m_LineArray.Add (PLine);
    SetModifiedFlag( );
    return PLine;
}

```

```

CLine *CScratchBookDoc::GetLine (int Index)
{
    if (Index < 0 || Index > m_LineArray.GetUpperBound ())
        return 0;
    return m_LineArray.GetAt (Index);
}

int CScratchBookDoc::GetNumLines ()
{
    return m_LineArray.GetSize ();
}

void CScratchBookDoc::DeleteContents()
{
    // TODO: Добавьте сюда собственный код или
    // вызов базового класса
    int Index = m_LineArray.GetSize ();
    while (Index--)
        delete m_LineArray.GetAt (Index);
    m_LineArray.RemoveAll ();

    CDocument::DeleteContents();
}

void CScratchBookDoc::On57633()
{
    // TODO: Добавьте сюда собственный код обработчика
    DeleteContents ();
    UpdateAllViews (0);
    SetModifiedFlag ( );
}

void CScratchBookDoc::OnUpdate57633(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика

    pCmdUI->Enable (m_LineArray.GetSize ());
}

void CScratchBookDoc::On57643()
{
    // TODO: Добавьте сюда собственный код обработчика
    int Index = m_LineArray.GetUpperBound ();
    if (Index > -1)
    {
        delete m_LineArray.GetAt (Index);
        m_LineArray.RemoveAt (Index);
    }
    UpdateAllViews (0);
    SetModifiedFlag ();
}

void CScratchBookDoc::OnUpdate57643(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->Enable (m_LineArray.GetSize ());
}

```

```

CRect CLine::GetDimRect ()
{
    return CRect
        (min (m_X1, m_X2), min (m_Y1, m_Y2),
         max (m_X1, m_X2) + 1, max (m_Y1, m_Y2) + 1);
}

```

---

#### Листинг 14.5.

```

// ScratchBookView.h : интерфейс класса CScratchBookView
//

#pragma once

class CScratchBookView : public CScrollView
{
protected:
    CString m_ClassName;
    int m_Dragging;
    HCURSOR m_HCross;
    HCURSOR m_HArrow;
    CPoint m_PointOld;
    CPoint m_PointOrigin;

protected: // используется только для сериализации
    CScratchBookView();
    DECLARE_DYNCREATE(CScratchBookView)

// Атрибуты
public:
    CScratchBookDoc* GetDocument() const;

// Операции
public:

// Переопределения
public:
    virtual void OnDraw(CDC* pDC);
    // переопределена для прорисовки этого вида
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
protected:

// Реализация
public:
    virtual ~CScratchBookView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Сгенерированные функции карты сообщений
protected:

```

```

        DECLARE_MESSAGE_MAP()
public:
    afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
    afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
    afx_msg void OnMouseMove(UINT nFlags, CPoint point);
    virtual void OnInitialUpdate();
protected:
    virtual void OnUpdate(CView* /*pSender*/, LPARAM /*lHint*/, CObject*
/*pHint*/);
};

#ifdef _DEBUG // отладочная версия в файле ScratchBookView.cpp
inline CScratchBookDoc* CScratchBookView::GetDocument() const
    { return (CScratchBookDoc*)m_pDocument; }
#endif

```

---

#### Листинг 14.6.

```

// ScratchBookView.cpp : реализация класса CScratchBookView
//

#include "stdafx.h"
#include "ScratchBook.h"

#include "ScratchBookDoc.h"
#include "ScratchBookView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// CScratchBookView

IMPLEMENT_DYNCREATE(CScratchBookView, CScrollView)

BEGIN_MESSAGE_MAP(CScratchBookView, CScrollView)
    ON_WM_LBUTTONDOWN()
    ON_WM_LBUTTONUP()
    ON_WM_MOUSEMOVE()
END_MESSAGE_MAP()

// Конструктор/деструктор класса CScratchBookView

CScratchBookView::CScratchBookView()
{
    // TODO: добавьте сюда собственный код конструктора

    m_Dragging = 0;
    m_HCross = AfxGetApp()->LoadStandardCursor (IDC_CROSS);
    m_HArrow = AfxGetApp()->LoadStandardCursor (IDC_ARROW);
}

CScratchBookView::~CScratchBookView()
{
}

```

```

BOOL CScratchBookView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Модифицируйте класс или стили окна,
    // добавляя и изменяя поля структуры cs

    m_ClassName = AfxRegisterWndClass
        (CS_HREDRAW | CS_VREDRAW,    // стили окна
        0,                          // без указателя
        (HBRUSH)::GetStockObject (WHITE_BRUSH),
        0);                          // задать белый фон
    cs.lpszClass = m_ClassName;      // без значка

    return CScrollView::PreCreateWindow(cs);
}

// Прорисовка CScratchBookView

void CScratchBookView::OnDraw(CDC* pDC)
{
    CScratchBookDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    // TODO: вставьте сюда код прорисовки собственных данных
    CSize ScrollSize = GetTotalSize ();
    pDC->MoveTo (ScrollSize.cx, 0);
    pDC->LineTo (ScrollSize.cx, ScrollSize.cy);
    pDC->LineTo (0, ScrollSize.cy);

    CRect ClipRect;
    CRect DimRect;
    CRect IntRect;
    CLine *PLine;
    pDC->GetClipBox (&ClipRect);

    int Index = pDoc->GetNumLines ();
    while (Index--)
    {
        PLine = pDoc->GetLine (Index);
        DimRect = PLine->GetDimRect ();
        if (IntRect.IntersectRect (DimRect, ClipRect))
            PLine->Draw(pDC);
    }
}

// Диагностика класса CScratchBookView

#ifdef _DEBUG
void CScratchBookView::AssertValid() const
{
    ,
    CScrollView::AssertValid();
}

void CScratchBookView::Dump(CDumpContext& dc) const
{
    CScrollView::Dump(dc);
}

```

```

CScratchBookDoc* CScratchBookView::GetDocument() const
// не отладочная версия встроена
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CScratchBookDoc)));
    return (CScratchBookDoc*)m_pDocument;
}
#endif // _DEBUG

// Обработчики сообщений класса CScratchBookView

void CScratchBookView::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: Вставьте сюда собственный код обработчика
    // или вызов стандартного
    CClientDC ClientDC (this);
    OnPrepareDC (&ClientDC);
    ClientDC.DPtoLP (&point);

    // проверка нахождения курсора внутри области
    // окна представления
    CSize ScrollSize = GetTotalSize ();
    CRect ScrollRect (0, 0, ScrollSize.cx, ScrollSize.cy);
    if (!ScrollRect.PtInRect (point))
        return;

    // сохранение позиции курсора, захват мыши и
    // установка флага перемещения
    m_PointOrigin = point;
    m_PointOld = point;
    SetCapture ();
    m_Dragging = 1;

    // ограничение перемещений курсора мыши
    ClientDC.LPtoDP (&ScrollRect);
    CRect ViewRect;
    GetClientRect (&ViewRect);
    CRect IntRect;
    IntRect.IntersectRect (&ScrollRect, &ViewRect);
    ClientToScreen (&IntRect);
    ::ClipCursor (&IntRect);

    CScrollView::OnLButtonDown(nFlags, point);
}

void CScratchBookView::OnLButtonUp(UINT nFlags, CPoint point)
{
    // TODO: Вставьте сюда собственный код обработчика
    // или вызов стандартного
    if (m_Dragging)
    {
        m_Dragging = 0;
        ::ReleaseCapture ();
        ::ClipCursor (NULL);
        CClientDC ClientDC (this);
        OnPrepareDC (&ClientDC);
    }
}

```



```

        ClientDC.DPtoLP (&point);

        ClientDC.SetROP2 (R2_NOT);
        ClientDC.MoveTo (m_PointOrigin);
        ClientDC.LineTo (m_PointOld);
        ClientDC.SetROP2 (R2_COPYPEN);
        ClientDC.MoveTo (m_PointOrigin);
        ClientDC.LineTo (point);

        CScratchBookDoc* PDoc = GetDocument();
        CLine *PCLine;
        PCLine = PDoc -> AddLine (m_PointOrigin.x,
                                   m_PointOrigin.y, point.x, point.y);

        PDoc->UpdateAllViews (this, 0, PCLine);
    }

    CScrollView::OnLButtonUp(nFlags, point);
}

void CScratchBookView::OnMouseMove(UINT nFlags, CPoint point)
{
    // TODO: Вставьте сюда собственный код обработчика
    // или вызов стандартного

    CClientDC ClientDC (this);
    OnPrepareDC (&ClientDC);
    ClientDC.DPtoLP (&point);

    CSize ScrollSize = GetTotalSize();
    CRect ScrollRect(0, 0, ScrollSize.cx, ScrollSize.cy);
    if (ScrollRect.PtInRect (point))
        ::SetCursor (m_HCross);
    else
        ::SetCursor (m_HArrow);

    if (m_Dragging)
    {
        ClientDC.SetROP2 (R2_NOT);
        ClientDC.MoveTo (m_PointOrigin);
        ClientDC.LineTo (m_PointOld);
        ClientDC.MoveTo (m_PointOrigin);
        ClientDC.LineTo (point);
        m_PointOld = point;
    }

    CScrollView::OnMouseMove(nFlags, point);
}

void CScratchBookView::OnInitialUpdate()
{
    CScrollView::OnInitialUpdate();

    // TODO: Добавьте сюда собственный код или
    // вызов базового класса

```

```

        SIZE Size = {800, 600};
        SetScrollSizes (MM_TEXT, Size);
    }

void CScratchBookView::OnUpdate(CView* pSender, LPARAM lHint, CObject* pHint)
{
    // TODO: Добавьте сюда собственный код или
    // вызов базового класса
    if (pHint !=0)
    {
        CRect InvalidRect = ((CLine *)pHint)->GetDimRect ();
        CClientDC ClientDC (this);
        OnPrepareDC (&ClientDC);
        ClientDC.LPtoDP (&InvalidRect);
        InvalidateRect (&InvalidRect);
    }
    else
        CScrollView::OnUpdate (pSender, lHint, pHint);
}

```

---

#### Листинг 14.7.

```

// MainFrm.h : интерфейс класса CMainFrame
//

#pragma once
class CMainFrame : public CFrameWnd
{
protected:
    CSplitterWnd m_SplitterWnd;

protected: // используется только для сериализации
    CMainFrame();
    DECLARE_DYNCREATE(CMainFrame)

// Атрибуты
public:

// Операции
public:

// Переопределения
public:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);

// Реализация
public:
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif
protected: // встроенные элементы управления
    CStatusBar m_wndStatusBar;
    CToolBar m_wndToolBar;
}

```

```

// Сгенерированные функции карты сообщений
protected:
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    DECLARE_MESSAGE_MAP()
    virtual BOOL OnCreateClient(LPCREATESTRUCT lpcs,
                                CCreateContext* pContext);
};

```

---

#### Листинг 14.8.

```

// MainFrm.cpp : реализация класса CMainFrame
//

#include "stdafx.h"
#include "ScratchBook.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
    ON_WM_CREATE()
END_MESSAGE_MAP()

static UINT indicators[] =
{
    ID_SEPARATOR,           // индикатор строки состояния
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
};

// Конструктор/деструктор класса CMainFrame

CMainFrame::CMainFrame()
{
    // TODO: добавьте сюда код инициализации элементов
}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;
}

```

```

        if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT,
            WS_CHILD | WS_VISIBLE | CBRS_TOP | CBRS_GRIPPER |
            CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC) ||
            !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
        {
            TRACE0("Failed to create toolbar\n");
            return -1;        // не удалось создать панель инструментов
        }

        if (!m_wndStatusBar.Create(this) ||
            !m_wndStatusBar.SetIndicators(indicators,
            sizeof(indicators)/sizeof(UINT)))
        {
            TRACE0("Failed to create status bar\n");
            return -1;        // не удалось создать строку состояния
        }
        // TODO: Удалите три следующие строки, если не хотите,
        // чтобы панель инструментов была паркующей
        m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
        EnableDocking(CBRS_ALIGN_ANY);
        DockControlBar(&m_wndToolBar);

        return 0;
    }

    BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
    {
        if( !CFrameWnd::PreCreateWindow(cs) )
            return FALSE;
        // TODO: Модифицируйте стили или классы окна здесь,
        // добавляя или изменяя поля структуры cs

        return TRUE;
    }

    // Диагностика класса CMainFrame

#ifdef _DEBUG
    void CMainFrame::AssertValid() const
    {
        CFrameWnd::AssertValid();
    }

    void CMainFrame::Dump(CDumpContext& dc) const
    {
        CFrameWnd::Dump(dc);
    }

#endif // _DEBUG

    // Обработчики сообщений класса CMainFrame

    BOOL CMainFrame::OnCreateClient(LPCREATESTRUCT lpcs, CCreateContext* pContext)
    {
        // TODO: Добавьте сюда собственный код обработчика или
        // вызов базового класса
        return m_SplitterWnd.Create

```

```

        (this,                // родительское окно разделенного окна;
        1,                    // максимальное число строк;
        2,                    // максимальное число столбцов;
        CSize (20, 20),       // минимальный размер окна представления;
        pContext);            // информация о контексте устройства
    }

```

## Резюме

---

На примере редактора графики мы узнали, как добавить перемещаемую панель инструментов и строку состояния в MFC-программу.

- *Перемещаемая панель инструментов.* Перемещаемая панель содержит кнопки, позволяющие быстро выполнять команды меню. Ее можно припарковать к любому краю окна приложения или превратить в свободно перемещаемое окно. При добавлении перемещаемой панели инструментов к существующей программе используйте редактор панели инструментов Visual Studio для разработки рисунка каждой кнопки панели. Чтобы добавить в программу панель инструментов, необходимо определить объект класса `CToolBar` как член класса главного окна, а затем создать и инициализировать панель инструментов, вызвав функции `Create` и `LoadToolBar` класса `CToolBar`. Они вызываются в классе главного окна из функции `OnCreate`, обрабатывающей сообщение `WM_CREATE`. Для придания панели инструментов свойства перемещаемости (т.е. чтобы пользователь мог изменять ее положение) из функции `OnCreate` вызываются методы `CControlBar::EnableDocking`, `CFrameWnd::EnableDocking` и `CFrameWnd::DockControlBar`.
- *Команды меню.* Чтобы предусмотреть альтернативный способ выполнения операции, вызываемой нажатием кнопки панели инструментов, можно определить соответствующие кнопкам панели инструментов команды меню.
- *Обработчики сообщений.* Обработчики сообщений для команд панели инструментов определяются таким же образом, как и обработчики сообщений для команд меню. Фактически, если оба эти элемента имеют один и тот же идентификатор, то одна функция может обрабатывать сообщения, передаваемые и от кнопки панели инструментов, и от соответствующей команды меню.
- *Строка состояния.* Строка состояния отображает интерактивную справку для команд меню и кнопок панели инструментов, а также показывает состояние некоторых клавиш. К существующей MFC-программе можно добавить строку состояния, определив экземпляр класса `CStatusBar` как член класса главного окна, а затем вызвав функции `Create` и `SetIndicators` класса `CStatusBar` из обработчика сообщения `OnCreate` класса главного окна. При этом функции передается массив, определяющий идентификаторы полей в строке состояния и размещение этих полей.

# Глава 15

## Диалоговые окна

---

- **Модальное диалоговое окно**
- **Немодальное диалоговое окно**
- **Диалоговое окно с вкладками**
- **Диалоговое окно общего назначения**

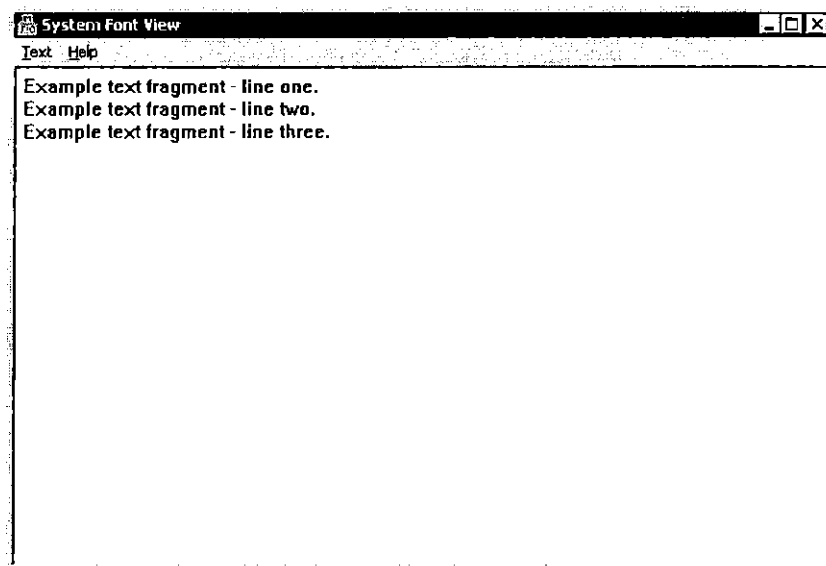
В этой главе вы научитесь самостоятельно создавать диалоговые окна и отображать их в своих программах. В созданных ранее программах мастер Application Wizard генерировал ресурсы и фрагмент программы, необходимые для отображения простого диалогового окна после выбора команды About в меню Help. Роль диалоговых окон для отображения информации и получения данных очень велика.

- В первой части главы рассматривается проектирование и отображение *модального* диалогового окна, которое временно отображается поверх главного окна программы и удаляется после чтения или ввода информации. В качестве примера создана программа FontView, отображающая диалоговое окно для форматирования текста.
- Кратко описаны способы отображения *немодального* диалогового окна, которое может быть открыто при работе в главном окне приложения.
- Далее приведены сведения о том, как создаются диалоговые окна *с вкладками*, состоящие из нескольких *страниц* с элементами управления и отображающие в своей верхней части строку *ярлычков* вкладок. Нужная страница открывается после щелчка на ярлычке соответствующей вкладки. Страницу, расположенную в диалоговом окне с вкладками, вместе с ее ярлычком, иногда для простоты называют вкладкой. Например, в этой книге при изучении диалоговых окон Visual Studio выражение “Откройте вкладку” используется вместо фразы “Щелкните на ярлычке для того, чтобы открыть страницу”. Для примера создана программа TabView, использующая вместо окна, отображающего сразу все элементы управления, диалоговое окно с вкладками.
- Завершается глава рассмотрением стандартных диалоговых окон *общего назначения*, предоставляемые Windows. Программа, главное окно которой основано на шаблоне диалогового окна и содержит набор элементов управления, будет создана в следующей главе.

## Модальные диалоговые окна

---

Этот раздел посвящен созданию и отображению модального диалогового окна на примере программы FontView. Программа FontView отображает несколько строк текста внутри окна представления, используя системный шрифт Windows.



Если в меню Text выбрать команду Text Properties... или нажать клавиши Ctrl+T, то программа отобразит диалоговое окно Text Properties, позволяющее изменять способ форматирования текста в окне представления. Диалоговое окно Text Properties задает:

- *начертание символов*: полужирное (шрифт System не может быть полужирным), курсив, подчеркнутое или любую их комбинацию;
- *выравнивание строк*: влево, по центру или вправо;
- *ширину символов*: переменная или фиксированная;
- *интервал между строками*: одинарный, двойной или тройной.

Если щелкнуть на кнопке ОК, то диалоговое окно закроется, а выбранные параметры форматирования будут применены к тексту в главном окне. Если закрыть диалоговое окно, щелкнув на кнопке Cancel или кнопке закрытия (или выбрав команду Close в системном меню либо нажав клавишу Esc), то диалоговое окно закроется без изменений формата текста в главном окне.

Пока диалоговое окно Text Properties открыто активизировать главное окно программы или выбрать какую-либо команду меню нельзя. (Если попытаться сделать это, программа подаст звуковой сигнал.) Чтобы продолжить работу в главном окне, необходимо закрыть диалоговое окно. Такое диалоговое окно называют *модальным* (наиболее простой тип диалоговых окон). Создание *немодальных* диалоговых окон, позволяющих работать в главном окне программы одновременно с отображением диалогового окна, рассмотрено далее в этой главе.

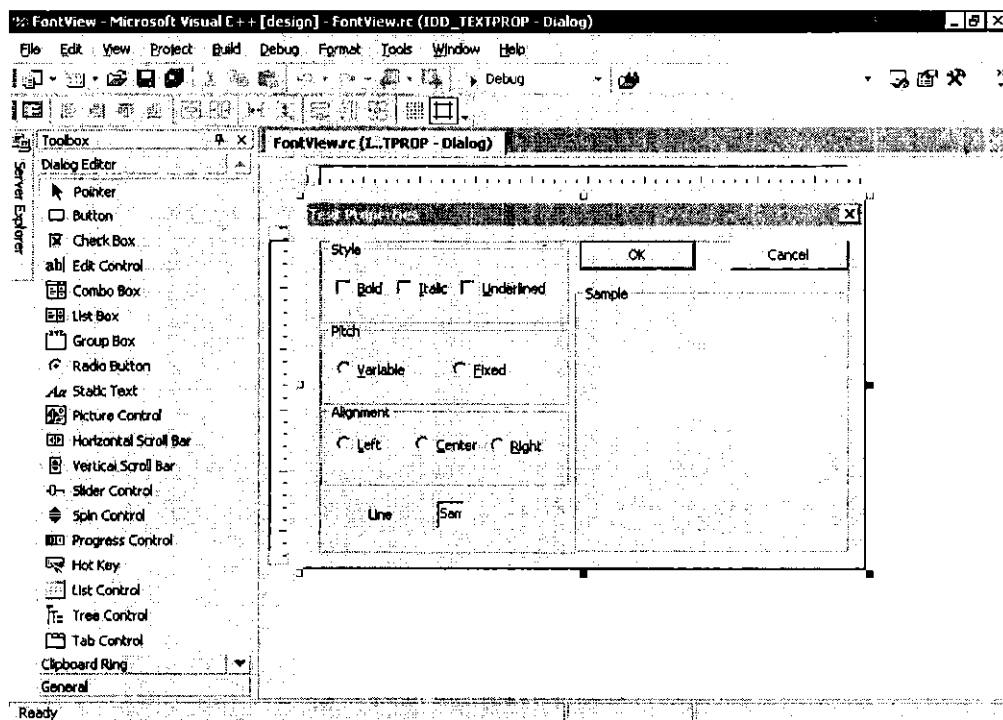
## Программа FontView

Воспользуемся мастером Application Wizard для создания исходных файлов программы FontView (см. гл. 9). Введите имя программы FontView в поле Name и соответствующий путь для каталога проекта в поле Location. На вкладках диалогового окна мастера отключите опции создания панели инструментов и строки состояния (в создаваемой программе они не используются).

## Диалоговое окно Text Properties

Займемся разработкой диалогового окна Text Properties, используя редактор диалоговых окон. Откройте вкладку Resource View, выберите в дереве ресурсов папку Dialog, щелкните на ней правой

кнопкой мыши и в контекстном меню выберите команду Add Resource. В открывшемся диалоговом окне выберите тип ресурса Dialog и нажмите кнопку New. В дерево ресурсов программы будет добавлено новое окно IDD\_DIALOG1. Дважды щелкните на его значке левой кнопкой мыши. Будет открыт редактор диалоговых окон. Он отображает точную копию создаваемого диалогового окна. Первоначально в нем есть только кнопки OK и Cancel. Панель инструментов содержит кнопки для создания элементов управления всех типов, которые можно добавлять в диалоговое окно.



На рисунке показано диалоговое окно Text Properties в законченном виде (в окне редактора диалоговых окон). В табл. 15.1 приведены свойства диалогового окна и его элементов управления. Ниже приведены инструкции по добавлению элементов управления и заданию свойств диалогового окна.

Выберите диалоговое окно. Для этого выберите в панели инструментов элемент Pointer (указатель, используемый для выбора элементов в окне), и щелкните на области заголовка редактируемого окна. Вокруг окна появятся размерные маркеры – объект выбран. Размер диалогового окна можно изменить, перемещая эти маркеры. Нажмите клавишу F4 для отображения окна Properties, в котором для редактируемого окна установите у свойства Caption (это надпись в строке заголовка) значение Text Properties.

При проектировании программ для Windows NT/2000/XP выбор стиля System Menu во вкладке Style диалогового окна Properties приводит к тому, что:

- на правом краю строки заголовка отображается *кнопка закрытия окна* (щелчок на этой кнопке равносителен щелчку на кнопке с идентификатором IDCANCEL);
- при щелчке правой кнопкой мыши на строке заголовка окна появляется *системное меню*, позволяющее перемещать или закрывать диалоговое окно (команда Close в системном меню действует так же, как и кнопка с идентификатором IDCANCEL);
- не отображается *значок системного меню* на левом краю строки заголовка, как это делается в главных окнах программ.



Табл. 15.1. Свойства диалогового окна Text Properties и его элементов управления

Идентификатор (ID)	Тип элемента управления (в разделе Dialog Editor панели инструментов)	Свойства, задаваемые принудительно
IDD_TEXTPROP	Диалоговое окно	Надпись (Caption): Text Properties
IDC_STATIC	Рамка	Надпись (Caption): Style
IDD_BOLD	Флажок	Надпись: (Caption); &Bold Group Tab Stop
IDC_ITALIC	Флажок	Надпись (Caption): &Italic
IDC_UNDERLINE	Флажок	Надпись (Caption): &Underline
IDC_STATIC	Рамка	Надпись (Caption): Alignment
IDC_LEFT	Переключатель	Надпись (Caption): &Left Group Tab Stop
IDC_CENTER	Переключатель	Надпись (Caption): &Center
IDC_RIGHT	Переключатель	Надпись (Caption): &Right
IDC_STATIC	Рамка	Надпись (Caption): Pitch
IDC_VARIABLE	Переключатель	Надпись (Caption): &Variable Group Tab Stop
IDC_FIXED	Переключатель	Надпись (Caption): &Fixed
IDC_STATIC	Надпись	Надпись (Caption): Line &Spacing:
IDC_SPACING	Поле	
IDC_SAMPLE	Рамка	Надпись (Caption): Sample
IDOK	Кнопка	Кнопка по умолчанию (Default Button)
IDCANCEL	Кнопка	










Заполняя окно другими элементами и выбирая их, в окне Properties можно устанавливать свойства этих элементов. Чтобы добавить элемент управления определенного типа, щелкните на соответствующем элементе панели инструментов, а затем – на интересующем вас месте внутри редактируемого окна. Элемент управления помещается в указанную позицию. Для перемещения или изменения размера элемента управления используйте мышь. Чтобы установить свойства элемента управления, выполните на нем щелчок правой кнопкой мыши и выберите в контекстном меню команду Properties для открытия диалогового окна Properties. Введите идентификатор элемента управления и задайте все его дополнительные свойства, которые не устанавливаются по умолчанию (табл. 15.1). Для некоторых элементов управления нужно оставлять стандартные идентификаторы, для других – идентификатор придется изменить.










Если добавленный элемент управления необходимо удалить, выберите его щелчком мыши, а затем нажмите клавишу Del. Заметим, что символ & внутри надписи элемента управления подчеркивает следующий за ним символ. Подчеркнутый символ называют *мнемоническим*. При отображении диалогового окна элемент управления активизируется в результате нажатия клавиши Alt и подчеркнутого символа. Реакция зависит от типа элемента:







- *флажок или переключатель* – нажатие клавиши Alt и клавиши подчеркнутого символа устанавливает элемент управления;
- *кнопка* – нажатие такого сочетания клавиш равносильно непосредственному щелчку на кнопке;
- *надпись*, – нажатие указанных клавиш размещает курсор в первом следующем за ней по порядку обхода поле (см. ниже).

Большинству элементов управления (см. табл. 15.1) присвоены уникальные идентификаторы, используемые для ссылок на элементы управления в исходном тексте программы. Однако элементам управления, ссылка на которые в программе *отсутствует*, уникальные идентификаторы не нужны. Для них используются стандартные идентификаторы, предлагаемые редактором диалоговых окон (это элементы управления с идентификатором IDC\_STATIC). Задание свойства Default Button для кнопки ОК приводит к отображению кнопки с утолщенной рамкой и позволяет активизировать кнопку (т.е. закрыть диалоговое окно) нажатием клавиши Enter (свойства Tab Stop и Group поясняются ниже).

Табл. 15.2. Разновидности элементов управления

Обозначение элемента в Редакторе окон	Описание элемента
 Picture Control	<i>Рисунок (picture control)</i> – элемент управления, отображающий пустую прямоугольную рамку, закрашенную прямоугольную область или значок. Такой объект задается установкой свойств элемента управления. Не принимает информацию, вводимую пользователем
 Static Text	<i>Надпись (static text control)</i> отображает строку текста. Обычно она используется как метка поля или элемента управления другого типа. Не принимает информацию, вводимую пользователем
 Group Box	<i>Рамка (group box)</i> – это групповая рамка с надписью, используемая для группирования набора связанных элементов управления. Не принимает информацию, вводимую пользователем
 Edit Control	<i>Поле (Edit box)</i> – элемент управления, который позволяет вводить текст и предоставляет средства редактирования (перемещение курсора, операции удаления, вырезания и вставки текста и т.д.). Если во вкладке Style диалогового окна Properties задать стиль Multi-line, то можно ввести в поле несколько строк, что позволяет использовать его в качестве небольшого текстового редактора
 Rich Edit 2.0 Control	<i>Расширенное поле (rich edit 2.0 control)</i> – поле, в котором можно применять форматирование символов и абзацев текста
 IP Address Control	<i>IP-адрес (IP-address control)</i> – поле, содержащее четыре отдельных поля, облегчающих ввод IP-адреса, например 255.116.39.1
 Button	<i>Кнопка (button)</i> используется для немедленного выполнения операции
 Check Box	<i>Флажок (check box)</i> используется для задания или отмены опции, которая не связана с другими опциями
 Radio Button	<i>Переключатель (radio button)</i> используется для выбора одного параметра из группы взаимоисключающих

Обозначение элемента в Редакторе окон	Описание элемента
 List Control	<i>Список (list box)</i> – поле, в котором отображается прокручиваемый список элементов (имен файлов или шрифтов). <i>Окно списка (list control)</i> – разновидность, отображающая набор значков с текстовыми метками. Пункты списка различаются мелкими или крупными значками, упорядоченными по горизонтали или вертикали. Примером такого списка является правая панель окна Windows Explorer при выборе в меню View пункта List
 Tree Control	<i>Дерево (tree control)</i> отображает такой же список, как в окне Solution Explorer в Visual Studio или в левой панели Windows Explorer
 Combo Box	<i>Поле со списком (combo box)</i> – элемент управления, объединяющий список с полем или надписью. Простое поле со списком состоит из поля и постоянно отображаемого под ним списка. <i>Поле с раскрывающимся списком (dropdown combo box)</i> состоит из поля и списка, появляющегося только после щелчка на стрелке справа от поля. <i>Статическое поле со списком (drop list)</i> состоит из надписи (с рамкой) и списка, появляющегося при щелчке на стрелке. Чтобы задать тип поля со списком, откройте вкладку Styles в диалоговом окне Combo Box Properties и выберите тип поля в списке Type
 Extended Combo...	<i>Расширенное поле со списком (Extended combo box)</i> позволяет в качестве элементов списка отображать рисунки
 Horizontal Scroll Bar  Vertical Scroll Bar	<i>Полоса прокрутки (scroll bar)</i> используется для отображения горизонтальных или вертикальных полос прокрутки в любом месте диалогового окна. Эти элементы управления позволяют настраивать некоторые изменяемые значения, например, размер шрифта, интенсивность цвета или скорость перемещения мыши
 Spin Control	<i>Счетчик (spin control)</i> состоит из пары кнопок со стрелками. Он увеличивает или уменьшает на единицу какое-либо значение или выполняет перемещение между различными элементами (например, вкладками диалогового окна). Стрелки можно ориентировать вертикально или горизонтально. Счетчик часто используется вместе с полем для ввода чисел
 Slider Control	<i>Регулятор (slider control или trackbar)</i> изменяет различные значения, например, оттенки цветов. Он содержит шкалу со специальным бегунком, который можно перетаскивать для коррекции значений, а также необязательные отметки. В отличие от полосы прокрутки (scroll bar) регулятор обычно (но не обязательно) имеет ряд фиксированных позиций (значений), между которыми производится переключение. Примеры можно увидеть в программе Keyboard панели управления Windows
 Progress Control	<i>Индикатор (Progress control)</i> – элемент управления, отображающий набор прямоугольников, количество которых зависит от хода процесса (например, печати документа или записи файла). Многие программы, например, Microsoft Word, отображают индикатор в строке состояния во время длительных операций

Обозначение элемента в Редакторе окон	Описание элемента
 Animation Control	<i>Анимационный элемент (Animation control)</i> – небольшое окно, отображающее видеоклип в стандартном формате AVI (Audio Video Interleaved). Можно использовать анимацию для развлечения пользователей во время выполнения длительной операции (например, в Windows 95 программа поиска файлов отображает вращающееся увеличительное стекло)
 Hot Key	<i>Горячая клавиша (Hot-key control)</i> – поле, позволяющее пользователю вводить сочетания клавиш, используемых как “горячие” клавиши. Примером такого элемента управления является поле Key в диалоговом окне Accel Properties, рассмотренное в параграфе “Создание программы MiniEdit” гл. 10
 Date Time Picker	<i>Элемент Дата/время (Date/time picker control)</i> облегчает ввод даты и времени. Этот элемент управления состоит из нескольких полей. Если он используется для ввода даты, то содержит раскрывающийся список с календарем для ее выбора. Если же он используется для ввода времени, то содержит счетчики для настройки времени
 Month Calendar ...	<i>Месячный календарь (Month calendar control)</i> отображает календарь, позволяющий выбрать месяц, а затем дату
 Tab Control	<i>Набор вкладок (tab control)</i> используется для добавления к диалоговому окну страниц (вкладок). Далее, в параграфе “Создание диалоговых окон с вкладками”, вы узнаете, как создавать диалоговое окно с вкладками с помощью MFC
 Custom Control	<i>Пользовательский элемент (Custom control)</i> – элемент управления, созданный и запрограммированный пользователем (или независимым поставщиком). Можно установить позицию и размер пользовательского элемента управления, но он будет отображаться как простой прямоугольник. Его фактический вид и поведение не проявятся до запуска программы. Создание специальных элементов управления рассмотрено в справочной системе

## Порядок обхода элементов управления

После создания и размещения элементов управления в диалоговом окне необходимо задать для них *порядок обхода*. При этом каждому элементу управления задается уникальный номер в общей сплошной последовательности (начиная с 1). Порядок обхода управляет последовательностью получения фокуса ввода элементами управления с установленным свойством Tab Stop при нажатии клавиш Tab или Shift+Tab.

- Нажатие клавиши Tab перемещает фокус ввода на *следующий* элемент управления в соответствии с порядком обхода.
- Нажатие клавиш Shift+Tab перемещает фокус на *предыдущий* элемент управления.

Когда фокус ввода находится на конкретном элементе управления, последний отвечает за ввод данных с клавиатуры. Если переместить фокус ввода на кнопку, то можно выполнить команду кнопки, нажав клавишу пробела. Перемещение фокуса на флажок или переключатель позволяет осуществить

выбор этих элементов нажатием клавиши пробела. Перемещая фокус в редактируемое поле, можно отредактировать текст. Таким образом, свойство Tab Stop позволяет без использования мыши вводить информацию в диалоговое окно.

Если элементу управления задано свойство Group, то данный элемент управления и все следующие за ним в порядке обхода принадлежат одной *группе*. Однако если следующий элемент управления также имеет свойство Group, то он начинает новую группу. Если несколько переключателей принадлежат одной и той же группе, то при каждом щелчке на одном из них автоматически удаляется отметка с предыдущего выбранного переключателя внутри группы (при условии, что они имеют свойство Auto). Можно использовать и клавиши со стрелками, чтобы переместить отметку выбора от одного переключателя к другому внутри группы.

Элемент управления, используемый как метка (надпись), должен предшествовать в порядке обхода связанному с ним полю. Если надпись элемента управления содержит подчеркнутый символ (т. е. этому символу предшествует & в определении элемента), то нажатие клавиши Alt и этого символа перемещает фокус в следующее в порядке обхода поле.

Для определения порядка обхода, выберите команду Tab Order в меню Format. После этого редактор диалоговых окон размещает номера на элементах управления, задавая порядок обхода, первоначально соответствующий порядку добавления элементов управления. Для изменения порядка обхода последовательно щелкните на каждом элементе управления в задаваемом вами порядке.

Диалоговое окно создано. Результат можно сохранить, выбрав команду Save All в меню File или щелкнув на кнопке Save All панели инструментов Toolbar. Окно диалогового редактора пока оставьте открытым.

## Управление диалоговым окном

Создадим класс, управляющий диалоговым окном Text Properties. Щелкните в окне редактора диалоговых окон, чтобы активировать его, и из контекстного меню редактируемого окна выберите команду Add Class.... Visual Studio распознает, что класс диалогового окна Text Properties еще не определен, и автоматически отобразит диалоговое окно создания класса. Обратите внимание: в списке Dialog ID выбран идентификатор диалогового окна Text Properties IDD\_TEXTPROP, а в списке Base Class – класс CDialog. CDialog – базовый MFC-класс для управления диалоговыми окнами. Создаваемый класс порождается непосредственно от класса CDialog. Когда мастер Class Wizard создает класс диалогового окна, то он сохраняет идентификатор IDD\_TEXTPROP ресурса диалогового окна Text Properties в определении класса (определяет элемент перечисления IDD\_TEXTPROP). Когда создается объект класса диалогового окна, конструктор класса передает идентификатор конструктору класса CDialog, сохраняющему данное значение. Наконец, чтобы для отображения окна использовался соответствующий ресурс шаблона диалогового окна, при отображении диалогового окна функция CDialog::DoModal передает идентификатор системе.

Имя нового класса CTextprop введите в поле Name, и мастер автоматически введет имя файла Textprop.cpp в область File name. Реализация класса CTextprop помещена в файл Textprop.cpp, а определение класса CTextprop – в файл Textprop.h (для изменения имен файлов щелкните на кнопке Change...). Для других установок оставьте стандартные значения. Чтобы создать исходные файлы, щелкните на кнопке OK. После создания нового класса в него необходимо добавить переменные, которые хранят состояние элементов управления окна Text Properties. При отображении диалогового окна в процессе выполнения программы MFC автоматически передает значение каждой из переменных соответствующему элементу управления. Например, при определении целочисленной переменной, связанной с полем, в случае, если переменная при отображении диалогового окна содержит значение 1, в соответствующем поле также появится 1.

При закрытии диалогового окна щелчком на кнопке OK значения, содержащиеся внутри каждого элемента управления, записываются обратно в соответствующие переменные. Если закрыть диалоговое окно щелчком на кнопке Cancel, то значения элементов управления *не* записываются в переменные.

Тип переменной, определяемой для заданного элемента управления, и работа механизма обмена зависят от типа элемента управления (ниже будут кратко описаны несколько элементов управления разных типов). MFC автоматически передает значения переменных в элементы управления при первом открытии диалогового окна, а также обратно при его закрытии. Можно заставить MFC выполнять обмен данными между элементами управления и переменными в любое время при отображении диалогового окна, вызывая из функции-члена класса диалогового окна функцию `CWnd::UpdateData`.

Откройте вкладку Class View, чтобы определить переменные для программы FontView. Так как класс `CTextProp` был только что определен, то он уже отображается в списке классов программы. Щелкните правой кнопкой мыши на классе `CTextProp`. В контекстном меню выберите пункт Add, а в подменю – пункт Variable. Чтобы задать переменную для флажка Bold, выберите в открывшемся окне идентификатор `IDC_BOLD`. Установите флажок Control Variable, в поле Name введите имя переменной `m_Bold`. Выберите значение Value в списке Category. При выборе типа `BOOL` в списке Type создается переменная, которой присваивается значение `TRUE` или `FALSE`. Аналогичным образом определите:

- для флажка Italic (`IDC_ITALIC`) переменную с именем `m_Italic`;
- для флажка Underlined (`IDC_UNDERLINED`) переменную с именем `m_Underlined`.

Создадим переменные для переключателей. Переменная назначается для *группы* позиций переключателя, а не для каждой из них. Перед отображением диалогового окна Text Properties переменной присваивается номер той позиции переключателя внутри группы, которая отмечена первоначально: 0 означает *первая позиция* переключателя в группе (в порядке обхода), 1 – *вторая* и т. д. Когда диалоговое окно Text Properties открывается первый раз, MFC отмечает (выбирает) указанную позицию переключателя (если присвоить переменной значение -1, то в группе не будет отмечена *ни одна позиция* переключателя). Когда пользователь щелкает на кнопке OK, MFC записывает в переменную номер отмеченной позиции переключателя в группе, а если *ни одна* позиция переключателя не отмечена, то значение -1. В списке Control IDs показаны идентификаторы *первых* позиций переключателей каждой из двух групп: `IDC_LEFT` – для группы, начинающейся с позиции переключателя Left, и `IDC_VARIABLE` – с позиции переключателя Variable. Сначала задайте переменную для группы переключателей `IDC_LEFT`. В диалоговом окне Add Member Variable введите имя переменной `m_Align`, тип переменной `int`, категорию Value. (Переменная типа `int` объявляется для хранения номера позиции переключателя.) Аналогичным образом назначьте переменную `m_Pitch` для группы позиций переключателя `IDC_VARIABLE`.

Для поля `IDC_SPACING` задайте переменную `m_Spacing`. В диалоговом окне оставьте в списке Category значение Value, выберите значение `int` вместо стандартного типа переменной `CString`, чтобы текстовая строка, отображаемая в поле, могла быть преобразована в целое число и из целого числа. Поля Minimal Value и Maximum Value, задающие диапазон допустимых значений для выбранной переменной. Если в них ввести числа, то после щелчка на кнопке OK библиотека MFC проверяет, находится ли введенное при работе с окном значение внутри заданного диапазона. Если значение не попадает в этот диапазон, то MFC выводит предупреждение и диалоговое окно остается открытым. Поскольку разрешается установить только одинарный, двойной или тройной интервал между строками, то для элемента управления `IDC_SPACING` введите 1 в поле Minimal Value и 3 в поле Maximum Value. Остается определить объект-член для управления полем. Для этого в списке Control IDs выберите идентификатор `IDC_SPACING`. Для одного элемента управления можно определять сразу две переменные при условии, что они принадлежат различным категориям, которые задаются значениями из списка Category. Введите имя `m_SpacingEdit` в поле Name. В списке Category выберите пункт Control. Теперь переменная является членом класса `CEdit` (а не переменной базового типа). Класс `CEdit` предоставляет набор функций для управления полем (см. ниже пример использования одной из них классом диалогового окна).

## Обработчики сообщений

В класс диалогового окна необходимо добавить обработчики сообщений, так как диалоговому окну передаются различные сообщения о событиях. Для программы FontView необходимы обработчики сообщений, посылаемых при первом открытии диалогового окна, его перерисовке, щелчке на флажке или переключателе. Чтобы определить требуемые обработчики сообщений, откройте вкладку Class View и из контекстного меню класса CTextprop выберите команду Properties. Для определения первых двух обработчиков сообщений, выберите раздел Overrides. Выберите функцию OnInitDialog и переопределите ее. Будет создана функция с именем OnInitDialog. Во вкладке Class View из контекстного меню класса CTextprop выберите пункт Add, а из подменю – пункт Function. Создайте функцию OnPaint типа void в классе CTextprop. Не указывайте для этой функции переменных-параметров. Каждый раз, когда потребуется отобразить или перерисовать диалоговое окно, будет вызываться эта функция.

Постройте обработчик сообщения для каждого флажка или переключателя, который будет получать управление всякий раз при выборе или отмене выбора элемента управления. Создайте следующие обработчики сообщений в классе CTextprop:

Табл. 15.3. Обработчики сообщения нажатия левой кнопки мыши (BN\_CLICKED)

Идентификатор объекта	Имя обработчика сообщения
IDC_BOLD	OnBold
IDC_CENTER	OnCenter
IDC_FIXED	OnFixed
IDC_ITALIC	OnItalic
IDC_LEFT	OnLeft
IDC_RIGHT	OnRight
IDC_UNDERLINE	OnUnderline
IDC_VARIABLE	OnVariable

Обработчики сообщений OnUpdate в данном случае не нужны.

В заключение определим функцию, которая получает управление при изменении содержимого поля IDC\_SPACING. В разделе Events окна Properties откройте пункт IDC\_SPACING и выберите подпункт EN\_CHANGE (это сообщение, посылаемое при каждом изменении текста). Создайте обработчик этого сообщения.

## MFC-классы и функции для элементов управления и диалоговых окон

В библиотеке MFC предусмотрены классы, позволяющие манипулировать элементами управления различных типов (табл. 15.4). Большинство этих классов порождаются от класса CWnd (некоторые – от других классов элементов управления). Функции-члены этих классов можно применять для получения информации или выполнения операций над элементом управления в диалоговом окне.

Например, в программе FontView функция LimitText класса CEdit используется для ограничения количества вводимых символов. Так как каждый из классов управления порождается (прямо или косвенно) от класса CWnd, то в число вызываемых функций-членов входят также функции, определенные в классе CWnd. Некоторые из них полезны при работе с элементами управления (например, чтобы сделать элемент управления доступным или недоступным, можно вызвать функцию CWnd::EnableWindow). Полный список функций-членов отдельных классов элементов управления или класса CWnd приведен в справочной системе.

Табл. 15.4. MFC-классы для манипулирования элементами управления

MFC-класс	Тип и описание элемента управления
CAnimateCtrl	Анимация
CButton	Кнопки, флажки, переключатели, рамки
CBitmapButton	Кнопки, отображающие растровые рисунки
CComboBox	Поле со списком
CComboBoxEx	Расширенное поле со списком
CDateTimeCtrl	Элемент Дата/Время
CEdit	Поле
CHeaderCtrl	Заголовок – поле, размещаемое над столбцами таблицы и содержащее заголовки столбцов с настраиваемой шириной (Windows Explorer выводит его при отображении детальной информации о файлах)
CHotKeyCtrl	Горячая клавиша
CIPAddressControl	IP-адрес
CListBox	Список
CCheckListBox	Список с флажками. Каждый пункт содержит флажок. Например, программа Setup в Visual Studio использует списки с флажками для предоставления пользователю возможности выбора устанавливаемых компонентов
CdragListBox	Перемещаемый список. Позволяет пользователю перемещать отдельные пункты списка для их переупорядочения
ClistCtrl	Поле списка
CMonthCalCtrl	Календарь по месяцам
ColeControl	Элементы управления ActiveX (ранее известные как элементы управления OLE, отсюда и название класса) – многократно используемые компоненты программного обеспечения. Предоставляют программам широкий спектр услуг (гл. 25)
CProgressCtrl	Индикатор
CReBarCtrl	Переключаемая панель. Эти элементы управления описаны во врезке “Диалоговые и переключаемые панели” (см. гл. 14)
CRichEditCtrl	Расширенное поле
CScrollBar	Полосы прокрутки: вертикальная или горизонтальная
CSliderCtrl	Регулятор
CSpinButtonCtrl	Счетчик
CStatic	Надписи, рисунки и рамки
CStatusBarCtrl	Строка состояния. Использование класса CStatusBarCtrl – <i>альтернатива</i> способу создания строк состояния (см. гл. 14)
CTabCtrl	Набор вкладок. Использование класса CTabCtrl – <i>альтернатива</i> способу создания диалоговых окон со вкладками в программах MFC (далее в этой главе)
CToolBarCtrl	Панель инструментов. Использование класса CToolBarCtrl – <i>альтернатива</i> способу создания панелей инструментов в программах MFC (см. гл. 14)
CTooltipCtrl	Всплывающая подсказка. При создании панели инструментов (см. гл. 14) MFC <i>автоматически</i> реализует всплывающие подсказки
CTreeCtrl	Дерево



Чтобы использовать одну из этих функций при работе с элементами управления в диалоговом окне, необходимо создать объект (*постоянный*) класса элемента управления, закрепляемый за элементом управления. Можно также вызвать функцию `CWnd::GetDlgItem` для объекта диалогового окна, чтобы получить указатель на *временный* объект для элемента управления внутри диалогового окна. Например, в программе `FontView` вместо создания постоянного объекта для поля можно из обработчика `OnInitDialog` вызвать функцию `GetDlgItem`, чтобы получить временный объект, для которого затем вызывается функция `LimitText`. Обратите внимание: так как функция `GetDlgItem` возвращает указатель класса `CWnd`, возвращаемое значение необходимо преобразовать в указатель на соответствующий класс элемента управления. Так как указатель временный, он используется только во время обработки текущего сообщения и в дальнейшем *не* сохраняется.

```
((CEdit *)GetDlgItem (IDC_SPACING))->LimitText (1);
```

Заметьте: функция `GetDlgItem` также использовалась для установки значения объекта `m_RectSample` во фрагменте, добавленном в обработчик `OnInitDialog` выше в этом параграфе.

Можно изменить диалоговое окно `Text Properties` в программе `FontView` так, чтобы переключатель `Bold` (который имеет идентификатор `IDC_BOLD`) был не отмечен и недоступен, когда переключатель `Variable` (`IDC_VARIABLE`) выбран. (Вспомните: пропорциональный шрифт `System` не может быть полужирным.) Этот код необходимо поместить в функции-члены `OnInitDialog`, `OnVariable` и `OnFixed` класса диалогового окна `CTextprop`. Для получения объекта `CButton`, связанного с переключателем `Bold`, можно использовать любой описанный выше метод. Чтобы выбрать переключатель или отменить его выбор, вызовите функцию `CButton::SetCheck`, а для разрешения или блокирования доступа к переключателю – функцию `CWnd::EnableWindow`.

Элементы управления обычно связаны с диалоговыми окнами, но их можно отображать внутри любого окна программы. Например, можно отобразить один или несколько элементов управления прямо внутри окна представления. Однако окно представления не имеет шаблона для создания и отображения элементов управления. Необходимо добавить код для явного создания, размещения и отображения элемента управления. Для этого объявите объект соответствующего MFC-класса элемента управления (можно сделать его членом класса главного окна). Вызовите функцию-член класса элемента управления `Create`, чтобы отобразить его, задав требуемый размер, позицию и другие атрибуты. Обратите внимание: элемент управления является дочерним по отношению к окну, в котором он отображается. Если окно представления в основном содержит лишь набор элементов управления, лучше всего породить класс представления от класса `CFormView` и использовать шаблон диалогового окна (гл. 16) вместо создания по отдельности элементов управления.

Класс диалогового окна порождается от класса `CDialog`, который, в свою очередь, порождается от класса `CWnd`. Оба эти класса предоставляют функции для управления диалоговыми окнами, которые можно вызвать из класса диалогового окна. Эти функции можно использовать *в дополнение* к функциям MFC-классов элементов управления (таких, как класс `CEdit`). Некоторые из данных функций воздействуют на отдельные элементы управления, находящиеся в диалоговом окне, другие – на группы элементов управления, а остальные – непосредственно на диалоговые окна. Обычно эти функции вызываются из обработчиков сообщений класса диалогового окна, получающих управление при отображении данного окна. Функции класса `CWnd` перечислены в табл. 15.5, а функции `CDialog` – в табл. 15.6. Подробная информация о них приведена в справочной системе.

## Управление диалоговым окном *Text Properties*

Для завершения создания класса `CTextprop` откройте файл `Textprop.h` и добавьте два следующих определения в начало файла.

```
enum {ALIGN_LEFT, ALIGN_CENTER, ALIGN_RIGHT};
enum {PITCH_VARIABLE, PITCH_FIXED};
```

Табл. 15.5. Функции класса CWnd для управления диалоговыми окнами

Функция	Назначение
CheckDlgButton	Выбирает или отменяет выбор флажка или переключателя
CheckRadioButton	Выбирает указанную позицию переключателя и отменяет выбор остальных позиций указанного переключателя
DlgDirList	Добавляет имена файлов, каталогов или дисков в список
DlgDirListComboBox	Добавляет имена файлов, каталогов или дисков в поле со списком
DlgDirSelect	Получает имя текущего, выбранного в списке, файла, каталога или диска
DlgDirSelectComboBox	Получает имя текущего, выбранного в поле со списком, файла, каталога или диска
GetCheckedRadioButton	Возвращает идентификатор выбранной позиции переключателя в указанном переключателе
GetDlgItem	Возвращает указатель на временный объект для заданного элемента управления
GetDlgItemInt	Возвращает числовое значение, представленное в текстовом виде в указанном элементе управления
GetDlgItemText	Получает текст, отображаемый внутри элемента управления
GetNextDlgGroupItem	Возвращает указатель на временный объект для следующего (или предыдущего) элемента управления внутри группы
GetNextDlgTabItem	Возвращает указатель на временный объект для следующего элемента управления (в порядке обхода), у которого задано свойство Tab Stop
IsDlgButtonChecked	Возвращает статус отметки флажка или позиции переключателя
SendDlgItemMessage	Посылает сообщение элементу управления
SetDlgItemInt	Преобразовывает целое число в текст и передает его элементу управления
SetDlgItemText	Задаст текст, отображаемый элементом управления

Таблица 15.6. Функции класса CDialog для манипулирования диалоговыми окнами и элементами управления

Функция	Назначение
EndDialog	Закрывает модальное диалоговое окно
GetDefID	Возвращает идентификатор кнопки внутри диалогового окна, которая в данный момент является кнопкой по умолчанию
GotoDlgCtrl	Передаст фокус ввода указанному элементу управления внутри диалогового окна
MapDialogRect	Преобразует координаты элемента управления из единиц диалогового окна в экранные
NextDlgCtrl	Передаст фокус ввода следующему в порядке обхода элементу управления
PrevDlgCtrl	Передаст фокус ввода предыдущему в порядке обхода элементу управления
SetDefID	Делает указанную кнопку кнопкой по умолчанию

В этих определениях перечисления используются для обращения к переключателям. Константы первого перечисления (равные 0, 1 и 2) – это ссылки на номера переключателей группы Alignment диалогового окна Text Properties, а константы второго перечисления – это ссылки на номера пере-

ключателей в группе Pitch. Затем добавьте определение защищенной переменной `m_RectSample` в определение класса `CTextprop`.

```
// Диалог CTextprop

class CTextprop : public CDialog
{
    DECLARE_DYNAMIC(CTextprop)

protected:
    RECT m_RectSample;
```

В функцию `OnInitDialog` в файле `Textprop.cpp` добавьте выделенные полужирным строки:

```
BOOL CTextprop::OnInitDialog()
{
    CDialog::OnInitDialog();

    // TODO: Добавьте сюда собственный код обработчика

    GetDlgItem (IDC_SAMPLE)->GetWindowRect (&m_RectSample);
    ScreenToClient (&m_RectSample);

    m_SpacingEdit.LimitText (1);

    return TRUE;
    // Возвращает TRUE, если не выбран элемент управления
    // Исключение: Страницы свойств элементов OCX возвращают FALSE
}
```

При открытии диалогового окна `Text Properties` непосредственно перед его отображением получает управление функция `OnInitDialog`. Первый добавленный оператор сохраняет экранные координаты рамки `Sample` в переменной `m_RectSample`. Второй – преобразует координаты экрана в координаты окна (т. е. в координаты точек внутри диалогового окна относительно его левого верхнего угла). Переменная `m_RectSample` отображает текст внутри рамки. Третий добавленный оператор использует переменную `m_SpacingEdit` класса `CEdit` для вызова функции `LimitText` класса `CEdit`. Для предотвращения ввода более чем одного символа в поле `IDC_SPACING` в функцию `LimitText` передается значение 1.

Далее необходимо завершить программирование обработчиков сообщений, получающих управление при каждом щелчке пользователя на флажке. В файле `Textprop.cpp` дополните функции следующим кодом:

```
// Обработчики сообщений класса CTextprop

void CTextprop::OnBnClickedBold()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_Bold = !m_Bold;
    InvalidateRect (&m_RectSample);
    UpdateWindow ();
}

void CTextprop::OnBnClickedCenter()
{
    // TODO: Добавьте сюда собственный код обработчика
    if (IsDlgButtonChecked (IDC_CENTER))
```

```

    {
        m_Align = ALIGN_CENTER;
        InvalidateRect (&m_RectSample);
        UpdateWindow ();
    }
}

void CTextprop::OnBnClickedFixed()
{
    // TODO: Добавьте сюда собственный код обработчика
    if (IsDlgButtonChecked (IDC_FIXED))
    {
        m_Pitch = PITCH_FIXED;
        InvalidateRect (&m_RectSample);
        UpdateWindow ();
    }
}

void CTextprop::OnBnClickedItalic()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_Italic = !m_Italic;
    InvalidateRect (&m_RectSample);
    UpdateWindow ();
}

void CTextprop::OnBnClickedLeft()
{
    // TODO: Добавьте сюда собственный код обработчика
    if (IsDlgButtonChecked (IDC_LEFT))
    {
        m_Align = ALIGN_LEFT;
        InvalidateRect (&m_RectSample);
        UpdateWindow ();
    }
}

void CTextprop::OnBnClickedRight()
{
    // TODO: Добавьте сюда собственный код обработчика
    if (IsDlgButtonChecked (IDC_RIGHT))
    {
        m_Align = ALIGN_RIGHT;
        InvalidateRect (&m_RectSample);
        UpdateWindow ();
    }
}

void CTextprop::OnBnClickedUnderlined()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_Underlined=!m_Underlined;
    InvalidateRect (&m_RectSample);
}

```

```

        UpdateWindow ();
    }

void CTextprop::OnBnClickedVariable()
{
    // TODO: Добавьте сюда собственный код обработчика
    if (IsDlgButtonChecked (IDC_VARIABLE))
    {
        m_Pitch = PITCH_VARIABLE;
        InvalidateRect (&m_RectSample);
        UpdateWindow ();
    }
}

void CTextprop::OnEnChangeSpacing()
{
    // TODO: Добавьте сюда собственный код обработчика
    int Temp;
    Temp = (int) GetDlgItemInt (IDC_SPACING);
    if (Temp > 0 && Temp < 4)
    {
        m_Spacing = Temp;
        InvalidateRect (&m_RectSample);
        UpdateWindow ();
    }
}

```

Рассмотрим первый обработчик. Первый оператор переключает значение переменной `m_Bold` между `TRUE` и `FALSE`. Таким образом, если флажок *не* выбран (и значение `m_Bold` равно `FALSE`), то щелчок на этом элементе управления приведет к его отметке и установке значения переменной `m_Bold` равным `TRUE`. Если флажок *выбран*, щелчок на нем приведет к удалению отметки и установке значения переменной равным `FALSE`. Второй добавленный оператор делает недействительной (т. е. отмечает для перерисовки) часть диалогового окна, занятую рамкой `Sample`, а третий – приводит к непосредственному вызову функции `OnPaint` класса диалогового окна. Затем функция `OnPaint` отображает текст в новом формате (код для этого мы напишем позднее).

Рассмотрим обработчики сообщений для переключателей. Каждая из этих функций вызывает функцию `CWnd::IsDlgButtonChecked`, чтобы убедиться, что переключатель выбран. (Обработчик вызывается при каждом щелчке на переключателе и его выборе, а также при нажатии на клавишу табуляции, когда фокус ввода перемещается на переключатель, но последний *не выбирается*.) Если переключатель установлен, то обработчик присваивает его номер соответствующей переменной (`m_Align` или `m_Pitch`). Ниже будет описано использование этих переменных.

Функция `OnEnChangeSpacing` вызывает функцию `CWnd::GetDlgItemInt`, чтобы получить содержимое поля в виде целочисленного значения. Если оно находится в допустимом диапазоне, функция сохраняет его в переменной `m_Spacing` и перерисовывает текст в поле.

В заключение необходимо добавить код в функцию `OnPaint`, вызываемую, когда один из обработчиков сообщения признает недействительной область диалогового окна или когда требуется перерисовать диалоговое окно из-за внешнего события (например, перемещения перекрывающего его окна). Функция `OnPaint` служит только для перерисовки трех строк образца текста в рамке `Sample`. Перерисовывать элементы управления *не* нужно – они перерисовывают себя сами. (Описание различных методов, используемых для вывода строк текста, приведено в гл. 18.) Если класс окна порождается от класса `CView`, то последний предоставляет обработчик сообщения `OnPaint`, вызывающий функцию `OnDraw` класса окна представления. Однако так как класс диалогового окна не

порождается от класса CView, то перерисовку этого окна выполняет сама функция OnPaint. Обратите внимание: функция OnPaint должна создать объект контекста устройства, принадлежащий MFC-классу CPaintDC (а не классу CClientDC, используемому для создания объектов контекста устройства в других функциях).

```
void CTextprop::OnPaint(void)
{
    CPaintDC dc(this);    //контекст устройства для рисования

    // TODO: Добавьте сюда собственный код обработчика
    CFont Font;
    LOGFONT LF;
    int LineHeight;
    CFont *PtrOldFont;
    int X, Y;

    // заполнение структуры LF свойствами
    // стандартного системного шрифта
    CFont TempFont;
    if (m_Pitch == PITCH_VARIABLE)
        TempFont.CreateStockObject (SYSTEM_FONT);
    else
        TempFont.CreateStockObject (SYSTEM_FIXED_FONT);
    TempFont.GetObject (sizeof (LOGFONT), &LF);

    // инициализация полей структуры LF
    if (m_Bold) LF.lfWeight = FW_BOLD;
    if (m_Italic) LF.lfWeight = 1;
    if (m_Underlined) LF.lfUnderline = 1;

    // создание и выбор шрифта
    Font.CreateFontIndirect (&LF);
    PtrOldFont = dc.SelectObject (&Font);

    // выбор выравнивания
    switch (m_Align)
    {
    case ALIGN_LEFT:
        dc.SetTextAlign (TA_LEFT);
        X = m_RectSample.left + 5;
        break;
    case ALIGN_CENTER:
        dc.SetTextAlign (TA_CENTER);
        X = (m_RectSample.left + m_RectSample.right) / 2;
        break;
    case ALIGN_RIGHT:
        dc.SetTextAlign (TA_RIGHT);
        X = m_RectSample.right + 5;
        break;
    }

    // установка режима отображения фона
    dc.SetBkMode (TRANSPARENT);
}
```

```

// вывод строк текста
LineHeight = LF.lfHeight * m_Spacing;
Y = m_RectSample.top + 15;
dc.TextOut (X, Y, "AaBbCcDdEeFfGg");
Y += LineHeight;
dc.TextOut (X, Y, "HhIiJjKkLlMmNn");
Y += LineHeight;
dc.TextOut (X, Y, "OoPpQqRrSsTtUu");

// отмена выбора шрифта
dc.SelectObject (PtrOldFont);
}

```

Функция OnPaint определяет свойства шрифта System (пропорциональный или моноширинный системный шрифт System в зависимости от значения m\_Pitch), модифицирует эти свойства в соответствии со значениями переменных m\_Bold, m\_Italic, m\_Underline и создает новый шрифт (на основе System), используемый для вывода текста. Затем задается выравнивание текста согласно значению переменной m\_Align. В заключение вызывается функция CDC::TextOut для вывода текста (значение переменной m\_Spacing определяет междустрочный интервал).

Стандартные обработчики сообщений, получающие управление при закрытии диалогового окна, предоставляются MFC-классом CDialog. После щелчка на кнопке ОК управление получает функция CDialog::OnOK, которая вызывает функцию CWnd::UpdateData, чтобы проверить и передать данные из элементов управления диалогового окна в связанные с ними переменные, а затем – функцию CDialog::EndDialog для закрытия диалогового окна. Если нажать кнопку Cancel, то управление получит функция CDialog::OnCancel, которая просто вызывает функцию CDialog::EndDialog для закрытия диалогового окна. Если перед закрытием диалогового окна необходимо выполнить дополнительные действия, то определите собственные обработчики сообщений. Для определения собственной версии обработчика нажатия кнопки ОК выберите идентификатор объекта IDOK и сообщение BN\_CLICKED. Чтобы определить собственную версию обработчика нажатия кнопки Cancel, выберите идентификатор объекта IDCANCEL и сообщение BN\_CLICKED. В каждом обработчике вызывается одноименная функция класса CDialog, выполняющая требуемую стандартную обработку. В функциях, сгенерированных Visual Studio, необходимо добавить собственные операторы перед обращением к функции базового класса.

## Отображение диалогового окна

Добавим в код операторы, которые создают объект класса диалогового окна и отображают его на экране. Первое действие – добавление команды меню, открывающей диалоговое окно Text Properties. Для этого откройте вкладку Resource View для проекта FontView и выполните двойной щелчок на идентификаторе IDR\_MAINFRAME в разделе Menu графа. В редакторе меню удалите меню File и Edit. Затем добавьте меню Text слева от меню Help. Свойства этого меню приведены в табл. 15.7. Добавьте к новому меню Text команду Text Properties, разделитель и команду Exit (см табл. 15.7).

Добавьте сочетание клавиш для команды Text Properties в таблицу горячих клавиш IDR\_MAINFRAME. Используйте метод, описанный в предыдущих главах, определяя одинаковый идентификатор ID\_TEXT\_FORMAT и для команды Text Properties, и для сочетания клавиш Ctrl+T. Определите обработчик сообщения, получающий управление при выборе команды Text Properties (или нажатии клавиш Ctrl+T). В программе FontView выбор класса для обработки команды Text Properties несколько произволен. Однако в реальных текстовых процессорах подобные команды непосредственно воздействуют на данные, сохраняемые в документе (а не на определенное представление документа

или на приложение в целом). Поэтому рекомендуется обрабатывать команду с помощью класса документа. Когда ресурсы программы будут определены, сохраните результаты работы, выбрав команду Save All в меню File или щелкнув в стандартной панели инструментов на кнопке Save All.

Табл. 15.7. Свойства меню Text и его элементов

Идентификатор (ID)	Надпись (Caption)	Другие свойства
Отсутствует	&Text	Ниспадающее меню
ID_TEXT_FORMAT	&Text Properties...\tCtrl+T	Отсутствуют
Отсутствует	Отсутствует	Разделитель
ID_APP_EXIT	E&xit	Отсутствуют

Остается внести необходимые дополнения в код класса документа и класса представления для отображения диалогового окна и использования его данных при выводе текста в окне представления программы. Сначала добавим определения переменных в начало определения класса CFontViewDoc в файле FontViewDoc.h.

```
class CFontViewDoc : public CDocument
{
public:
    BOOL m_Bold;
    BOOL m_Italic;
    BOOL m_Underlined;
    int m_Align;
    int m_Pitch;
    int m_Spacing;

protected: // используются только для сериализации
    CFontViewDoc();
    DECLARE_DYNCREATE(CFontViewDoc)
```

Определяемые переменные сохраняют значения параметров форматирования документа и соответствуют переменным класса диалогового окна с такими же именами. Добавьте строки инициализации этих переменных в класс CFontViewDoc в файле FontViewDoc.cpp. Здесь задается отображение текста без полужирного, курсивного или подчеркнутого начертаний с использованием пропорционального шрифта, выравниванием по левому краю и одинарным междустрочным интервалом.

```
CFontViewDoc::CFontViewDoc()
{
    // TODO: добавьте сюда собственный код конструктора
    m_Bold = FALSE;
    m_Italic = FALSE;
    m_Underlined = FALSE;
    m_Align = ALIGN_LEFT;
    m_Pitch = PITCH_VARIABLE;
    m_Spacing = 1;
}
```

Добавьте оператор include для файла заголовков Textprop.h в файл FontViewDoc.cpp, чтобы объявление класса диалогового окна было доступно в файле FontViewDoc.cpp.



```

// FontViewDoc.cpp : реализация класса CFontViewDoc
//

#include "stdafx.h"
#include "FontView.h"

#include "FontViewDoc.h"

#include "Textprop.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

```

Добавьте следующий фрагмент в функцию `OnTextFormat`, сгенерированную Visual Studio. В добавленном коде создается экземпляр класса диалогового окна `CTextprop`. Для модального диалогового окна этот объект обычно определяется как локальная переменная, удаляемая сразу после завершения функции, в которой она определена. Затем значения переменных, сохраняющих информацию о форматировании, копируются в соответствующие объекты диалогового окна, чтобы текущие значения этих переменных использовались при первом его открытии (при первоначальном отображении диалогового окна MFC автоматически передает значения переменных объекта соответствующим элементам управления). Далее вызывается функция `DoModal` класса `CDialog`, отображающая диалоговое окно. Пока оно открыто, функция `DoModal` не завершается. Если его закрыть, щелкнув на кнопке OK, то функция `DoModal` возвращает значение `IDOK`. В этом случае функция `OnTextFormat` передает новые значения переменных объекта диалогового окна обратно переменным объекта документа, а затем вызывает функцию `UpdateAllViews`, инициирующую перерисовку окна представления и передающую управление функции `OnDraw` класса представления. Если диалоговое окно закрыть, щелкнув на кнопке OK, то MFC автоматически проверит содержимое поля `Line Spacing` и передаст текущее содержимое элементов управления обратно соответствующим переменным объекта диалогового окна. При нажатии кнопки Cancel функция `DoModal` возвращает значение `IDCANCEL`. В этом случае переменные объекта документа остаются неизменными, а MFC содержимое элементов управления не проверяет и не сохраняет.

```

void CFontViewDoc::OnTextFormat()
{
    // TODO: Добавьте сюда собственный код обработчика

    // объявление объекта класса диалогового окна
    CTextprop TextPropDlg;

    // инициализация переменных класса
    TextPropDlg.m_Bold = m_Bold;
    TextPropDlg.m_Italic = m_Italic;
    TextPropDlg.m_Underlined = m_Underlined;
    TextPropDlg.m_Align = m_Align;
    TextPropDlg.m_Pitch = m_Pitch;
    TextPropDlg.m_Spacing = m_Spacing;

    // отображение диалогового окна
    if (TextPropDlg.DoModal () == IDOK)
    {
        // сохранение значений, полученных из диалогового окна
        m_Bold = TextPropDlg.m_Bold;

```

```

        m_Italic = TextPropDlg.m_Italic;
        m_Underlined = TextPropDlg.m_Underlined;
        m_Align = TextPropDlg.m_Align;
        m_Pitch = TextPropDlg.m_Pitch;
        m_Spacing = TextPropDlg.m_Spacing;
    }

    // перерисовка текста
    UpdateAllViews (NULL);
}

```

Откройте файл FontViewView.cpp и добавьте оператор include для включения файла заголовков TextProp.h, чтобы можно было использовать соответствующие стили и константы с номерами переключателей.

```

// FontViewView.cpp : реализация класса CFontViewView
//

#include "stdafx.h"
#include "FontView.h"

#include "FontViewDoc.h"
#include "FontViewView.h"

#include "Textprop.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

```

Получая управление, функция OnDraw перерисовывает строки текста в окне представления, используя новые параметры форматирования, сохраненные в объекте документа. Добавьте приведенный и выделенный ниже полужирным код для перерисовки строк в функцию OnDraw. Этот код напоминает код, добавленный в функцию OnPaint класса диалогового окна. Вместо простого вывода текста с использованием стандартного черного цвета мы вызываем функции ::GetSysColor и CDC::SetTextColor, устанавливающие цвет текста Window Font, выбранный в утилите Display в панели управления Windows (MFC автоматически устанавливает цвет фона окна, используя установки Windows).

```

// Прорисовка CFontViewView

void CFontViewView::OnDraw(CDC* pDC)
{
    CFontViewDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    // TODO: добавьте сюда собственный код рисования
    RECT ClientRect;
    CFont Font;
    LOGFONT LF;
    int LineHeight;
    CFont *PtrOldFont;
    int X, Y;
}

```

```

// заполнение структуры LF свойствами
// стандартного системного шрифта
CFont TempFont;
if (pDoc->m_Pitch == PITCH_VARIABLE)
    TempFont.CreateStockObject (SYSTEM_FONT);
else
    TempFont.CreateStockObject (SYSTEM_FIXED_FONT);
TempFont.GetObject (sizeof (LOGFONT), &LF);

// инициализация полей структуры LF
if (pDoc->m_Bold) LF.lfWeight = FW_BOLD;
if (pDoc->m_Italic) LF.lfItalic = 1;
if (pDoc->m_Underlined) LF.lfUnderline = 1;

// создание и выбор шрифта
Font.CreateFontIndirect (&LF);
PtrOldFont = pDC->SelectObject (&Font);

// установка выравнивания
GetClientRect (&ClientRect);
switch (pDoc->m_Align)
{
case ALIGN_LEFT:
    pDC->SetTextAlign (TA_LEFT);
    X = ClientRect.left + 5;
    break;
case ALIGN_CENTER:
    pDC->SetTextAlign (TA_CENTER);
    X = (ClientRect.left + ClientRect.right) / 2;
    break;
case ALIGN_RIGHT:
    pDC->SetTextAlign (TA_RIGHT);
    X = ClientRect.right + 5;
    break;
}

// установка цвета текста и режима фона
pDC->SetTextColor (::GetSysColor (COLOR_WINDOWTEXT));
pDC->SetBkMode (TRANSPARENT);

// вывод строк текста
LineHeight = LF.lfHeight * pDoc->m_Spacing;
Y = 5;
pDC->TextOut (X, Y, "Example text fragment - line one.");
Y += LineHeight;
pDC->TextOut (X, Y, "Example text fragment - line two.");
Y += LineHeight;
pDC->TextOut (X, Y, "Example text fragment - line three.");

// отмена выбора шрифта
pDC->SelectObject (PtrOldFont);
}

```

MFC отображает в строке заголовка программы FontView текст “Untitled – FontView”, позволяющий сделать ошибочный вывод о том, что программа создает документы. Чтобы установить более подходящий заголовок, добавьте в файле FontView.cpp вызов функции CWnd::SetWindowText в конец функции InitInstance.

```
m_pMainWnd->ShowWindow(SW_SHOW);
m_pMainWnd->UpdateWindow();

m_pMainWnd->SetWindowText ("System Font View");
return TRUE;
```

Используемая в этом фрагменте переменная m\_pMainWnd – это указатель на объект главного окна. Обращение к функции SetWindowText должно находиться *после* обращения к создающей главное окно функции ProcessShellCommand.

## Текст программы FontView

Разработка программы FontView завершена, теперь можете построить и запустить ее. В листингах 15.1—15.10 приведен текст программы FontView.

---

### Листинг 15.1

```
// FontView.h : главный заголовочный файл приложения FontView
//
#pragma once

#ifdef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"          // основные символы

// CFontViewApp:
// Смотрите реализацию этого класса в файле FontView.cpp
//

class CFontViewApp : public CWinApp
{
public:
    CFontViewApp();

    // Переопределения
public:
    virtual BOOL InitInstance();

    // Реализация
    afx_msg void OnAppAbout();
    DECLARE_MESSAGE_MAP()
};

extern CFontViewApp theApp;
```

---

### Листинг 15.2

```
// FontView.cpp : определяет поведение классов приложения
//
```

```

#include "stdafx.h"
#include "FontView.h"
#include "MainFrm.h"

#include "FontViewDoc.h"
#include "FontViewView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CFontViewApp

BEGIN_MESSAGE_MAP(CFontViewApp, CWinApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
    // Стандартные команды работы с файлами документов
    ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
END_MESSAGE_MAP()

// Конструктор класса CFontViewApp

CFontViewApp::CFontViewApp()
{
    // TODO: Добавьте сюда собственный код конструктора
    // Поместите все существенные команды инициализации в
    // функцию InitInstance
}

// Единственный объект класса CFontViewApp

CFontViewApp theApp;

// Инициализация CFontViewApp

BOOL CFontViewApp::InitInstance()
{
    CWinApp::InitInstance();

    // Инициализация библиотек OLE
    if (!AfxOleInit())
    {
        AfxMessageBox(IDP_OLE_INIT_FAILED);
        return FALSE;
    }
    AfxEnableControlContainer();
    // Стандартная инициализация.
    // Если вы не используете какие-то из функций, указанных ниже,
    // и хотите уменьшить размер конечного исполняемого модуля,
    // удалите соответствующие процедуры инициализации.
    // Измените строку-аргумент функции (реестровый ключ, под
    // которым хранятся ваши установки) на что-нибудь
    // запоминающееся, например, название вашей компании
    // или организации
    SetRegistryKey(_T("Local AppWizard-Generated Applications"));
}

```

```

LoadStdProfileSettings(4); // Загрузка установок из INI-файла
                             // (включая MRU)
// Регистрация шаблонов документов приложения. Они служат связью
// между документами, окнами документов и окнами представлений.
CSingleDocTemplate* pDocTemplate;
pDocTemplate = new CSingleDocTemplate(
    IDR_MAINFRAME,
    RUNTIME_CLASS(CFontViewDoc),
    RUNTIME_CLASS(CMainFrame), // Главное окно
                                // SDI-приложения
    RUNTIME_CLASS(CFontViewView));
AddDocTemplate(pDocTemplate);
// Поиск в командной строке команд оболочки, DDE, открытия
// файлов
CCommandLineInfo cmdInfo;
ParseCommandLine(cmdInfo);
// Выполнение команд, обнаруженных в командной строке.
// Вернет False, если приложение было запущено с /RegServer,
// /Register, /Unregserver or /Unregister.
if (!ProcessShellCommand(cmdInfo))
    return FALSE;
// Обновление и отображение единственного окна приложения
m_pMainWnd->ShowWindow(SW_SHOW);
m_pMainWnd->UpdateWindow();

m_pMainWnd->SetWindowText ("System Font View");
return TRUE;
}

// CAboutDlg - диалог, используемый в App About

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

// Данные для диалога
    enum { IDD = IDD_ABOUTBOX };

protected:
    virtual void DoDataExchange(CDataExchange* pDX);
    // Поддержка DDX/DDV

// Реализация
protected:
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
}

```

```

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
END_MESSAGE_MAP()

// Отображение диалогового окна
void CFontViewApp::OnAppAbout()
{
    CAboutDlg aboutDlg;
    aboutDlg.DoModal();
}

// Обработчики сообщений класса CFontViewApp

```

---

### Листинг 15.3

```

// FontViewDoc.h : интерфейс класса CFontViewDoc
//

#pragma once

class CFontViewDoc : public CDocument
{
public:
    BOOL m_Bold;
    BOOL m_Italic;
    BOOL m_Underlined;
    int m_Align;
    int m_Pitch;
    int m_Spacing;

protected: // используются только для сериализации
    CFontViewDoc();
    DECLARE_DYNCREATE(CFontViewDoc)

// Атрибуты
public:

// Операции
public:

// Переопределения
public:
    virtual BOOL OnNewDocument();
    virtual void Serialize(CArchive& ar);

// Реализация
public:
    virtual ~CFontViewDoc();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif
protected:

```

```

// Сгенерированные функции карты сообщений
protected:
    DECLARE_MESSAGE_MAP()
public:
    afx_msg void OnTextFormat();
};

```

---

#### Листинг 15.4

```

// FontViewDoc.cpp : реализация класса CFontViewDoc
//

#include "stdafx.h"
#include "FontView.h"

#include "FontViewDoc.h"

#include "Textprop.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CFontViewDoc

IMPLEMENT_DYNCREATE(CFontViewDoc, CDocument)

BEGIN_MESSAGE_MAP(CFontViewDoc, CDocument)
    ON_COMMAND(ID_TEXT_FORMAT, OnTextFormat)
END_MESSAGE_MAP()

// Конструктор/деструктор класса CFontViewDoc

CFontViewDoc::CFontViewDoc()
{
    // TODO: добавьте сюда собственный код конструктора
    m_Bold = FALSE;
    m_Italic = FALSE;
    m_Underlined = FALSE;
    m_Align = ALIGN_LEFT;
    m_Pitch = PITCH_VARIABLE;
    m_Spacing = 1;
}

CFontViewDoc::~CFontViewDoc()
{
}

BOOL CFontViewDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;

    // TODO: добавьте сюда код реинициализации
    // (SDI-документы будут использовать этот документ повторно)
}

```



```

        return TRUE;
    }

    // Сериализация CFontViewDoc

void CFontViewDoc::Serialize(CArchive& ar)
{
    if (ar.IsStoring())
    {
        // TODO: добавьте сюда собственный код сохранения
    }
    else
    {
        // TODO: добавьте сюда собственный код загрузки
    }
}

// Диагностика CFontViewDoc

#ifdef _DEBUG
void CFontViewDoc::AssertValid() const
{
    CDocument::AssertValid();
}

void CFontViewDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}
#endif //_DEBUG

// Команды CFontViewDoc

void CFontViewDoc::OnTextFormat()
{
    // TODO: Добавьте сюда собственный код обработчика

    // объявление объекта класса диалогового окна
    CTextprop TextPropDlg;

    // инициализация переменных класса
    TextPropDlg.m_Bold = m_Bold;
    TextPropDlg.m_Italic = m_Italic;
    TextPropDlg.m_Underlined = m_Underlined;
    TextPropDlg.m_Align = m_Align;
    TextPropDlg.m_Pitch = m_Pitch;
    TextPropDlg.m_Spacing = m_Spacing;

    // отображение диалогового окна
    if (TextPropDlg.DoModal () == IDOK)
    {
        // сохранение значений, полученных из диалогового окна
        m_Bold = TextPropDlg.m_Bold;
        m_Italic = TextPropDlg.m_Italic;
        m_Underlined = TextPropDlg.m_Underlined;
        m_Align = TextPropDlg.m_Align;
    }
}

```

```

        m_Pitch = TextPropDlg.m_Pitch;
        m_Spacing = TextPropDlg.m_Spacing;
    }

    // перерисовка текста
    UpdateAllViews (NULL);
}

```

---

### Листинг 15.5

```

// FontViewView.h : интерфейс класса CFontViewView
//

#pragma once

class CFontViewView : public CView
{
protected: // используются только для сериализации
    CFontViewView();
    DECLARE_DYNCREATE(CFontViewView)

// Атрибуты
public:
    CFontViewDoc* GetDocument() const;

// Операции
public:

// Переопределения
public:
    virtual void OnDraw(CDC* pDC);
    // переопределена для прорисовки этого вида

virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
protected:

// Реализация
public:
    virtual ~CFontViewView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Сгенерированные функции карты сообщений
protected:
    DECLARE_MESSAGE_MAP()
};

#ifdef _DEBUG // отладочная версия в файле FontViewView.cpp
inline CFontViewDoc* CFontViewView::GetDocument() const
{ return reinterpret_cast<CFontViewDoc*>(m_pDocument); }
#endif

```

---

## Листинг 15.6

```
// FontViewView.cpp : реализация класса CFontViewView
//

#include "stdafx.h"
#include "FontView.h"

#include "FontViewDoc.h"
#include "FontViewView.h"

#include "Textprop.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CFontViewView

IMPLEMENT_DYNCREATE(CFontViewView, CView)

BEGIN_MESSAGE_MAP(CFontViewView, CView)
END_MESSAGE_MAP()

// Конструктор CFontViewView

CFontViewView::CFontViewView()
{
    // TODO: добавьте сюда собственный код конструктора
}

CFontViewView::~CFontViewView()
{
}

BOOL CFontViewView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Измените класс или стиль окна, изменяя и добавляя
    // поля структуры cs

    return CView::PreCreateWindow(cs);
}

// Прорисовка CFontViewView

void CFontViewView::OnDraw(CDC* pDC)
{
    CFontViewDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    // TODO: добавьте сюда собственный код рисования
    RECT ClientRect;
    CFont Font;
    LOGFONT LF;
    int LineHeight;
```

```

CFont *PtrOldFont;
int X, Y;

// заполнение структуры LF свойствами
// стандартного системного шрифта
CFont TempFont;
if (pDoc->m_Pitch == PITCH_VARIABLE)
    TempFont.CreateStockObject (SYSTEM_FONT);
else
    TempFont.CreateStockObject (SYSTEM_FIXED_FONT);
TempFont.GetObject (sizeof (LOGFONT), &LF);

// инициализация полей структуры LF
if (pDoc->m_Bold) LF.lfWeight = FW_BOLD;
if (pDoc->m_Italic) LF.lfItalic = 1;
if (pDoc->m_Underlined) LF.lfUnderline = 1;

// создание и выбор шрифта
Font.CreateFontIndirect (&LF);
PtrOldFont = pDC->SelectObject (&Font);

// установка выравнивания
GetClientRect (&ClientRect);
switch (pDoc->m_Align)
{
case ALIGN_LEFT:
    pDC->SetTextAlign (TA_LEFT);
    X = ClientRect.left + 5;
    break;
case ALIGN_CENTER:
    pDC->SetTextAlign (TA_CENTER);
    X = (ClientRect.left + ClientRect.right) / 2;
    break;
case ALIGN_RIGHT:
    pDC->SetTextAlign (TA_RIGHT);
    X = ClientRect.right + 5;
    break;
}

// установка цвета текста и режима фона
pDC->SetTextColor (::GetSysColor (COLOR_WINDOWTEXT));
pDC->SetBkMode (TRANSPARENT);

// вывод строк текста
LineHeight = LF.lfHeight * pDoc->m_Spacing;
Y = 5;
pDC->TextOut (X, Y, "Example text fragment - line one.");
Y += LineHeight;
pDC->TextOut (X, Y, "Example text fragment - line two.");
Y += LineHeight;
pDC->TextOut (X, Y, "Example text fragment - line three.");

// отмена выбора шрифта
pDC->SelectObject (PtrOldFont);
}

```

```

// диагностика CFontViewView

#ifdef _DEBUG
void CFontViewView::AssertValid() const
{
    CView::AssertValid();
}

void CFontViewView::Dump(CDumpContext& dc) const
{
    CView::Dump(dc);
}

CFontViewDoc* CFontViewView::GetDocument() const
// не отладочная версия встроена
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CFontViewDoc)));
    return (CFontViewDoc*)m_pDocument;
}
#endif // _DEBUG

// Обработчики сообщений класса CFontViewView

```

---

### Листинг 15.7

```

// MainFrm.h : интерфейс класса CMainFrame
//

#pragma once
class CMainFrame : public CFrameWnd
{

protected: // используется только для сериализации
    CMainFrame();
    DECLARE_DYNCREATE(CMainFrame)

// Атрибуты
public:

// Операции
public:

// Переопределения
public:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);

// Реализация
public:
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

// Сгенерированные функции карты сообщений

```

```
protected:
    DECLARE_MESSAGE_MAP()
};
```

---

## Листинг 15.8

```
// MainFrm.cpp : реализация класса CMainFrame
//

#include "stdafx.h"
#include "FontView.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
END_MESSAGE_MAP()

// Конструктор CMainFrame

CMainFrame::CMainFrame()
{
    // TODO: добавьте сюда код инициализации элементов
}

CMainFrame::~CMainFrame()
{
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    if( !CFrameWnd::PreCreateWindow(cs) )
        return FALSE;
    // TODO: Измените класс или стили окна, изменяя и добавляя
    // поля в структуре cs

    return TRUE;
}

// Диагностика класса CMainFrame

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
    CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
```

```

{
    CFrameWnd::Dump(dc);
}

#endif // _DEBUG

// Обработчики сообщений класса CMainFrame

```

---

### Листинг 15.9

```

#pragma once
#include "afxwin.h"

enum {ALIGN_LEFT, ALIGN_CENTER, ALIGN_RIGHT};
enum {PITCH_VARIABLE, PITCH_FIXED};

// Диалог CTextprop

class CTextprop : public CDialog
{
    DECLARE_DYNAMIC(CTextprop)

protected:
    RECT m_RectSample;

public:
    CTextprop(CWnd* pParent = NULL);
    // стандартный конструктор
    virtual ~CTextprop();

    // Данные для диалога
    enum { IDD = IDD_TEXTPROP };

protected:
    virtual void DoDataExchange(CDataExchange* pDX);
    // поддержка DDX/DDV

    DECLARE_MESSAGE_MAP()
public:
    BOOL m_Bold;
    BOOL m_Italic;
    BOOL m_Align;
    BOOL m_Underlined;
    BOOL m_Pitch;
    int m_Spacing;
    CEdit m_SpacingEdit;
    afx_msg void OnBnClickedBold();
    afx_msg void OnBnClickedCenter();
    afx_msg void OnBnClickedFixed();
    afx_msg void OnBnClickedItalic();
    afx_msg void OnBnClickedLeft();
    afx_msg void OnBnClickedRight();
    afx_msg void OnBnClickedUnderlined();
    afx_msg void OnBnClickedVariable();
    afx_msg void OnEnChangeSpacing();

```

```

        virtual BOOL OnInitDialog();
        void OnPaint(void);
};

```

---

## Листинг 15.10

```

// Textprop.cpp : файл реализации
//

#include "stdafx.h"
#include "FontView.h"
#include "Textprop.h"

// Диалог CTextprop

IMPLEMENT_DYNAMIC(CTextprop, CDialog)
CTextprop::CTextprop(CWnd* pParent /*=NULL*/)
    : CDialog(CTextprop::IDD, pParent)
    , m_Bold(FALSE)
    , m_Italic(FALSE)
    , m_Align(FALSE)
    , m_Underlined(FALSE)
    , m_Pitch(FALSE)
    , m_Spacing(0)
{
}

CTextprop::~CTextprop()
{
}

void CTextprop::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    DDX_Check(pDX, IDC_BOLD, m_Bold);
    DDX_Check(pDX, IDC_ITALIC, m_Italic);
    DDX_Radio(pDX, IDC_LEFT, m_Align);
    DDX_Check(pDX, IDC_UNDERLINED, m_Underlined);
    DDX_Radio(pDX, IDC_VARIABLE, m_Pitch);
    DDX_Text(pDX, IDC_SPACING, m_Spacing);
    DDV_MinMaxInt(pDX, m_Spacing, 1, 3);
    DDX_Control(pDX, IDC_SPACING, m_SpacingEdit);
}

BEGIN_MESSAGE_MAP(CTextprop, CDialog)
    ON_BN_CLICKED(IDC_BOLD, OnBnClickedBold)
    ON_BN_CLICKED(IDC_CENTER, OnBnClickedCenter)
    ON_BN_CLICKED(IDC_FIXED, OnBnClickedFixed)
    ON_BN_CLICKED(IDC_ITALIC, OnBnClickedItalic)
    ON_BN_CLICKED(IDC_LEFT, OnBnClickedLeft)
    ON_BN_CLICKED(IDC_RIGHT, OnBnClickedRight)
    ON_BN_CLICKED(IDC_UNDERLINED, OnBnClickedUnderlined)
    ON_BN_CLICKED(IDC_VARIABLE, OnBnClickedVariable)
    ON_EN_CHANGE(IDC_SPACING, OnEnChangeSpacing)
END_MESSAGE_MAP()

```



```

// Обработчики сообщений класса CTextprop

void CTextprop::OnBnClickedBold()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_Bold = !m_Bold;
    InvalidateRect (&m_RectSample);
    UpdateWindow ();
}

void CTextprop::OnBnClickedCenter()
{
    // TODO: Добавьте сюда собственный код обработчика
    if (IsDlgButtonChecked (IDC_CENTER))
    {
        m_Align = ALIGN_CENTER;
        InvalidateRect (&m_RectSample);
        UpdateWindow ();
    }
}

void CTextprop::OnBnClickedFixed()
{
    // TODO: Добавьте сюда собственный код обработчика
    if (IsDlgButtonChecked (IDC_FIXED))
    {
        m_Pitch = PITCH_FIXED;
        InvalidateRect (&m_RectSample);
        UpdateWindow ();
    }
}

void CTextprop::OnBnClickedItalic()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_Italic = !m_Italic;
    InvalidateRect (&m_RectSample);
    UpdateWindow ();
}

void CTextprop::OnBnClickedLeft()
{
    // TODO: Добавьте сюда собственный код обработчика
    if (IsDlgButtonChecked (IDC_LEFT))
    {
        m_Align = ALIGN_LEFT;
        InvalidateRect (&m_RectSample);
        UpdateWindow ();
    }
}

void CTextprop::OnBnClickedRight()
{
    // TODO: Добавьте сюда собственный код обработчика
    if (IsDlgButtonChecked (IDC_RIGHT))

```

```

        {
            m_Align = ALIGN_RIGHT;
            InvalidateRect (&m_RectSample);
            UpdateWindow ();
        }
    }

void CTextprop::OnBnClickedUnderlined()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_Underlined=!m_Underlined;
    InvalidateRect (&m_RectSample);
    UpdateWindow ();
}

void CTextprop::OnBnClickedVariable()
{
    // TODO: Добавьте сюда собственный код обработчика
    if (IsDlgButtonChecked (IDC_VARIABLE))
    {
        m_Pitch = PITCH_VARIABLE;
        InvalidateRect (&m_RectSample);
        UpdateWindow ();
    }
}

void CTextprop::OnEnChangeSpacing()
{
    // TODO: Добавьте сюда собственный код обработчика
    int Temp;
    Temp = (int) GetDlgItemInt (IDC_SPACING);
    if (Temp > 0 && Temp < 4)
    {
        m_Spacing = Temp;
        InvalidateRect (&m_RectSample);
        UpdateWindow ();
    }
}

BOOL CTextprop::OnInitDialog()
{
    CDialog::OnInitDialog();

    // TODO: Добавьте сюда собственный код обработчика

    GetDlgItem (IDC_SAMPLE)->GetWindowRect (&m_RectSample);
    ScreenToClient (&m_RectSample);

    m_SpacingEdit.LimitText (1);

    return TRUE;
    // Возвращает TRUE, если не выбран элемент управления
    // Исключение: Страницы свойств элементов ОСХ возвращают FALSE
}

void CTextprop::OnPaint(void)

```

```

{
    CPaintDC dc(this); //контекст устройства для рисования

    // TODO: Добавьте сюда собственный код обработчика
    CFont Font;
    LOGFONT LF;
    int LineHeight;
    CFont *PtrOldFont;
    int X, Y;

    // заполнение структуры LF свойствами
    // стандартного системного шрифта
    CFont TempFont;
    if (m_Pitch == PITCH_VARIABLE)
        TempFont.CreateStockObject (SYSTEM_FONT);
    else
        TempFont.CreateStockObject (SYSTEM_FIXED_FONT);
    TempFont.GetObject (sizeof (LOGFONT), &LF);

    // инициализация полей структуры LF
    if (m_Bold) LF.lfWeight = FW_BOLD;
    if (m_Italic) LF.lfWeight = 1;
    if (m_Underlined) LF.lfUnderline = 1;

    // создание и выбор шрифта
    Font.CreateFontIndirect (&LF);
    PtrOldFont = dc.SelectObject (&Font);

    // выбор выравнивания
    switch (m_Align)
    {
    case ALIGN_LEFT:
        dc.SetTextAlign (TA_LEFT);
        X = m_RectSample.left + 5;
        break;
    case ALIGN_CENTER:
        dc.SetTextAlign (TA_CENTER);
        X = (m_RectSample.left + m_RectSample.right) / 2;
        break;
    case ALIGN_RIGHT:
        dc.SetTextAlign (TA_RIGHT);
        X = m_RectSample.right + 5;
        break;
    }

    // установка режима отображения фона
    dc.SetBkMode (TRANSPARENT);

    // вывод строк текста
    LineHeight = LF.lfHeight * m_Spacing;
    Y = m_RectSample.top + 15;
    dc.TextOut (X, Y, "AaBbCcDdEeFfGg");
    Y += LineHeight;
    dc.TextOut (X, Y, "HhIiJjKkLlMmNn");
    Y += LineHeight;
    dc.TextOut (X, Y, "OoPpQqRrSsTtUu");
}

```

```
dc.SelectObject (PtrOldFont);  
}
```

## Немодальное диалоговое окно

Рассмотрим вкратце методы отображения *немодальных* диалоговых окон. Окна этой разновидности распространены менее, чем модальные, однако в определенных ситуациях бывают очень полезны. При отображении модального диалогового окна главное окно программы блокируется, поэтому модальное окно необходимо закрыть для продолжения работы в главном окне. А при отображении немодального диалогового окна доступ к главному окну программы не блокируется. В результате можно продолжать работу внутри главного окна с одновременным отображением немодального. Фокус ввода переключается между немодальным диалоговым окном и главным. Таким образом, немодальное диалоговое окно служит вспомогательным окном и используется вместе с главным. Например, команда проверки орфографии для текстового процессора обычно отображает немодальное диалоговое окно (можно внести исправления в документ, а затем продолжить проверку орфографии). В гл. 14 кратко описана разновидность панелей элементов MFC, называемая *диалоговой панелью*, – фактически немодальное диалоговое окно, которым управляет класс `CDialogBar`, а не `CDialog`. Как и модальное диалоговое окно, немодальное создают, используя редактор диалоговых окон Visual Studio. Чтобы породить класс для управления немодальным диалоговым окном от класса `CDialog` и определить переменные-члены и обработчики сообщений, используют те же методы, что и для модального диалогового окна. Существует ряд различий в способах отображения диалоговых окон немодальных и модальных.

Для начала экземпляра класса немодального диалогового окна необходимо объявить как глобальный объект или создать с помощью оператора `new` вместо объявления его локальным объектом. Это необходимо потому, что немодальное диалоговое окно остается открытым после завершения функции, которая его отображает, а объект, управляющий диалоговым окном, сохраняется. При создании объекта с использованием оператора `new` убедитесь, что оператор `delete` используется для уничтожения этого объекта.

Отображается немодальное диалоговое окно путем вызова функции `CDialog::Create` вместо `CDialog::DoModal`. В отличие от последней функция `Create` возвращает управление, оставляя диалоговое окно на экране. При отображении такого диалогового окна главное окно, как и любые окна представлений, остается доступным. Пользователь может работать с главным окном при отображенном диалоговом окне, а программа может продолжать обработку информации, вводимой из главного окна.

Для закрытия немодального диалогового окна следует вызвать функцию `CWnd::DestroyWindow` вместо функции `EndDialog`. Функцию `DestroyWindow` можно вызвать из функции-члена класса диалогового окна или из любой другой функции.

Для класса немодального диалогового окна необходимо определить обработчик сообщения `OnCancel`. Если диалоговое окно содержит кнопку ОК (т. е. кнопку с идентификатором `IDOK`), то необходимо определить обработчик сообщения `OnOK`. (Эти функции описаны в параграфе “Управление диалоговым окном Text Properties” выше в этой главе.) Однако для немодального диалогового окна в созданных обработчиках функций `OnOK` и `OnCancel` необходимо вызывать функцию `DestroyWindow`, закрывающую диалоговое окно, и *не* вызывать обработчик сообщения базового класса, поскольку версии функций базового класса `OnCancel` и `OnOK` вызывают функцию `EndDialog`, скрывающую диалоговое окно, но не уничтожающую его. Заметьте: стандартная функция `OnOK` должна была бы вызвать функцию `CWnd::UpdateData`, передавая ей значение `TRUE` или вызывая ее без параметров, чтобы сохранить и проверить содержимое элементов управления. В модальном диалоговом окне эта задача выполняется версией функции `OnOK` базового класса.

## Диалоговое окно с вкладками

---

Диалоговые окна с вкладками широко применяются в современных приложениях Windows. Библиотека MFC поддерживает эту разновидность окон. Такое окно позволяет отображать несколько страниц связанных элементов управления внутри одного окна. Способ обращения к каждой странице подобен выбору закладки в книге: щелчок на ярлычке в строке ярлычков вкладок, отображаемой в верхней части окна. Диалоговое окно с вкладками поддерживается объектом класса `CPropertySheet`, а каждая страница – класса `CPropertyPage`. Для разработки страницы используется редактор диалоговых окон, как будто это отдельное окно. Однако при этом для диалогового окна выбирается определенный набор стилей.

Для демонстрации процедуры проектирования окон со вкладками, в этом параграфе будет создана программа `TabView`. Она похожа на предыдущий пример `FontView`. Отличие в том, что диалоговое окно `Text Properties` – это окно с вкладками (для простоты примера оно не отображает образец текста):

- первая страница (вкладка) диалогового окна `Text Properties` – страница `Style` – содержит флажки для выбора начертания шрифта;
- вторая страница – `Alignment` – переключатели задания параметров выравнивания текста;
- третья страница – `Pitch and Spacing` – переключатели для выбора ширины символов и поле для установки интервала между строками.

Заметим: класс `CPropertySheet` создает страницы, по крайней мере, такой же ширины, как ширина трех кнопок внизу диалогового окна с учетом пустого пространства для дополнительной кнопки `Help`. Следовательно, в программе `TabView` страницы имеют больший размер, чем необходимо для размещения отображаемых элементов управления. Однако, если в реальных приложениях не требуется отображать большое количество элементов управления, то диалоговые окна с вкладками обычно не используются.

Программу `TabView` будем проектировать, используя мастер `Application Wizard`. В диалоговом окне `New Project Workspace` введите в поле `Name` имя программы `TabView`, а в поле `Location` – соответствующую папку проекта. В диалоговых окнах мастера `Application Wizard` задайте те же параметры, что и для программы `FontView`. После того как исходные файлы будут сгенерированы создайте шаблон диалогового окна для страницы `Style`. В редакторе диалоговых окон щелкните правой кнопкой мыши внутри создаваемого диалогового окна, выберите из появившегося контекстного меню команду `Properties` и откройте диалоговое окно `Properties`. В поле `Caption` введите `Style`. Эта надпись отобразится на ярлычке создаваемой страницы. В пункте `Style` в списке выберите пункт `Child`, а в пункте `Border` в списке – пункт `Thin`. Установите в пункте `Disabled` значение `True`. Все заданные установки необходимы для определения диалогового окна, используемого как страница окна с вкладками (а не как отдельное диалоговое окно). Удалите ненужные кнопки и добавьте необходимые элементы управления (в табл. 15.8 приведены эти элементы управления вместе с элементами, добавляемыми к двум оставшимся диалоговым окнам).

Для управления окном `Style` создайте класс (как это было описано ранее в этой главе). В поле `Name` введите `CStyle`, а в списке `Base class` выберите `CPropertyPage` (класс, управляющий страницами диалоговых окон с вкладками, должен порождаться от класса `CPropertyPage`, а не `CDialog`). Оставьте без изменения другие стандартные установки и щелкните на кнопке `OK`. Следуя инструкциям, приведенным при создании программы `FontView`, определите переменные для флажков `Bold`, `Italic` и `Underlined`. Для каждой переменной укажите имя, категорию и тип, как в программе `FontView`.

Аналогичным образом создайте шаблоны второго и третьего диалоговых окон для двух оставшихся страниц, повторив для них описанные выше действия. Свойства элементов управления приведены в табл. 15.8. Класс диалогового окна `Alignment` должен называться `CAlignment`, а класс окна `Pitch and Spacing` – `CPitch`. Для каждой группы переключателей и для поля `Line Spacing` добавьте такие же переменные-члены, как были определены для программы `FontView`.

Табл. 15.8. Свойства элементов управления, добавляемые в шаблоны диалоговых окон Style, Alignment и Pitch and Spacing

Идентификатор (ID)	Тип элемента управления (в разделе Dialog Editor панели инструментов)	Устанавливаемые свойства
IDC_STATIC	Рамка	Надпись (Caption): Font Styles
IDC_BOLD	Флажок	Надпись (Caption): & Bold Group
IDC_ITALIC	Флажок	Надпись (Caption): &Italic
IDC_UNDERLINED	Флажок	Надпись (Caption): &Underlined
IDC_STATIC	Рамка	Надпись (Caption): Text Alignment
IDC_LEFT	Переключатель	Надпись (Caption): &Left Group
IDC_CENTER	Переключатель	Надпись (Caption): &Center
IDC_RIGHT	Переключатель	Надпись (Caption): &Right
IDC_STATIC	Рамка	Надпись (Caption): Font Pitch
IDC_VARIABLE	Переключатель	Надпись (Caption): &Variable Group
IDC_FIXED	Переключатель	Надпись (Caption): &Fixed
IDC_STATIC	Надпись	Надпись (Caption): Line &Spacing:
IDC_SPACING	Поле	

Если для разных страниц диалоговые окна имеют разные размеры, то *первую* страницу делают самой большой, потому что класс CPropertySheet вычисляет размер страницы, основываясь на размере диалогового окна первой страницы. Однако к программе TabView это не относится, потому что в ней размеры страниц меньше минимальных. При компоновке набора страниц в диалоговом окне с вкладками старайтесь размещать схожие группы элементов управления на разных страницах в одних и тех же местах. Иначе будет казаться, что при переключении с одной страницы на другую элементы управления прыгают.

Создайте обработчик сообщения класса CPitch с именем OnInitDialog. Руководствуйтесь методикой, описанной в параграфе “Определение обработчиков сообщений” при создании функции OnInitDialog класса CTextprop программы FontView. Функция OnInitDialog вызывается только при *первом* отображении связанной с ней страницы в диалоговом окне с вкладками, и *не* вызывается при повторном отображении страницы, вызванном щелчком мышью на ярлычке. Добавьте к этой функции следующий фрагмент в файле Pitch.cpp.

```
// Обработчики сообщений класса CPitch

BOOL CPitch::OnInitDialog()
{
    CPropertyPage::OnInitDialog();

    // TODO: Добавьте сюда команды дополнительной инициализации
    m_SpacingEdit.LimitText (1);
}
```

```

return TRUE; // Вернет TRUE, если фокус не устанавливался
              // на элементы управления. Исключение: Страницы
              // свойств OCX-элементов возвращают FALSE
}

```

Внесите изменения в меню программы, удалив меню File и Edit и добавив меню Text с командами Text Properties и Exit, а также сочетание клавиш Ctrl+T, как было сделано в программе FontView. Сгенерируйте обработчик командного сообщения для команды Text Properties, как в программе FontView (см. параграф “Обработчики сообщений”). Добавьте в функцию OnTextFormat в файле TabViewDoc.cpp строки, выделенные полужирным в приведенном ниже листинге. Добавленный код напоминает код отображения стандартного диалогового окна в программе FontView. Однако вместо создания экземпляра класса диалогового окна функция OnTextFormat создает экземпляр класса CPropertySheet. Этот объект управляет отображением диалогового окна с вкладками. Затем функция OnTextFormat создает экземпляры всех классов, порождаемых от класса CPropertyPage. Каждый из этих объектов связан с одним из разработанных шаблонов диалогового окна и управляет определенной страницей. Функция OnTextFormat вызывает функцию AddPage для объектов класса CPropertySheet, чтобы добавить в окно новую страницу, и функцию DoModal класса CPropertySheet для создания и отображения диалогового окна со вкладками, содержащего для каждого добавленного объекта отдельную страницу.

```

// Команды класса CTabViewDoc

void CTabViewDoc::OnTextFormat()
{
    // TODO: Добавьте сюда собственный код обработчика

    // Создание объекта окна со вкладками
    CPropertySheet PropertySheet ("Text Properties");

    // Создание объектов страниц
    CStyle StylePage;
    CAlignment AlignmentPage;
    CPitch PitchPage;

    // Добавление страниц к объекту окна
    PropertySheet.AddPage (&StylePage);
    PropertySheet.AddPage (&AlignmentPage);
    PropertySheet.AddPage (&PitchPage);

    // Инициализация объектов страниц
    StylePage.m_Bold = m_Bold;
    StylePage.m_Italic = m_Italic;
    StylePage.m_Underlined = m_Underlined;
    AlignmentPage.m_Align = m_Align;
    PitchPage.m_Pitch = m_Pitch;
    PitchPage.m_Spacing = m_Spacing;

    // Отображение окна со вкладками
    if (PropertySheet.DoModal () == IDOK)
    {
        // сохранение значений из окна
        m_Bold = StylePage.m_Bold;
        m_Italic = StylePage.m_Italic;
    }
}

```

```

        m_Underlined = StylePage.m_Underlined;
        m_Align = AlignmentPage.m_Align;
        m_Pitch = PitchPage.m_Pitch;
        m_Spacing = PitchPage.m_Spacing;
    }

    // перерисовка текста
    UpdateAllViews (NULL);
}

```

Класс CPropertyPage (как и класс CDialog) создает обработчики для кнопок OK и Cancel, а также проверяет и передает данные между элементами управления и функциями объектов-страниц. Чтобы сделать доступной кнопку Apply или добавить кнопку Help и обработчик для одной из этих кнопок, необходимо породить собственный класс от класса CPropertyPage. (В программе TabView кнопка Apply заблокирована. В документации фирмы Microsoft утверждается, что лучше оставить эту кнопку заблокированной, а не пытаться ее удалить, так как это – стандартное средство интерфейса диалоговых окон с вкладками.) Аналогичное действие необходимо выполнить, если вы хотите увеличить размер диалогового окна с вкладками и добавить дополнительные элементы управления в само диалоговое окно (а не в одну из вкладок). Например, можно включить рамку для отображения образца, показывающего результаты применения параметров диалогового окна (подобно рамке Sample диалогового окна Text Properties программы FontView). Дополнительная информация содержится, в частности, в разделах справочной системы по классам CPropertySheet и CPropertyPage.

Для каждого из классов страниц включите в файл TabViewDoc.cpp файлы заголовков.

```

#include "stdafx.h"
#include "TabView.h"

#include "TabViewDoc.h"

#include "Style.h"
#include "Pitch.h"
#include "Alignment.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

```

Для переменных, хранящих параметры форматирования, добавьте код инициализации.

```

// Конструктор класса CTabViewDoc

CTabViewDoc::CTabViewDoc()
{
    // TODO: добавьте сюда собственный код конструктора
    m_Bold = FALSE;
    m_Italic = FALSE;
    m_Underlined = FALSE;
    m_Align = ALIGN_LEFT;
    m_Pitch = PITCH_VARIABLE;
    m_Spacing = 1;
}

```

Определите следующие перечисления и переменные в файле TabViewDoc.h.

```

enum {ALIGN_LEFT, ALIGN_CENTER, ALIGN_RIGHT};
enum {PITCH_VARIABLE, PITCH_FIXED};

```



```

class CTabViewDoc : public CDocument
{
public:
    BOOL m_Bold;
    BOOL m_Italic;
    BOOL m_Underlined;
    int m_Align;
    int m_Pitch;
    int m_Spacing;

```

В функцию OnDraw в файле TabViewView.cpp добавьте для отображения текста в окне *такой же* код, как был добавлен в функцию OnDraw класса представления программы FontView. В файле TabView.cpp добавьте в функцию InitInstance вызов функции SetWindowText.

```

m_pMainWnd->ShowWindow(SW_SHOW);
m_pMainWnd->UpdateWindow();

m_pMainWnd->SetWindowText ("Dialog Box Tabs Demo");

```

Мы изучили процедуру создания модального диалогового окна с вкладками. Можно также отобразить немодальное диалоговое окно с вкладками, используя методы, описанные в этом параграфе в сочетании с методами, рассмотренными в предыдущем параграфе “Немодальное диалоговое окно”. Дополнительная информация содержится в справочной системе.

## Текст программы TabView

Теперь можете построить и запустить программу TabView. Текст программы приведен в листингах 15.11—15.24.

---

### Листинг 15.11

```

// TabView.h : главный заголовочный файл приложения TabView
//

#pragma once

#ifdef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"          // основные символы

// CTabViewApp:
// Смотрите реализацию данного класса в файле TabView.cpp

class CTabViewApp : public CWinApp
{
public:
    CTabViewApp();

    // Переопределения
public:
    virtual BOOL InitInstance();

```

```

// Реализация
afx_msg void OnAppAbout();
DECLARE_MESSAGE_MAP()
};

extern CTabViewApp theApp;

```

---

### Листинг 15.12

```

// TabView.cpp : Определяет поведение классов приложения.
//

#include "stdafx.h"
#include "TabView.h"
#include "MainFrm.h"

#include "TabViewDoc.h"
#include "TabViewView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CTabViewApp

BEGIN_MESSAGE_MAP(CTabViewApp, CWinApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
    // Standard file based document commands
    ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
END_MESSAGE_MAP()

// Конструктор класса CTabViewApp

CTabViewApp::CTabViewApp()
{
    // TODO: добавьте сюда собственный код конструктора
    // Поместите все существенные команды инициализации в
    // функцию InitInstance
}

// Единственный объект класса CFontViewApp

CTabViewApp theApp;

// Инициализация CFontViewApp

BOOL CTabViewApp::InitInstance()
{
    CWinApp::InitInstance();

    // Инициализация библиотек OLE
    if (!AfxOleInit())
    {
        AfxMessageBox(IDP_OLE_INIT_FAILED);
    }
}

```

```

        return FALSE;
    }
    AfxEnableControlContainer();
    // Стандартная инициализация
    // Если вы не используете какие-то из функций, указанных ниже,
    // и хотите уменьшить размер конечного исполняемого модуля,
    // удалите соответствующие процедуры инициализации.
    // Измените строку-аргумент функции (реестровый ключ,
    // под которым хранятся ваши установки) на что-нибудь
    // запоминающееся, например, название вашей компании
    // или организации
    SetRegistryKey(_T("Local AppWizard-Generated Applications"));
    LoadStdProfileSettings(4); // Загрузка установок из INI-файла
                                // (включая MRU)

    // Регистрация шаблонов документов приложения. Они служат связью
    // между документами, окнами документов и окнами представлений.
    CSingleDocTemplate* pDocTemplate;
    pDocTemplate = new CSingleDocTemplate(
        IDR_MAINFRAME,
        RUNTIME_CLASS(CTabViewDoc),
        RUNTIME_CLASS(CMainFrame), // Главное окно
                                // SDI-приложения
        RUNTIME_CLASS(CTabViewView));
    AddDocTemplate(pDocTemplate);
    // Поиск в командной строке команд оболочки,
    // DDE, открытия файлов

    CCommandLineInfo cmdInfo;
    ParseCommandLine(cmdInfo);
    // Выполнение команд, обнаруженных в командной строке.
    // Вернет False, если приложение было запущено с /RegServer,
    // /Register, /Unregserver or /Unregister.
    if (!ProcessShellCommand(cmdInfo))
        return FALSE;
    // Обновление и отображение единственного окна приложения
    m_pMainWnd->ShowWindow(SW_SHOW);
    m_pMainWnd->UpdateWindow();

    m_pMainWnd->SetWindowText ("Dialog Box Tabs Demo");

    return TRUE;
}

// CAboutDlg диалог, используемый в App About

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

    // Dialog Data
    enum { IDD = IDD_ABOUTBOX };

protected:
    virtual void DoDataExchange(CDataExchange* pDX);
    // Поддержка DDX/DDV

```

```

// Реализация
protected:
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
END_MESSAGE_MAP()

// Команда приложения на запуск диалога
void CTabViewApp::OnAppAbout()
{
    CAboutDlg aboutDlg;
    aboutDlg.DoModal();
}

// Обработчики сообщений класса CTabViewApp

```

---

### Листинг 15.13

```

// TabViewDoc.h : интерфейс класса CTabViewDoc
//

#pragma once

enum {ALIGN_LEFT, ALIGN_CENTER, ALIGN_RIGHT};
enum {PITCH_VARIABLE, PITCH_FIXED};

class CTabViewDoc : public CDocument
{
public:
    BOOL m_Bold;
    BOOL m_Italic;
    BOOL m_Underlined;
    int m_Align;
    int m_Pitch;
    int m_Spacing;

protected: // используется только для сериализации
    CTabViewDoc();
    DECLARE_DYNCREATE(CTabViewDoc)

// Attributes
public:

// Операции
public:

```

```

// Переопределения
public:
    virtual BOOL OnNewDocument();
    virtual void Serialize(CArchive& ar);

// Реализация
public:
    virtual ~CTabViewDoc();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Сгенерированные функции карты сообщений
protected:
    DECLARE_MESSAGE_MAP()
public:
    afx_msg void OnTextFormat();
};

```

---

### Листинг 15.14

```

// TabViewDoc.cpp : реализация класса CTabViewDoc
//

#include "stdafx.h"
#include "TabView.h"

#include "TabViewDoc.h"

#include "Style.h"
#include "Pitch.h"
#include "Alignment.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CTabViewDoc

IMPLEMENT_DYNCREATE(CTabViewDoc, CDocument)

BEGIN_MESSAGE_MAP(CTabViewDoc, CDocument)
    ON_COMMAND(ID_TEXT_FORMAT, OnTextFormat)
END_MESSAGE_MAP()

// Конструктор класса CTabViewDoc

CTabViewDoc::CTabViewDoc()
{
    // TODO: добавьте сюда собственный код конструктора
    m_Bold = FALSE;
}

```

```

        m_Italic = FALSE;
        m_Underlined = FALSE;
        m_Align = ALIGN_LEFT;
        m_Pitch = PITCH_VARIABLE;
        m_Spacing = 1;
    }

CTabViewDoc::~CTabViewDoc()
{
}

BOOL CTabViewDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;

    // TODO: добавьте сюда код реинициализации
    // (SDI-документы будут повторно использовать этот документ)

    return TRUE;
}

// Сериализация CTabViewDoc

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

// Диагностика класса CTabViewDoc

#ifdef _DEBUG
void CTabViewDoc::AssertValid() const
{
    CDocument::AssertValid();
}

void CTabViewDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}
#endif // _DEBUG

// Команды класса CTabViewDoc

void CTabViewDoc::OnTextFormat()
{
    // TODO: Добавьте сюда собственный код обработчика

```

```

// Создание объекта окна со вкладками
CPropertySheet PropertySheet ("Text Properties");

// Создание объектов страниц
CStyle StylePage;
CAlignment AlignmentPage;
CPitch PitchPage;

// Добавление страниц к объекту окна
PropertySheet.AddPage (&StylePage);
PropertySheet.AddPage (&AlignmentPage);
PropertySheet.AddPage (&PitchPage);

// Инициализация объектов страниц
StylePage.m_Bold = m_Bold;
StylePage.m_Italic = m_Italic;
StylePage.m_Underlined = m_Underlined;
AlignmentPage.m_Align = m_Align;
PitchPage.m_Pitch = m_Pitch;
PitchPage.m_Spacing = m_Spacing;

// Отображение окна со вкладками
if (PropertySheet.DoModal () == IDOK)
{
    // сохранение значений из окна
    m_Bold = StylePage.m_Bold;
    m_Italic = StylePage.m_Italic;
    m_Underlined = StylePage.m_Underlined;
    m_Align = AlignmentPage.m_Align;
    m_Pitch = PitchPage.m_Pitch;
    m_Spacing = PitchPage.m_Spacing;
}

// перерисовка текста
UpdateAllViews (NULL);
}

```

---

### Листинг 15.15

```

// TabViewView.h : интерфейс класса CTabViewView
//

#pragma once

class CTabViewView : public CView
{
protected: // используется только для сериализации
    CTabViewView();
    DECLARE_DYNCREATE(CTabViewView)

// Атрибуты
public:
    CTabViewDoc* GetDocument() const;

```

```

// Операции
public:

// Переопределения
public:
    virtual void OnDraw(CDC* pDC);
// переопределена для прорисовки этого вида
virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
protected:

// Реализация
public:
    virtual ~CTabViewView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Сгенерированные функции карты сообщений
protected:
    DECLARE_MESSAGE_MAP()
};

#ifdef _DEBUG // отладочная версия в файле TabViewView.cpp
inline CTabViewDoc* CTabViewView::GetDocument() const
    { return reinterpret_cast<CTabViewDoc*>(m_pDocument); }
#endif

```

---

### Листинг 15.16

```

// TabViewView.cpp : реализация класса CTabViewView
//

#include "stdafx.h"
#include "TabView.h"

#include "TabViewDoc.h"
#include "TabViewView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CTabViewView

IMPLEMENT_DYNCREATE(CTabViewView, CView)

BEGIN_MESSAGE_MAP(CTabViewView, CView)
END_MESSAGE_MAP()

// Конструктор класса CTabViewView

CTabViewView::CTabViewView()

```



```

{
    // TODO: добавьте сюда собственный код конструктора
}

CTabViewView::~CTabViewView()
{
}

BOOL CTabViewView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Измените класс или стили окна, изменяя и
    // добавляя поля в структуру cs

    return CView::PreCreateWindow(cs);
}

// Прорисовка CTabViewView

void CTabViewView::OnDraw(CDC* pDC)
{
    CTabViewDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    // TODO: добавьте код для прорисовки собственных данных
    RECT ClientRect;
    CFont Font;
    LOGFONT LF;
    int LineHeight;
    CFont *PtrOldFont;
    int X, Y;

    // Заполнение структуры LF свойствами
    // стандартного системного шрифта
    CFont TempFont;
    if (pDoc->m_Pitch == PITCH_VARIABLE)
        TempFont.CreateStockObject (SYSTEM_FONT);
    else
        TempFont.CreateStockObject (SYSTEM_FIXED_FONT);
    TempFont.GetObject (sizeof (LOGFONT), &LF);

    // Инициализация полей структуры LF
    if (pDoc->m_Bold) LF.lfWeight = FW_BOLD;
    if (pDoc->m_Italic) LF.lfItalic = 1;
    if (pDoc->m_Underlined) LF.lfUnderline = 1;

    // Создание и выбор шрифта
    Font.CreateFontIndirect (&LF);
    PtrOldFont = pDC->SelectObject (&Font);

    // установка выравнивания
    GetClientRect (&ClientRect);
    switch (pDoc->m_Align)
    {
    case ALIGN_LEFT:
        pDC->SetTextAlign (TA_LEFT);
    }
}

```

```

        X = ClientRect.left + 5;
        break;
    case ALIGN_CENTER:
        pDC->SetTextAlign (TA_CENTER);
        X = (ClientRect.left + ClientRect.right) / 2;
        break;
    case ALIGN_RIGHT:
        pDC->SetTextAlign (TA_RIGHT);
        X = ClientRect.right + 5;
        break;
}

// установка цвета текста и режима фона
pDC->SetTextColor (::GetSysColor (COLOR_WINDOWTEXT));
pDC->SetBkMode (TRANSPARENT);

// вывод строк текста
LineHeight = LF.lfHeight * pDoc->m_Spacing;
Y = 5;
pDC->TextOut (X, Y, "Example text fragment - line one.");
Y += LineHeight;
pDC->TextOut (X, Y, "Example text fragment - line two.");
Y += LineHeight;
pDC->TextOut (X, Y, "Example text fragment - line three.");

// отмена выбора шрифта
pDC->SelectObject (PtrOldFont);
}

// Диагностика CTabViewView

#ifdef _DEBUG
void CTabViewView::AssertValid() const
{
    CView::AssertValid();
}

void CTabViewView::Dump(CDumpContext& dc) const
{
    CView::Dump(dc);
}

CTabViewDoc* CTabViewView::GetDocument() const
// не отладочная версия встроена
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CTabViewDoc)));
    return (CTabViewDoc*)m_pDocument;
}
#endif // _DEBUG

// Обработчики сообщений класса CTabViewView

```

### Листинг 15.17

```
// MainFrm.h : интерфейс класса CMainFrame
//

#pragma once
class CMainFrame : public CFrameWnd
{

protected: // используется только для сериализации
    CMainFrame();
    DECLARE_DYNCREATE(CMainFrame)

// Атрибуты
public:

// Операции
public:

// Переопределения
public:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);

// Реализация
public:
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

// Сгенерированные функции карты сообщений
protected:
    DECLARE_MESSAGE_MAP()
};
```

---

### Листинг 15.18

```
// MainFrm.cpp : реализация класса CMainFrame
//

#include "stdafx.h"
#include "TabView.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
END_MESSAGE_MAP()
```

```

// Конструктор класса CMainFrame

CMainFrame::CMainFrame()
{
    // TODO: добавьте сюда код инициализации элементов
}

CMainFrame::~CMainFrame()
{
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    if( !CFrameWnd::PreCreateWindow(cs) )
        return FALSE;
    // TODO: Измените класс или стили окна, изменяя или добавляя
    // поля структуры cs

    return TRUE;
}

// Диагностика CMainFrame

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
    CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
    CFrameWnd::Dump(dc);
}

#endif //_DEBUG

// Обработчики сообщений класса CMainFrame

```

---

## Листинг 15.19

```

#pragma once

// Диалог CAlignment

class CAlignment : public CPropertyPage
{
    DECLARE_DYNAMIC(CAlignment)

public:
    CAlignment();
    virtual ~CAlignment();

    // Данные для диалога
    enum { IDD = IDD_DIALOG2 };

```

```
protected:
    virtual void DoDataExchange(CDataExchange* pDX);
// Поддержка DDX/DDV

    DECLARE_MESSAGE_MAP()
public:
    BOOL m_Align;
};
```

---

### Листинг 15.20

```
// Alignment.cpp : файл реализации
//

#include "stdafx.h"
#include "TabView.h"
#include "Alignment.h"

// Диалог CAlignment

IMPLEMENT_DYNAMIC(CAlignment, CPropertyPage)
CAlignment::CAlignment()
    : CPropertyPage(CAlignment::IDD)
    , m_Align(FALSE)
{
}

CAlignment::~CAlignment()
{
}

void CAlignment::DoDataExchange(CDataExchange* pDX)
{
    CPropertyPage::DoDataExchange(pDX);
    DDX_Radio(pDX, IDC_LEFT, m_Align);
}

BEGIN_MESSAGE_MAP(CAlignment, CPropertyPage)
END_MESSAGE_MAP()

// Обработчики сообщений класса CAlignment
```

---

### Листинг 15.21

```
#pragma once
#include "afxwin.h"

// Диалог CPitch

class CPitch : public CPropertyPage
{
    DECLARE_DYNAMIC(CPitch)

public:
    CPitch();
```

```

        virtual ~CPitch();

// Данные для диалога
        enum { IDD = IDD_DIALOG3 };

protected:
        virtual void DoDataExchange(CDataExchange* pDX);
// Поддержка DDX/DDV

        DECLARE_MESSAGE_MAP()
public:
        int m_Pitch;
        CEdit m_SpacingEdit;
        int m_Spacing;
        virtual BOOL OnInitDialog();
};

```

---

## Листинг 15.22

```

// Pitch.cpp : файл реализации
//

#include "stdafx.h"
#include "TabView.h"
#include "Pitch.h"

// Диалог CPitch

IMPLEMENT_DYNAMIC(CPitch, CPropertyPage)
CPitch::CPitch()
    : CPropertyPage(CPitch::IDD)
    , m_Pitch (1)
    , m_Spacing(0)
{
}

CPitch::~CPitch()
{
}

void CPitch::DoDataExchange(CDataExchange* pDX)
{
    CPropertyPage::DoDataExchange(pDX);
    DDX_Radio(pDX, IDC_VARIABLE, m_Pitch);
    DDX_Control(pDX, IDC_SPACING, m_SpacingEdit);
    DDX_Text(pDX, IDC_SPACING, m_Spacing);
    DDV_MinMaxInt(pDX, m_Spacing, 1, 3);
}

BEGIN_MESSAGE_MAP(CPitch, CPropertyPage)
END_MESSAGE_MAP()

// Обработчики сообщений класса CPitch

BOOL CPitch::OnInitDialog()
{

```

```

        CPropertyPage::OnInitDialog();

        // TODO: Добавьте сюда собственный код обработчика
        m_SpacingEdit.LimitText (1);

        return TRUE; // Вернет TRUE, если фокус не был установлен на
                    // элементы управления окна
// Исключение: страницы свойств элементов ОСХ возвращают FALSE
}

```

---

### Листинг 15.23

```

#pragma once

// Диалог CStyle
class CStyle : public CPropertyPage
{
    DECLARE_DYNAMIC(CStyle)

public:
    CStyle();
    virtual ~CStyle();

    // Данные для диалога
    enum { IDD = IDD_DIALOG1 };

protected:
    virtual void DoDataExchange(CDataExchange* pDX);
    // Поддержка DDX/DDV

    DECLARE_MESSAGE_MAP()
public:
    BOOL m_Bold;
    BOOL m_Italic;
    BOOL m_Underlined;
};

```

---

### Листинг 15.24

```

// Style.cpp : файл реализации
//

#include "stdafx.h"
#include "TabView.h"
#include "Style.h"

// Диалог CStyle

IMPLEMENT_DYNAMIC(CStyle, CPropertyPage)
CStyle::CStyle()
    : CPropertyPage(CStyle::IDD)
    , m_Bold(FALSE)
    , m_Italic(FALSE)
    , m_Underlined(FALSE)
{
}

```

```

CStyle::~CStyle()
{
}

void CStyle::DoDataExchange(CDataExchange* pDX)
{
    CPropertyPage::DoDataExchange(pDX);
    DDX_Check(pDX, IDC_BOLD, m_Bold);
    DDX_Check(pDX, IDC_ITALIC, m_Italic);
    DDX_Check(pDX, IDC_UNDERLINED, m_Underlined);
}

BEGIN_MESSAGE_MAP(CStyle, CPropertyPage)
END_MESSAGE_MAP()

// Обработчики сообщений класса CStyle

```

## Диалоговое окно общего назначения

В системе Windows предусмотрен набор *диалоговых окон общего назначения*, предназначенных для выполнения специфических задач, например, открытия файла или выбора цвета. Библиотека MFC предоставляет классы для управления диалоговыми окнами общего назначения всех типов (табл. 15.9).

Табл. 15.9. Диалоговые окна общего назначения

MFC-класс	Управляемое диалоговое окно общего назначения
CColorDialog	Color – выбор цвета
CFileDialog	Open – открытие файла. Save As – сохранение файла под указанным именем
CFindReplaceDialog	Find – поиск текста. Replace – замена текста
CFontDialog	Font – выбор шрифта текста
ColeDialog	Данный класс (как и порожденные от него) предназначен для создания диалоговых окон приложений OLE
CPageSetupDialog	OLE Page Setup – определение установок страниц и полей печати для приложений OLE
CPrintDialog	Print – печать файла. Print Setup – задание установок принтера

Некоторые из этих диалоговых окон используются библиотекой MFC. Для выполнения команд Open... и Save As... меню File используются диалоговые окна общего назначения, управляемые классом CFileDialog, а для команд Print... и Print Setup... – диалоговые окна общего назначения, управляемые классом CPrintDialog. Класс CEditView использует окно класса CFindReplaceDialog для реализации команд Find... и Replace... меню Edit (см. гл. 10). Эти диалоговые окна можно вызывать из собственной программы. Кроме того, можно настроить вид и работу такого окна при его отображении. Использование класса CFontDialog рассматривается в гл. 18, а класса CColorDialog – в гл. 19. Дополнительная информация об отображении диалоговых окон общего назначения содержится в справочной системе.



## Резюме

---

Мы рассмотрели способы разработки, отображения и управления модальными диалоговыми окнами – как стандартными одностраничными, так и диалоговыми окнами со вкладками. Рассмотрены немодальные и системные диалоговые окна общего назначения. Разработаны демонстрационные программы.

- *Редактор диалоговых окон.* После создания программы с помощью мастера Application Wizard можно сконструировать диалоговое окно, используя редактор диалоговых окон Visual Studio, который отображает точную полноразмерную копию создаваемого диалогового окна и содержит панель инструментов для добавления элементов управления в окно. В процессе редактирования можно использовать мышь для изменения размеров или положения диалогового окна и любого элемента управления, находящегося внутри него. Свойства диалогового окна или любого элемента управления можно устанавливать, вводя требуемые значения в диалоговое окно Properties. Различные типы элементов управления рассмотрены в начале главы.
- *Модальное диалоговое окно.* Это временное окно, открывающееся поверх главного для отображения информации и получения данных. Перед продолжением работы в главном окне модальное необходимо закрыть. Для отображения модального диалогового окна следует создать объект класса диалогового окна и вызвать функцию `CDialog::DoModal` для этого объекта. Эта функция не завершается до тех пор, пока пользователь не закроет диалоговое окно.
- *Немодальное диалоговое окно.* Такое окно можно оставить открытым при работе с главным окном программы. Немодальное диалоговое окно, как и модальное, создается в редакторе диалоговых окон.
- *Диалоговое окно со вкладками.* Это окно позволяет отображать несколько страниц взаимосвязанных элементов управления в одном диалоговом окне. Для управления диалоговым окном со вкладками необходимо создать экземпляр класса `CPropertySheet` (или класса, порожденного от него). Кроме того, чтобы управлять каждой отображаемой страницей, необходимо создать объект класса, порожденного от класса `CPropertyPage`. Каждый объект страницы связан с созданным в редакторе диалоговых окон шаблоном диалогового окна (определяющим размещение элементов управления на странице) и добавляется к объекту `CPropertySheet` при вызове функции `CPropertySheet::AddPage`. Функция `CPropertySheet::DoModal` позволяет отобразить диалоговое окно с вкладками.
- *Диалоговое окно общего назначения.* Windows предоставляет набор диалоговых окон общего назначения для выполнения стандартных задач (например, выбора шрифта или открытия файлов).

## Глава 16

### Диалоговые программы

---

- Простая программа с окном диалога **DialogView**
- Программа просмотра формы **FormView**

У разработанных в этой книге программ окно программы содержит рабочую область, в которой отображаются текст и графика. Подобные прикладные программы предназначены для создания, просмотра и редактирования различных документов. К таким приложениям относятся текстовые процессоры, средства обработки электронных таблиц и графические редакторы. В данной главе разрабатываются *диалоговые приложения*, т.е. прикладные программы, у которых в главном окне отображается совокупность элементов управления (поля ввода, переключатели, списки), а само окно основано на шаблоне, разработанном с помощью редактора Visual Studio. Это – программы ввода данных, поиска файлов, набора телефонных номеров, калькуляторы, программы работы с дисками и т.д. Они удобны для сбора и отображения структурированной информации (например, записей баз данных).

В этой главе рассматриваются диалоговые программы двух типов:

- Сначала мы создадим *простую программу*, отображающую диалоговое окно *без* главного окна или окна представления.
- Затем разработаем *полнофункциональную программу* для *просмотра форм*, отображающую главное окно с объектами пользовательского интерфейса и окно представления, основанное на шаблоне диалогового окна, с элементами управления.

#### ***Простая программа с окном диалога DialogView***

---

Разрабатываемая программа не создает главное окно или окно представления, а лишь отображает диалоговое окно (создание диалоговых окон см. в гл. 15). При написании такой программы мастер Application Wizard генерирует только классы приложения и диалогового окна. Следовательно, эта модель используется для простых утилит и различных программ, не управляющих документами. В следующих параграфах разрабатывается программа DialogView, позволяющая пользователю выбрать цвета, а затем отобразить результат их смешивания в заданной области диалогового окна. Сначала для создания основной оболочки программы используется мастер Application Wizard, а затем необходимые средства добавляются в программу.

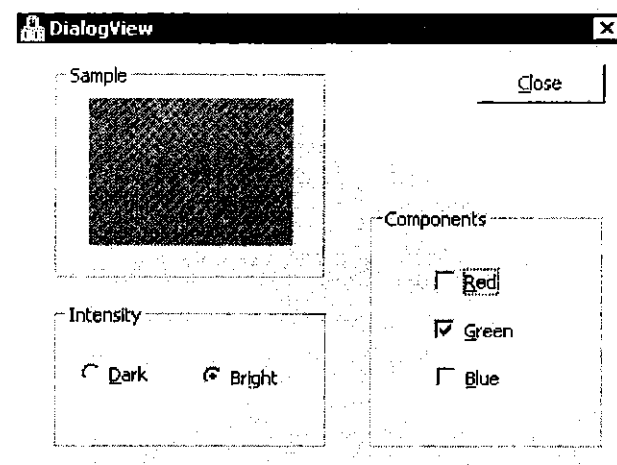
#### ***Создание программы DialogView***

Создание программы DialogView начинается с формирования нового проекта типа MFC Application с именем DialogView. Кроме того, необходимо на вкладке Application Type окна Application Wizard выбрать установку Dialog Based. Для диалоговой программы мастер AppWizard генерирует два основных класса: приложения и диалогового окна.

- *Класс приложения* управляет программой в целом и служит для отображения ее диалогового окна. Способ отображения диалогового окна рассмотрим позже. В программе DialogView класс приложения назван CDialogViewApp и определен в файлах DialogView.h и DialogView.cpp.

- *Класс диалогового окна* управляет окном программы. В программе DialogView класс диалогового окна – CDialogViewDlg – определен в файлах DialogViewDlg.h и DialogViewDlg.cpp. При выборе опции About Box будет сгенерирован еще один класс – для управления диалоговым окном About.

Так как диалоговое приложение не имеет главного окна или окна представления, мастер Application Wizard не генерирует соответствующие классы (класс документа также не генерируется).



## Формирование диалогового окна

Сформируем диалоговое окно программы DialogView, добавив в него элементы управления, и напомним для их поддержки программный код. Чтобы открыть макет диалогового окна программы, откройте вкладку Resource View и выполните двойной щелчок на идентификаторе IDD\_DIALOGVIEW\_DIALOG в разделе Dialog. В диалоговом окне удалите кнопку OK и надпись “TODO”. Щелкните правой кнопкой мыши на кнопке Cancel и выберите в контекстном меню команду Properties. В открывшемся диалоговом окне Properties замените название кнопки Cancel на Close. Она будет использоваться для завершения программы, а не отмены выбора. Добавьте в диалоговое окно элементы управления (табл. 16.1), показанные на рисунке, приведенном выше.

Табл. 16.1. Свойства элементов управления, добавляемых в диалоговое окно программы DialogView

Идентификатор (ID)	Тип элемента управления в разделе Dialog Controls панели инструментов	Задаваемые свойства
IDC_STATIC	Рамка	Надпись (Caption): Components
IDC_RED	Флажок	Надпись (Caption): &Red Group
IDC_GREEN	Флажок	Надпись (Caption): &Green
IDC_BLUE	Флажок	Надпись (Caption): &Blue
IDC_STATIC	Рамка	Надпись (Caption): Color Brightness
IDC_DARK	Переключатель	Надпись (Caption): &Dark Group Tab Stop
IDC_BRIGHT	Переключатель	Надпись (Caption): &Bright
IDC_SAMPLE	Рамка	Надпись (Caption): Sample

Установите желаемый порядок обхода. Смысл данного действия объяснялся в гл. 15, там же описана процедура определения порядка обхода.

Определите переменные для некоторых элементов управления диалогового окна (добавьте их в определение класса `CDialogViewDlg`) согласно таблице 16.2. Они используются для прорисовки цветового образца соответствующей подпрограммой.

Табл. 16.2. Элементы данных для добавления в класс `CDialogViewDlg`

Идентификатор элемента управления	Имя переменной	Категория	Тип переменной
IDC_RED	m_Red	Value	BOOL
IDC_GREEN	m_Green	Value	BOOL
IDC_BLUE	m_Blue	Value	BOOL
IDC_DARK	m_Brightness	Value	int

Добавьте обработчики сообщения `BnClicked` для каждого из следующих элементов управления: `IDC_RED`, `IDC_GREEN`, `IDC_BLUE`, `IDC_DARK` и `IDC_BRIGHT`. Для каждой функции примите имя, заданное по умолчанию. Откройте исходный файл `DialogViewDlg.cpp` и добавьте в сгенерированные обработчики сообщений следующий текст.

```
void CDialogViewDlg::OnBnClickedBlue()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_Blue = IsDlgButtonChecked (IDC_BLUE);
    InvalidateRect (&m_RectSample);
    UpdateWindow ();
}

void CDialogViewDlg::OnBnClickedBright()
{
    // TODO: Добавьте сюда собственный код обработчика
    if (IsDlgButtonChecked (IDC_BRIGHT))
    {
        m_Brightness = INT_BRIGHT;
        InvalidateRect (&m_RectSample);
        UpdateWindow ();
    }
}

void CDialogViewDlg::OnBnClickedDark()
{
    // TODO: Добавьте сюда собственный код обработчика
    if (IsDlgButtonChecked (IDC_DARK))
    {
        m_Brightness = INT_DARK;
        InvalidateRect (&m_RectSample);
        UpdateWindow ();
    }
}

void CDialogViewDlg::OnBnClickedGreen()
{
    // TODO: Добавьте сюда собственный код обработчика
```

```

    m_Green = IsDlgButtonChecked (IDC_GREEN);
    InvalidateRect (&m_RectSample);
    UpdateWindow ();
}

void CDialogViewDlg::OnBnClickedRed()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_Red = IsDlgButtonChecked (IDC_RED);
    InvalidateRect (&m_RectSample);
    UpdateWindow ();
}

```

Если флажок Blue установлен, то обработчик присвоит переменной m\_Blue значение TRUE, а если его сбросить, то установится значение FALSE. Затем обработчик вызовет функции InvalidateRect и UpdateWindow для перерисовки цветового образца в области Sample с помощью функции OnPaint (используя значение переменной m\_Blue). Функции InvalidateRect и UpdateWindow описаны ранее. Обработчики флажков Green и Red работают так же, как и обработчик флажка Blue. Обработчики позиций переключателей Dark и Bright устанавливают значение переменной m\_Brightness и перерисовывают область Sample только в том случае, если позиция, в которую установлен переключатель, была изменена. В некоторых случаях обработчик сообщения BnClicked вызывается, даже когда переключатель не меняет состояния (см. гл. 15).

В обработчик сообщения OnInitDialog в файле DialogViewDlg.cpp добавьте строки, выделенные полужирным в приведенном ниже листинге. В добавленном фрагменте определяются координаты прямоугольника, в котором выводится цветовой образец, после чего они сохраняются в объекте m\_RectSample класса CRect. Такой же фрагмент добавлен в программу FontView (см. гл. 15), кроме вызова функции CRect::InflateRect для объекта m\_RectSample, уменьшающей размер прямоугольника, чтобы между цветовым прямоугольником и рамкой Sample был небольшой зазор.

```

BOOL CDialogViewDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // Добавление элемента "About..." в системное меню.

    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty())
        {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
                                strAboutMenu);
        }
    }

    // Установка значка для данного диалога. Среда делает это
    // автоматически, если главное окно приложения - не диалоговое
    SetIcon(m_hIcon, TRUE); // Установка большого значка
}

```

```

SetIcon(m_hIcon, FALSE);    // Установка маленького значка

// TODO: Добавьте сюда дополнительный код инициализации

GetDlgItem (IDC_SAMPLE)->GetWindowRect (&m_RectSample);
ScreenToClient (&m_RectSample);
int Border = (m_RectSample.right - m_RectSample.left) / 8;
m_RectSample.InflateRect (-Border, -Border);

return TRUE; // Вернуть TRUE, если элементы управления
             // окна не получили фокуса
}

```

Измените в файле DialogViewDlg.cpp присваивание переменной m\_Brightness в конструкторе класса диалогового окна так, чтобы она инициализировалась значением INT\_BRIGHT. В результате первоначально выбранной будет позиция Bright переключателя. Изменение сгенерированного значения, используемого для инициализации переменной класса диалогового окна – это исключение из правил (в отличие от редактирования кода внутри разделов, отмеченных специальными комментариями).

```

CDialogViewDlg::CDialogViewDlg(CWnd* pParent /*=NULL*/)
: CDialog(CDialogViewDlg::IDD, pParent)
, m_Blue(FALSE)
, m_Green(FALSE)
, m_Red(FALSE)
, m_Brightness(INT_BRIGHT)
{
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

```

Удалите вызов функции CDialog::OnPaint в файле DialogViewDlg.cpp внутри оператора else функции OnPaint и замените его строками, выделенными полужирным в приведенном ниже фрагменте листинга. Код, сгенерированный мастером Application Wizard в операторе if, выводит значок программы, когда ее окно минимизировано. Код, добавленный к оператору else, закрашивает прямоугольник внутри рамки Sample. Цвет прямоугольника определяется смешиванием светлых или темных основных цветов, выбранных пользователем (в гл. 19 описана методика формирования цвета).

```

else
{
    COLORREF Color = RGB
    (m_Red ? (m_Brightness==INT_DARK ? 64 : 192) : 0,
    m_Green ? (m_Brightness==INT_DARK ? 64 : 192) : 0,
    m_Blue ? (m_Brightness==INT_DARK ? 64 : 192) : 0);
    CBrush Brush (Color);
    CPaintDC dc(this);
    dc.FillRect (&m_RectSample, &Brush);
}
}

```

В файле DialogViewDlg.h добавьте в начало объявления класса CDialogViewDlg определение переменной m\_RectSample и перечисления задающего выбранную интенсивность цвета (для позиций переключателя).

```

// Диалог CDialogViewDlg
class CDialogViewDlg : public CDialog
{
// Конструктор

```

```

public:
    CRect m_RectSample;
    enum {INT_DARK, INT_BRIGHT};

public:
    CDialogViewDlg(CWnd* pParent = NULL);    // стандартный конструктор

```

Функция `InitInstance`, сгенерированная мастером `Application Wizard`, не создает шаблон документа и не выполняет вызова функции `ProcessShellCommand` для обработки командных строк, как в предыдущих программах. Вместо этого она создает объект класса диалогового окна `CDialogViewDlg`. Затем вызывается функция `CDialog::DoModal` для объекта диалогового окна. Функция `DoModal` создает и отображает диалоговое окно, основанное на шаблоне `IDD_DIALOGVIEW_DIALOG`, который мы редактировали. После закрытия окна функция `DoModal` завершается. Если программа выполняет какие-то действия в зависимости от значений, введенных в диалоговое окно, то она должна проверить значение, возвращенное функции `DoModal`. Если закрыть диалоговое окно, нажав кнопку `Close`, то функция `DoModal` возвратит значение `IDCANCEL`. После щелчка на кнопке `OK` (с идентификатором `IDOK`) функция `DoModal` возвращает значение `IDOK`. В этом случае содержимое всех элементов управления передается в соответствующие переменные, и функция `InitInstance` считывает эти элементы по мере необходимости.

В завершение функция `InitInstance` возвращает значение `FALSE` (в предыдущих программах она возвращала `TRUE`), чтобы MFC завершила выполнение программы, а не продолжила обработку сообщений.

## Текст программы *DialogView*

Теперь можно построить и выполнить программу `DialogView`. В листингах 16.1—16.4 приведены исходные тексты программы `DialogView`.

---

### Листинг 16.1.

```

// DialogView.h : основной заголовочный файл приложения
//

#pragma once

#ifndef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"           // основные символы

// CDialogViewApp:
// Смотрите реализацию этого класса в файле DialogView.cpp
//

class CDialogViewApp : public CWinApp
{
public:
    CDialogViewApp();

// Переопределения
public:
    virtual BOOL InitInstance();

```

```
// Реализация

DECLARE_MESSAGE_MAP()
};

extern CDialogViewApp theApp;
```

---

## Листинг 16.2.

```
// DialogView.cpp : Определяет поведение классов приложения
//

#include "stdafx.h"
#include "DialogView.h"
#include "DialogViewDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CDialogViewApp

BEGIN_MESSAGE_MAP(CDialogViewApp, CWinApp)
    ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()

// Конструктор класса CDialogViewApp

CDialogViewApp::CDialogViewApp()
{
    // TODO: добавьте сюда собственный код конструктора.
    // Поместите весь существенный код инициализации в
    // функцию InitInstance
}

// Единственный объект класса CDialogViewApp

CDialogViewApp theApp;

// Инициализация CDialogViewApp

BOOL CDialogViewApp::InitInstance()
{
    CWinApp::InitInstance();

    AfxEnableControlContainer();

    CDialogViewDlg dlg;
    m_pMainWnd = &dlg;
    INT_PTR nResponse = dlg.DoModal();
    if (nResponse == IDOK)
    {
        // TODO: Поместите сюда код обработчика
        // завершения работы кнопкой OK
    }
}
```



```

else if (nResponse == IDCANCEL)
{
    // TODO: Поместите сюда код обработчика
    // завершения работы кнопкой Cancel
}

// Поскольку диалог был закрыт, вернуть FALSE для завершения
// приложения вместо начала обработки карты сообщений
return FALSE;
}

```

---

### Листинг 16.3.

```

// DialogViewDlg.h : файл заголовка
//

#pragma once

// Диалог CDialogViewDlg
class CDialogViewDlg : public CDialog
{
// Конструктор
public:
    CRect m_RectSample;
    enum {INT_DARK, INT_BRIGHT};

public:
    CDialogViewDlg(CWnd* pParent = NULL);    // стандартный конструктор

// Данные для диалога
    enum { IDD = IDD_DIALOGVIEW_DIALOG };

protected:
    virtual void DoDataExchange(CDataExchange* pDX);
    // поддержка DDX/DDV

// Реализация
protected:
    HICON m_hIcon;

    // Сгенерированные функции карты сообщений
    virtual BOOL OnInitDialog();
    afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
    afx_msg void OnPaint();
    afx_msg HCURSOR OnQueryDragIcon();
    DECLARE_MESSAGE_MAP()

public:
    BOOL m_Blue;
    BOOL m_Green;
    BOOL m_Red;
    BOOL m_Brightness;
    afx_msg void OnBnClickedBlue();
    afx_msg void OnBnClickedBright();
    afx_msg void OnBnClickedDark();
    afx_msg void OnBnClickedGreen();
    afx_msg void OnBnClickedRed();
};

```

```

// DialogViewDlg.cpp : файл реализации
//

#include "stdafx.h"
#include "DialogView.h"
#include "DialogViewDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CAboutDlg диалог, используемый в App About

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

// Данные для диалога
    enum { IDD = IDD_ABOUTBOX };

protected:
    virtual void DoDataExchange(CDataExchange* pDX);
    // поддержка DDX/DDV

// реализация
protected:
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
END_MESSAGE_MAP()

// Диалог CDialogViewDlg

CDialogViewDlg::CDialogViewDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CDialogViewDlg::IDD, pParent)
    , m_Blue(FALSE)
    , m_Green(FALSE)
    , m_Red(FALSE)
    , m_Brightness(INT_BRIGHT)
{
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CDialogViewDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    DDX_Check(pDX, IDC_BLUE, m_Blue);
}

```

```

        DDX_Check(pDX, IDC_GREEN, m_Green);
        DDX_Check(pDX, IDC_RED, m_Red);
        DDX_Radio(pDX, IDC_BRIGHT, m_Brightness);
    }

BEGIN_MESSAGE_MAP(CDialogViewDlg, CDialog)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    //}}AFX_MSG_MAP
    ON_BN_CLICKED(IDC_BLUE, OnBnClickedBlue)
    ON_BN_CLICKED(IDC_BRIGHT, OnBnClickedBright)
    ON_BN_CLICKED(IDC_DARK, OnBnClickedDark)
    ON_BN_CLICKED(IDC_GREEN, OnBnClickedGreen)
    ON_BN_CLICKED(IDC_RED, OnBnClickedRed)
END_MESSAGE_MAP()

// Обработчики сообщений класса CDialogViewDlg

BOOL CDialogViewDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // Добавление элемента "About..." в системное меню.

    ASSERT((IDM_ABOUTBOX & 0xFFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty())
        {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
                                strAboutMenu);
        }
    }

    // Установка значка для данного диалога. Среда делает это
    // автоматически, если главное окно приложения - не диалоговое
    SetIcon(m_hIcon, TRUE);      // Установка большого значка
    SetIcon(m_hIcon, FALSE);    // Установка маленького значка

    // TODO: Добавьте сюда дополнительный код инициализации

    GetDlgItem(IDC_SAMPLE)->GetWindowRect (&m_RectSample);
    ScreenToClient (&m_RectSample);
    int Border = (m_RectSample.right - m_RectSample.left) / 8;
    m_RectSample.InflateRect (-Border, -Border);

    return TRUE;
}

// Вернуть TRUE, если элементы управления окна не получили фокуса
}

void CDialogViewDlg::OnSysCommand(UINT nID, LPARAM lParam)

```

```

        if ((nID & 0xFFF0) == IDM_ABOUTBOX)
        {
            CAboutDlg dlgAbout;
            dlgAbout.DoModal();
        }
        else
        {
            CDialog::OnSysCommand(nID, lParam);
        }
    }

// Если в приложении есть кнопка Minimize, нижеследующий код
// используется для отображения значка. Для MFC-приложений,
// использующих модель документ/представление, это действие
// производится средой.

void CDialogViewDlg::OnPaint()
{
    if (IsIconic())
    {
        CPaintDC dc(this); // контекст устройства для рисования

        SendMessage(WM_ICONERASEBKGND,
            reinterpret_cast<WPARAM>(dc.GetSafeHdc()), 0);

        // Центровка значка в клиентском прямоугольнике
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
        int x = (rect.Width() - cxIcon + 1) / 2;
        int y = (rect.Height() - cyIcon + 1) / 2;

        // Рисование значка
        dc.DrawIcon(x, y, m_hIcon);
    }
    else
    {
        COLORREF Color = RGB
            (m_Red ? (m_Brightness==INT_DARK ? 64 : 192) : 0,
            m_Green ? (m_Brightness==INT_DARK ? 64 : 192) : 0,
            m_Blue ? (m_Brightness==INT_DARK ? 64 : 192) : 0);
        CBrush Brush(Color);
        CPaintDC dc(this);
        dc.FillRect(&m_RectSample, &Brush);
    }
}

// Система вызывает эту функцию для получения курсора для отображения
// при перетаскивании минимизированного окна.
HCURSOR CDialogViewDlg::OnQueryDragIcon()
{
    return static_cast<HCURSOR>(m_hIcon);
}

void CDialogViewDlg::OnBnClickedBlue()

```

```

{
    // TODO: Добавьте сюда собственный код обработчика
    m_Blue = IsDlgButtonChecked (IDC_BLUE);
    InvalidateRect (&m_RectSample);
    UpdateWindow ();
}

void CDialogViewDlg::OnBnClickedBright()
{
    // TODO: Добавьте сюда собственный код обработчика
    if (IsDlgButtonChecked (IDC_BRIGHT))
    {
        m_Brightness = INT_BRIGHT;
        InvalidateRect (&m_RectSample);
        UpdateWindow ();
    }
}

void CDialogViewDlg::OnBnClickedDark()
{
    // TODO: Добавьте сюда собственный код обработчика
    if (IsDlgButtonChecked (IDC_DARK))
    {
        m_Brightness = INT_DARK;
        InvalidateRect (&m_RectSample);
        UpdateWindow ();
    }
}

void CDialogViewDlg::OnBnClickedGreen()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_Green = IsDlgButtonChecked (IDC_GREEN);
    InvalidateRect (&m_RectSample);
    UpdateWindow ();
}

void CDialogViewDlg::OnBnClickedRed()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_Red = IsDlgButtonChecked (IDC_RED);
    InvalidateRect (&m_RectSample);
    UpdateWindow ();
}

```

## ***Программа просмотра формы FormView***

В этом разделе будет создана программа просмотра формы, которая представляет собой полнофункциональное приложение, ориентированное на использование диалогового окна. Оно имеет ту же базовую архитектуру, что и программы MFC, рассмотренные в предыдущих главах. Основное отличие этой программы состоит в том, что ее класс представления порождается от класса CFormView, а не от классов представления MFC (таких, как CView, CScrollView или CEditView). Поэтому соответствующее окно представления отображает совокупность элементов управления, а не пустую рабочую область. Элементы размещаются в соответствии с шаблоном диалогового окна. Приложения данного типа хорошо подходят для разработки, сохранения, отображения и изменения отдельных элементов структурированных

документов (например, записей баз данных). От простой диалоговой программы, описанной в предыдущих разделах, программа просмотра формы отличается тем, что имеет четыре стандартных MFC-класса:

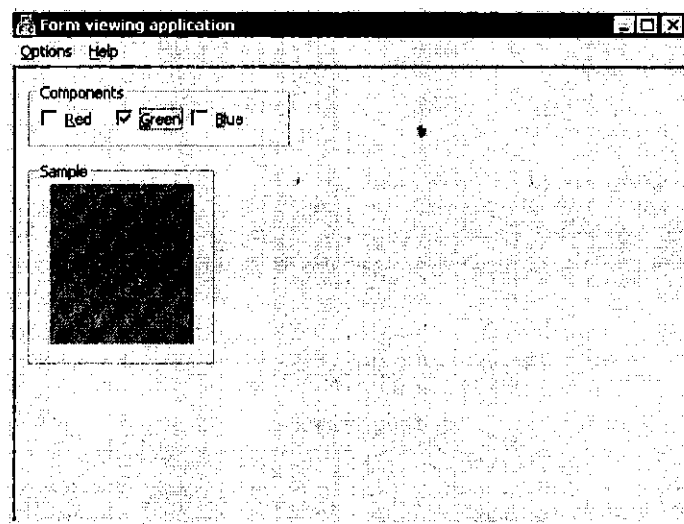
- класс приложения;
- класс главного окна;
- класс документа;
- класс представления.

Данное приложение имеет главное окно со стандартными компонентами окна (такими как рамка с изменяемыми размерами и кнопка максимизации). В нем можно использовать любой из стандартных элементов пользовательского интерфейса MFC (меню, панель инструментов или строку состояния), а любой из основных классов программы может обрабатывать сообщения, передаваемые этими элементами. Так как класс `CFormView` порожден от класса `CScrollView` (см. гл. 13), то в случае, если все элементы управления не помещаются внутри окна представления, оно отобразит горизонтальную или вертикальную, или обе полосы прокрутки, которые можно использовать, чтобы получить доступ к любому элементу управления.

Разрабатываемая ниже программа `FormView` напоминает созданную ранее программу `DialogView`, однако дает возможность пользователю менять размеры окна, так как окно программы имеет рамку изменяемого размера и кнопку максимизации. Если уменьшить размер окна так, что элементы управления будут видны не полностью, то отобразятся полосы прокрутки, позволяющие просматривать содержимое окна. Окно программы содержит меню, содержащее, среди прочего, пункт `Help` с командой `About`. С помощью команд меню `Options` можно задать интенсивность цвета.

## Генерация и настройка программы `FormView`

Базовая часть программы `FormView` генерируется с помощью мастера `Application Wizard`. Выполняемые при этом пользователем действия описаны в гл. 9 (при создании программы `WinHello`). Во вкладке диалогового окна мастера `Application Wizard` установите опцию `Single Document`, но *не* задавайте опцию `Dialog Based`. Введите в диалоговые окна мастера `Application Wizard` те же значения, которые вводились при создании программы `WinHello`, но отключите строку состояния и панель инструментов для создаваемого приложения. Чтобы класс представления был порожден от класса `CFormView`, на вкладке `Generated Classes` необходимо выбрать класс представления `CFormViewView` в списке создаваемых классов и выбрать класс `CFormView` в списке `Base Class`.



Для настройки диалогового окна программы оно должно быть открыто в редакторе диалоговых окон Visual Studio. В противном случае откройте вкладку Resource View и выполните двойной щелчок на идентификаторе IDD\_FORMVIEW\_FORM. Шаблон диалогового окна IDD\_FORMVIEW\_FORM создан мастером Application Wizard и связан с классом представления, т.е. окно представления автоматически отображает элементы управления, содержащиеся внутри данного шаблона. Обратите внимание: первоначально шаблон содержит только надпись "TODO". Не изменяйте свойства диалогового окна (откройте окно Properties, чтобы посмотреть их), так как значения, первоначально заданные мастером Application Wizard, соответствуют его шаблону, отображающемуся в окне формы. В этом шаблоне не задается рамка, строка заголовка и другие видимые элементы, потому что они предоставляются главным окном, содержащим окно представления. Удалите существующую надпись "TODO" и добавьте описанные в табл. 16.3 элементы управления.

Табл. 16.3. Свойства элементов управления, добавляемых в диалоговое окно IDD\_FORMVIEW\_FORM

Идентификатор (ID)	Тип элемента управления в разделе Dialog Controls панели инструментов	Устанавливаемые свойства
IDC_STATIC	Рамка	Надпись (Caption): Components
IDC_RED	Флажок	Надпись (Caption): &Red Group
IDC_GREEN	Флажок	Надпись (Caption): &Green
IDC_BLUE	Флажок	Надпись (Caption): &Blue
IDC_SAMPLE	Рамка	Надпись (Caption): &Sample

Для флажков IDC\_RED, IDC\_GREEN и IDC\_BLUE определите соответствующие переменные. Для каждой переменной необходимо принять стандартную категорию, тип и имя – m\_Red, m\_Green и m\_Blue. Откройте редактор меню для меню IDR\_MAINFRAME. Удалите меню Edit и измените заголовок меню File на &Options. Теперь удалите все команды в меню Options за исключением команды Exit и разделителя над ней. Наконец, над разделителем добавьте команды Bright Colors и Dark Colors (табл. 16.4).

Табл. 16.4. Новые команды меню Options

Идентификатор (ID)	Надпись (Caption)
ID_OPTIONS_BRIGHT	&Bright Colors
ID_OPTIONS_DARK	&Dark Colors

В класс CFormViewView добавьте перечисленные ниже обработчики сообщений. (В программе FormView сообщения от добавленных элементов меню и кнопок обрабатываются классом представления.) Сгенерируйте обработчики сообщений COMMAND и UPDATE\_COMMAND\_UI для команд меню ID\_OPTIONS\_DARK и ID\_OPTIONS\_BRIGHT. Определите обработчик сообщения BnClicked для флагов IDC\_RED, IDC\_GREEN и IDC\_BLUE. Для всех функций примите имена, заданные по умолчанию. Откройте файл FormViewView.cpp и добавьте к сгенерированным обработчикам сообщений следующие строки.

```
void CFormViewView::OnBnClickedBlue()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_Blue = IsDlgButtonChecked (IDC_BLUE);

    CCClientDC ClientDC(this);
```

```

        OnPrepareDC (&ClientDC);
        CRect Rect = m_RectSample;
        ClientDC.LPtoDP (&Rect);
        InvalidateRect (&Rect);
        UpdateWindow ();
    }

void CFormViewView::OnBnClickedGreen()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_Green = IsDlgButtonChecked (IDC_GREEN);

    CClientDC ClientDC(this);
    OnPrepareDC (&ClientDC);
    CRect Rect = m_RectSample;
    ClientDC.LPtoDP (&Rect);
    InvalidateRect (&Rect);
    UpdateWindow ();
}

void CFormViewView::OnBnClickedRed()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_Red = IsDlgButtonChecked (IDC_RED);

    CClientDC ClientDC(this);
    OnPrepareDC (&ClientDC);
    CRect Rect = m_RectSample;
    ClientDC.LPtoDP (&Rect);
    InvalidateRect (&Rect);
    UpdateWindow ();
}

void CFormViewView::OnOptionsBrightcolors()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_Intensity = INT_BRIGHT;

    CClientDC ClientDC(this);
    OnPrepareDC (&ClientDC);
    CRect Rect = m_RectSample;
    ClientDC.LPtoDP (&Rect);
    InvalidateRect (&Rect);
    UpdateWindow ();
}

void CFormViewView::OnUpdateOptionsBrightcolors(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetRadio (m_Intensity == INT_BRIGHT);
}

void CFormViewView::OnOptionsDarkcolors()
{

```



```

// TODO: Добавьте сюда собственный код обработчика
m_Intensity = INT_DARK;

CClientDC ClientDC(this);
OnPrepareDC (&ClientDC);
CRect Rect = m_RectSample;
ClientDC.LPtoDP (&Rect);
InvalidateRect (&Rect);
UpdateWindow ();
}

void CFormViewView::OnUpdateOptionsDarkcolors(CCmdUI *pCmdUI)
{
// TODO: Добавьте сюда собственный код обработчика
pCmdUI->SetRadio (m_Intensity = INT_DARK);
}

```

Добавленный в обработчики сообщений код напоминает добавления к программе DialogView. Так как интенсивность цвета устанавливается с помощью команд меню, а не позиций переключателя, программа должна предоставить обработчики, обновляющие каждую такую команду (OnUpdateOptionsDark и OnUpdateOptionsBright). Обратите внимание: обработчики вызывают функцию CCmdUI::SetRadio вместо CCmdUI::SetCheck, чтобы выбранная команда меню была отмечена специальным значком позиции переключателя, а не флажком (совокупность позиций переключателя обычно используется для обозначения группы взаимоисключающих опций). Как и в программе DialogView, обработчики сообщений делают недействительной прямоугольную область диалогового окна, в которой отображен цветовой образец. Однако они сначала преобразовывают координаты области образца из логических в координаты устройства (такое преобразование рассмотрено в параграфе “Логические и фактические координаты” гл. 13). Этот шаг необходим, потому что класс окна представления косвенно порожден от класса CScrollView и, если все элементы управления не помещаются внутри окна представления, можно прокручивать его содержимое. Таким образом, логические координаты области образца, сохраненные в переменной m\_RectSample, отличаются от координат устройства (последние должны передаваться в функцию InvalidateRect).

Добавьте в класс представления переопределенную версию функции OnDraw (для программы просмотра формы мастер Application Wizard по умолчанию не включает функцию OnDraw). Добавьте следующие операторы в код функции (они работают так же, как и добавленные в программу DialogView).

```

void CFormViewView::OnDraw(CDC* pDC)
{
// TODO: Добавьте сюда собственный код обработчика или
// вызов базового класса
COLORREF Color = RGB
    (m_Red ? (m_Intensity==INT_DARK ? 64 : 192) : 0,
    m_Green ? (m_Intensity==INT_DARK ? 64 : 192) : 0,
    m_Blue ? (m_Intensity==INT_DARK ? 64 : 192) : 0);
CBrush Brush (Color);
pDC->FillRect (&m_RectSample, &Brush);
}

```

Вставьте в функцию OnInitialUpdate приведенный ниже код.

```

void CFormViewView::OnInitialUpdate()
{
    CFormView::OnInitialUpdate();
    GetParentFrame()->RecalcLayout();
}

```

```

ResizeParentToFit();

GetDlgItem (IDC_SAMPLE)->GetWindowRect (&m_RectSample);
ScreenToClient (&m_RectSample);
int Border = (m_RectSample.right - m_RectSample.left) / 8;
m_RectSample.InflateRect (-Border, -Border);
}

```

Непосредственно перед *первым* отображением документа в окне представления вызывается виртуальная функция OnInitialUpdate класса представления (см. гл. 13). Для размещения всех управляющих панелей программы (в программе FormView они отсутствуют) код, сгенерированный мастером Application Wizard, после вызова версии функции OnInitialUpdate базового класса вызывает функцию RecalcLayout объекта главного окна. Чтобы главное окно отображало содержимое окна представления, программа вызывает функцию CScrollView::ResizeParentToFit. Четыре новых оператора совпадают с операторами, добавленными в функцию OnInitDialog программы DialogView. Этот код добавлен в функцию OnInitialUpdate, а не в функцию OnInitDialog, потому что окно представления не получает сообщения WM\_INITDIALOG, и, следовательно, функция OnInitDialog никогда не получила бы управление. При вызове функции OnInitialUpdate окно представления еще не было прокручено, поэтому в переменной m\_RectSample сохраняются координаты устройства, совпадающие с логическими координатами области образца. После прокрутки окна в этой переменной будут находиться только логические координаты.

В файле FormViewView.cpp добавьте код инициализации переменной m\_Intensity в конструктор класса представления.

```

// Конструктор CFormViewView

CFormViewView::CFormViewView()
: CFormView(CFormViewView::IDD)
, m_Red(FALSE)
, m_Green(FALSE)
, m_Blue(FALSE)
, m_Intensity (INT_BRIGHT)
{
    // TODO: добавьте сюда собственный код конструктора
}

```

Добавьте в файл FormViewView.h определения констант INT\_DARK и INT\_LIGHT и определите переменные m\_DialogBrush, m\_Intensity и m\_RectSample класса представления.

```

class CFormViewView : public CFormView
{
public:
    CBrush m_DialogBrush;
    int m_Intensity;
    CRect m_RectSample;

    enum {INT_DARK, INT_BRIGHT};

protected: // используется только для сериализации
    CFormViewView();
    DECLARE_DYNCREATE(CFormViewView)
}

```

Вызовите функцию SetWindowText из функции InitInstance в файле FormView.cpp, чтобы задать строку заголовка программы.

```

        m_pMainWnd->ShowWindow(SW_SHOW);
        m_pMainWnd->UpdateWindow();

        m_pMainWnd->SetWindowText ("Form viewing application");
        return TRUE;
}

```

## Текст программы FormView

Теперь программу FormView можно построить и выполнить. В листингах 16.5—16.12 приведены исходные тексты программы FormView.

---

### Листинг 16.5.

```

// FormView.h : главный заголовочный файл приложения FormView
//

#pragma once

#ifndef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"          // основные символы

// CFormViewApp:
// Смотрите реализацию этого класса в файле FormView.cpp
//

class CFormViewApp : public CWinApp
{
public:
    CFormViewApp();

    // Переопределения
public:
    virtual BOOL InitInstance();

    // Реализация
    afx_msg void OnAppAbout();
    DECLARE_MESSAGE_MAP()
};

extern CFormViewApp theApp;

```

---

### Листинг 16.6.

```

// FormView.cpp : Определяет поведение классов приложения
//

#include "stdafx.h"
#include "FormView.h"
#include "MainFrm.h"

#include "FormViewDoc.h"

```

```

#include "FormViewView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CFormViewApp

BEGIN_MESSAGE_MAP(CFormViewApp, CWinApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
    // Стандартные команды работы с файлами документов
    ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
END_MESSAGE_MAP()

// Конструктор CFormViewApp

CFormViewApp::CFormViewApp()
{
    // TODO: добавьте сюда собственный код конструктора.
    // Поместите весь существенный код инициализации
    // в функцию InitInstance
}

// Единственный объект класса CFormViewApp

CFormViewApp theApp;

// Инициализация CFormViewApp

BOOL CFormViewApp::InitInstance()
{
    CWinApp::InitInstance();

    // Инициализация библиотек OLE
    if (!AfxOleInit())
    {
        AfxMessageBox(IDP_OLE_INIT_FAILED);
        return FALSE;
    }
    AfxEnableControlContainer();

    // Стандартная инициализация.
    // Если вы не используете какие-то из предоставленных
    // возможностей и хотите уменьшить размер конечного
    // исполняемого модуля, удалите из последующего кода
    // отдельные команды инициализации элементов, которые
    // вам не нужны.
    // Измените строку - аргумент функции (ключ, под которым
    // ваши установки хранятся в реестре).
    // TODO: Измените эту строку на что-нибудь подходящее,
    // например, на имя вашей компании или организации
    SetRegistryKey(_T("Local AppWizard-Generated Applications"));
    LoadStdProfileSettings(4); // Загрузка стандартных установок
                               // из INI-файла (включая MRU).
    // Регистрация шаблона документа приложения. Шаблоны

```

```

// документов служат связью между документами, окнами
// документов и окнами приложений

CSingleDocTemplate* pDocTemplate;
pDocTemplate = new CSingleDocTemplate(
    IDR_MAINFRAME,
    RUNTIME_CLASS(CFormViewDoc),
    RUNTIME_CLASS(CMainFrame),
    // основное окно SDI-приложения
    RUNTIME_CLASS(CFormViewView));
AddDocTemplate(pDocTemplate);
// Просмотр командной строки для обнаружения стандартных
// команд оболочки, DDE, открытия файлов
CCommandLineInfo cmdInfo;
ParseCommandLine(cmdInfo);
// Выполнение команд, указанных в командной строке.
// Вернет FALSE, если приложение было запущено с
// /RegServer, /Register, /Unregserver или /Unregister.
if (!ProcessShellCommand(cmdInfo))
    return FALSE;
// Показ и обновление единственного проинициализированного окна
m_pMainWnd->ShowWindow(SW_SHOW);
m_pMainWnd->UpdateWindow();

m_pMainWnd->SetWindowText ("Form viewing application");
return TRUE;
}

// CAboutDlg диалог, используемый в App About

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

// Данные для диалога
    enum { IDD = IDD_ABOUTBOX };

protected:
    virtual void DoDataExchange(CDataExchange* pDX);
    // Поддержка DDX/DDV

// Реализация
protected:
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
END_MESSAGE_MAP()

```

```
// Команда приложения на запуск диалога
void CFormViewApp::OnAppAbout()
{
    CAboutDlg aboutDlg;
    aboutDlg.DoModal();
}

// Обработчики сообщений класса CFormViewApp
```

---

#### Листинг 16.7.

```
// FormViewDoc.h : интерфейс класса CFormViewDoc
//

#pragma once

class CFormViewDoc : public CDocument
{
protected: // используется только для сериализации
    CFormViewDoc();
    DECLARE_DYNCREATE(CFormViewDoc)

// Атрибуты
public:

// Операции
public:

// Определения
public:
    virtual BOOL OnNewDocument();
    virtual void Serialize(CArchive& ar);

// Реализация
public:
    virtual ~CFormViewDoc();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif
protected:

// Сгенерированные функции карты сообщений
protected:
    DECLARE_MESSAGE_MAP()
};
```

---

#### Листинг 16.8.

```
// FormViewDoc.cpp : реализация класса CFormViewDoc
//

#include "stdafx.h"
#include "FormView.h"

#include "FormViewDoc.h"
```

```

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CFormViewDoc

IMPLEMENT_DYNCREATE(CFormViewDoc, CDocument)

BEGIN_MESSAGE_MAP(CFormViewDoc, CDocument)
END_MESSAGE_MAP()

// Конструктор CFormViewDoc

CFormViewDoc::CFormViewDoc()
{
    // TODO: добавьте сюда однократный код конструктора
}

CFormViewDoc::~CFormViewDoc()
{
}

BOOL CFormViewDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;

    // TODO: добавьте сюда код реинициализации
    // (SDI-документы будут повторно использовать этот документ)

    return TRUE;
}

// Сериализация CFormViewDoc

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

// Диагностика CFormViewDoc

#ifdef _DEBUG
void CFormViewDoc::AssertValid() const
{
    CDocument::AssertValid();
}

```

```

void CFormViewDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}
#endif //_DEBUG

// Команды CFormViewDoc

```

---

### Листинг 16.9.

```

// FormViewView.h : интерфейс класса CFormViewView
//

#pragma once

class CFormViewView : public CFormView
{
public:
    CBrush m_DialogBrush;
    int m_Intensity;
    CRect m_RectSample;

    enum {INT_DARK, INT_BRIGHT};

protected: // используется только для сериализации
    CFormViewView();
    DECLARE_DYNCREATE(CFormViewView)

public:
    enum{ IDD = IDD_FORMVIEW_FORM };

    // Атрибуты
public:
    CFormViewDoc* GetDocument() const;

    // Операции
public:

    // Переопределения
    public:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
protected:
    virtual void DoDataExchange(CDataExchange* pDX);
    // Поддержка DDX/DDV
    virtual void OnInitialUpdate();
    // в первый раз вызывается после конструктора

    // Реализация
public:
    virtual ~CFormViewView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif
}

```



```

protected:

// Сгенерированные функции карты сообщений
protected:
    DECLARE_MESSAGE_MAP()
public:
    BOOL m_Red;
    BOOL m_Green;
    BOOL m_Blue;
protected:
    virtual void OnDraw(CDC* /*pDC*/);
public:
    afx_msg void OnBnClickedBlue();
    afx_msg void OnBnClickedGreen();
    afx_msg void OnBnClickedRed();
    afx_msg void OnOptionsBrightcolors();
    afx_msg void OnUpdateOptionsBrightcolors(CCmdUI *pCmdUI);
    afx_msg void OnOptionsDarkcolors();
    afx_msg void OnUpdateOptionsDarkcolors(CCmdUI *pCmdUI);
};

#ifdef _DEBUG // отладочная версия в файле FormViewView.cpp
inline CFormViewDoc* CFormViewView::GetDocument() const
    { return reinterpret_cast<CFormViewDoc*>(m_pDocument); }
#endif

```

---

#### Листинг 16.10.

```

// FormViewView.cpp : реализация класса CFormViewView
//

#include "stdafx.h"
#include "FormView.h"

#include "FormViewDoc.h"
#include "FormViewView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CFormViewView

IMPLEMENT_DYNCREATE(CFormViewView, CFormView)

BEGIN_MESSAGE_MAP(CFormViewView, CFormView)
    ON_BN_CLICKED(IDC_BLUE, OnBnClickedBlue)
    ON_BN_CLICKED(IDC_GREEN, OnBnClickedGreen)
    ON_BN_CLICKED(IDC_RED, OnBnClickedRed)
    ON_COMMAND(ID_OPTIONS_BRIGHTCOLORS, OnOptionsBrightcolors)
    ON_UPDATE_COMMAND_UI(ID_OPTIONS_BRIGHTCOLORS,
        OnUpdateOptionsBrightcolors)
    ON_COMMAND(ID_OPTIONS_DARKCOLORS, OnOptionsDarkcolors)
    ON_UPDATE_COMMAND_UI(ID_OPTIONS_DARKCOLORS,
        OnUpdateOptionsDarkcolors)
END_MESSAGE_MAP()

```

```

// Конструктор CFormViewView
CFormViewView::CFormViewView()
    : CFormView(CFormViewView::IDD)
    , m_Red(FALSE)
    , m_Green(FALSE)
    , m_Blue(FALSE)
    , m_Intensity (INT_BRIGHT)
{
    // TODO: добавьте сюда собственный код конструктора
}

CFormViewView::~CFormViewView()
{
}

void CFormViewView::DoDataExchange(CDataExchange* pDX)
{
    CFormView::DoDataExchange(pDX);
    DDX_Check(pDX, IDC_RED, m_Red);
    DDX_Check(pDX, IDC_GREEN, m_Green);
    DDX_Check(pDX, IDC_BLUE, m_Blue);
}

BOOL CFormViewView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Измените класс или стили окна здесь,
    // изменяя и добавляя поля структуры CS

    return CFormView::PreCreateWindow(cs);
}

void CFormViewView::OnInitialUpdate()
{
    CFormView::OnInitialUpdate();
    GetParentFrame()->RecalcLayout();
    ResizeParentToFit();

    GetDlgItem (IDC_SAMPLE)->GetWindowRect (&m_RectSample);
    ScreenToClient (&m_RectSample);
    int Border = (m_RectSample.right - m_RectSample.left) / 8;
    m_RectSample.InflateRect (-Border, -Border);
}

// Диагностика CFormViewView
#ifdef _DEBUG
void CFormViewView::AssertValid() const
{
    CFormView::AssertValid();
}

void CFormViewView::Dump(CDumpContext& dc) const
{
    CFormView::Dump(dc);
}

```

```

CFormViewDoc* CFormViewView::GetDocument() const
// не отладочная версия встроена
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CFormViewDoc)));
    return (CFormViewDoc*)m_pDocument;
}
#endif // _DEBUG

// Обработчики сообщений CFormViewView

void CFormViewView::OnDraw(CDC* pDC)
{
    // TODO: Добавьте сюда собственный код обработчика или
    // вызов базового класса
    COLORREF Color = RGB
        (m_Red ? (m_Intensity==INT_DARK ? 64 : 192) : 0,
         m_Green ? (m_Intensity==INT_DARK ? 64 : 192) : 0,
         m_Blue ? (m_Intensity==INT_DARK ? 64 : 192) : 0);
    CBrush Brush (Color);
    pDC->FillRect (&m_RectSample, &Brush);
}

void CFormViewView::OnBnClickedBlue()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_Blue = IsDlgButtonChecked (IDC_BLUE);

    CClientDC ClientDC(this);
    OnPrepareDC (&ClientDC);
    CRect Rect = m_RectSample;
    ClientDC.LPtoDP (&Rect);
    InvalidateRect (&Rect);
    UpdateWindow ();
}

void CFormViewView::OnBnClickedGreen()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_Green = IsDlgButtonChecked (IDC_GREEN);

    CClientDC ClientDC(this);
    OnPrepareDC (&ClientDC);
    CRect Rect = m_RectSample;
    ClientDC.LPtoDP (&Rect);
    InvalidateRect (&Rect);
    UpdateWindow ();
}

void CFormViewView::OnBnClickedRed()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_Red = IsDlgButtonChecked (IDC_RED);

    CClientDC ClientDC(this);
    OnPrepareDC (&ClientDC);
    CRect Rect = m_RectSample;

```

```

        ClientDC.LPtoDP (&Rect);
        InvalidateRect (&Rect);
        UpdateWindow ();
    }

void CFormViewView::OnOptionsBrightcolors()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_Intensity = INT_BRIGHT;

    CClientDC ClientDC(this);
    OnPrepareDC (&ClientDC);
    CRect Rect = m_RectSample;
    ClientDC.LPtoDP (&Rect);
    InvalidateRect (&Rect);
    UpdateWindow ();
}

void CFormViewView::OnUpdateOptionsBrightcolors(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetRadio (m_Intensity = INT_BRIGHT);
}

void CFormViewView::OnOptionsDarkcolors()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_Intensity = INT_DARK;

    CClientDC ClientDC(this);
    OnPrepareDC (&ClientDC);
    CRect Rect = m_RectSample;
    ClientDC.LPtoDP (&Rect);
    InvalidateRect (&Rect);
    UpdateWindow ();
}

void CFormViewView::OnUpdateOptionsDarkcolors(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetRadio (m_Intensity = INT_DARK);
}

```

---

#### Листинг 16.11.

```

// MainFrm.h : интерфейс класса CMainFrame
//

#pragma once
class CMainFrame : public CFrameWnd
{
protected: // используется только для сериализации
    CMainFrame();
    DECLARE_DYNCREATE(CMainFrame)

```

```

// Атрибуты
public:

// Операции
public:

// Переопределения
public:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);

// Реализация
public:
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

// Сгенерированные функции карты сообщений
protected:
    DECLARE_MESSAGE_MAP()
};

```

---

#### Листинг 16.12.

```

// MainFrm.cpp : реализация класса CMainFrame
//

#include "stdafx.h"
#include "FormView.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
END_MESSAGE_MAP()

// Конструктор CMainFrame

CMainFrame::CMainFrame()
{
    // TODO: добавьте сюда код инициализации элементов класса
}

CMainFrame::~CMainFrame()
{
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)

```

```

        if( !CFrameWnd::PreCreateWindow(cs) )
            return FALSE;
        // TODO: Измените класс или стили окна приложения,
        // добавляя и изменяя поля структуры cs

        return TRUE;
    }

    // Диагностика CMainFrame

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
    CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
    CFrameWnd::Dump(dc);
}

#endif // _DEBUG

// Обработчики сообщений класса CMainFrame

```

## Резюме

---

Мы рассмотрели два различных способа создания диалоговых приложений, т.е. программ, в главном окне которых отображается набор определенных элементов управления. Работа таких программ основана на разработанном в редакторе диалоговых окон шаблоне диалогового окна.

- *Программа с простым диалоговым окном.* Для создания такой диалоговой программы необходимо выбрать опцию Dialog Based в диалоговом окне мастера Application Wizard. Полученная программа отобразит в качестве главного диалоговое окно (а не главное окно и окно представления). Создание диалоговых программ с использованием Application Wizard – это лучший способ разработки простых утилит или программ ввода данных, которые не должны управлять документами. Чтобы такая программа отображала диалоговое окно, необходимо модифицировать исходный шаблон окна. Затем добавьте функции и обработчики сообщений в созданный с помощью мастера Application Wizard класс диалогового окна.
- *Диалоговая программа просмотра формы.* Для создания такого диалогового приложения необходимо выбрать опцию Single Document или Multiple Documents в диалоговом окне Application Wizard с последующим заданием класса CFormView как базового класса представления программы. Задание этих параметров позволяет сгенерировать программу *просмотра формы*. В программе просмотра формы окно представления отображает набор элементов управления, размещение которых основано на шаблоне диалогового окна. Мастер Application Wizard создает исходный шаблон. Чтобы модифицировать его, необходимо использовать редактор диалоговых окон. Программа просмотра формы содержит те же классы, окна и элементы пользовательского интерфейса, что и обычная программа, сгенерированная мастером Application Wizard. Это позволяет разработать полнофункциональную программу управления документами. Окно представления позволяет прокручивать шаблон, если целиком он не виден.

## Глава 17

# Мультидокументные программы

---

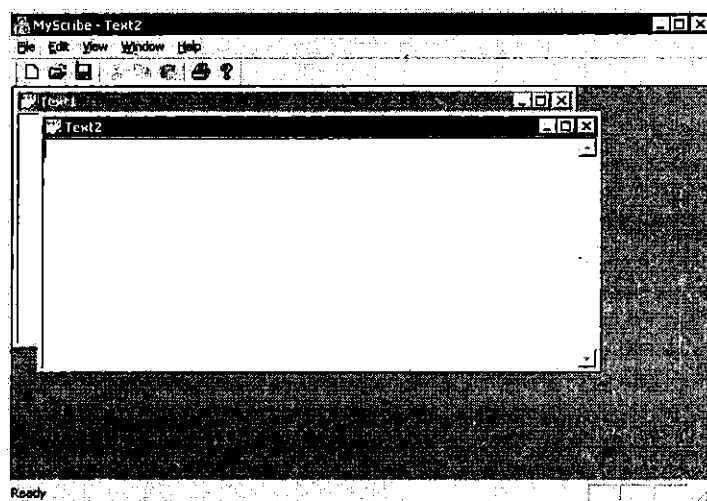
- Мультидокументный интерфейс
- Мультидокументная версия программы MyScribe
- Ресурсы

До сих пор мы создавали программы с *однодокументным интерфейсом*, называемые также SDI-приложениями (SDI – Single Document Interface). SDI-приложения предназначены для одновременного просмотра и редактирования только одного документа. В этой главе вы научитесь создавать программы с *мультидокументным интерфейсом*, называемые MDI-приложениями (MDI – Multiple Document Interface). Программа с мультидокументным интерфейсом позволяет открывать одновременно несколько документов. Каждый из них просматривается и редактируется в отдельном дочернем окне, находящемся в рабочей области приложения внутри главного окна программы. Несмотря на то, что управление несколькими открытыми документами может показаться сложной задачей, библиотека MFC выполняет *большую* часть необходимых действий. Воспользуйтесь средствами MFC, если возможности приложения не выходят за рамки структуры модели стандартного MDI-приложения. В данной главе методы написания MDI-приложений продемонстрированы на примере построения MDI-версии программы MyScribe.

## Мультидокументный интерфейс

---

MDI-версия программы MyScribe при первом запуске открывает в дочернем окне новый пустой документ. Если для создания нового или чтения существующего документа выбрать в меню File команду New или Open..., то созданный/открытый документ отобразится в *отдельном дочернем окне*, не заменяя первоначально открытый (как это имело бы место в программе с однодокументным интерфейсом). Команды Open... и New позволяют открыть необходимое количество документов. На рисунке показано окно программы после открытия нескольких документов. Для работы с одним из них активизируйте соответствующее дочернее окно. Для этого щелкните на нем мышью или выберите его название в меню Window. Можно также активизировать отдельное дочернее окно, нажав (если нужно, несколько раз) клавиши Ctrl+F6, активизирующие *следующее* дочернее окно.



Команды Save, Save As... и Print... из меню File, как и команды меню Edit, воздействуют на активное дочернее окно. Закрывать активное окно документа можно, выбрав команду Close в меню File, либо щелкнув на кнопке закрытия окна документа в правом верхнем углу окна, либо выбрав команду Close системного меню окна документа. Если все документы закрыты, отображаются только меню File и Help, а в меню File отображаются только команды создания нового или открытия существующего документа и выхода из программы. Чтобы окно заполнило всю рабочую область главного окна приложения, щелкните на кнопке разворачивания дочернего окна. Для того чтобы скрыть окно, можно щелкнуть на кнопке свертывания дочернего окна. Оно будет представлено уменьшенной строкой заголовка в рабочей области приложения.

Если выбрать из меню File команду Open... для открытия документа, уже отображенного в дочернем окне, программа активизирует существующее окно, не создавая нового. Для отображения двух или более дочерних окон, отображающих один и тот же документ, воспользуйтесь командой New Window меню Window. Командами меню Window можно упорядочить расположение дочерних окон в виде каскадной (с перекрытиями) или мозаичной (без перекрытия) структуры либо упорядочить заголовки всех свернутых дочерних окон. Последняя процедура выполняется командой Arrange Icon – упорядочить значки (в ранних версиях Windows свернутые окна представлялись значками).

## ***Мультидокументная версия программы MyScribe***

Во всех примерах других глав этой книги для простоты используется программная модель с однодокументным интерфейсом. Однако при создании этих программ можно использовать и мультидокументную модель. В MDI-приложение можно добавить панель инструментов, строку состояния, диалоговую панель (см. гл. 14). Каждое дочернее окно будет содержать собственные полосы прокрутки. Более того, порождая класс представления от класса CFormView (см. гл. 16), в каждом окне представления можно отображать набор элементов управления, основанный на шаблоне диалогового окна.

Чтобы сгенерировать *новый набор* файлов для MDI-версии программы MyScribe воспользуйтесь мастером Application Wizard по методике, описанной в параграфе “Как сгенерировать исходный код” гл. 9. Введите имя программы MyScribe, укажите путь к папке проекта. На вкладках диалогового окна мастера Application Wizard выберите те же опции, что и для программы WinHello в гл. 9, но вместо Single Document выберите опцию Multiple Documents. Это все, что необходимо выполнить, чтобы вместо SDI-приложения создать MDI-приложение. Откройте вкладку Document Template Strings в диалоговом окне Advanced Options. Введите в поле File Extension стандартное расширение файла – txt, в поле Doc type name – строку Text, в поле Filter name – строку Text Files (\*.txt). Этот параметр используется для назначения стандартных имен новым документам. Первому новому документу дается имя Text1, второму – Text2 и т.д. При сохранении только что созданных документов эти имена будут предлагаться в окне сохранения файла в качестве имен, назначенных по умолчанию. Программа со стандартным расширением файла, вводимым в поле File Extension, описана в гл. 12 в параграфе “Модификация меню File”.

В качестве базового класса для класса CMyScribeView выберите CEditView. Класс представления, порождаемый от класса CEditView, а не от CView, автоматически реализует полный набор функций текстового редактора (см. параграф “MyScribe – проектирование программы” гл. 10).

## ***Классы и программный код***

Для MDI-приложения создаваемые мастером Application Wizard классы и файлы, похожи на классы и файлы, создаваемые для SDI-приложений и описанные в параграфе “Классы и файлы программы” гл. 9. MDI-приложения, как и SDI-программы, содержат:

- класс приложения;



- класс документа;
- класс главного окна;
- класс представления.

Однако в задачах, выполняемых этими классами, есть некоторые различия. Кроме того, в программах с мультидокументным интерфейсом используется:

- класс дочернего масштабируемого окна.

*Класс приложения* программы с мультидокументным интерфейсом, подобно такому же классу SDI-приложения, управляет программой в целом и использует для инициализации программы функцию `InitInstance`. В программе `MyScribe` класс приложения называется `CMyScribeApp`, его файл заголовков – `MyScribe.h`, а файл реализации – `MyScribe.cpp`.

*Класс документа* MDI-приложения (как и для SDI-приложения) хранит данные документа и выполняет ввод/вывод файлов. Однако программа с мультидокументным интерфейсом создает отдельные экземпляры этого класса для каждого открытого документа вместо повторного использования одного и того же экземпляра. В программе `MyScribe` класс документа называется `CMyScribeDoc`, его файл заголовков – `MyScribeDoc.h`, а файл реализации – `MyScribeDoc.cpp`.

*Класс главного окна* MDI-приложения, как и соответствующий класс SDI-приложения, управляет главным окном программы. Он порождается от класса `CMDIFrameWnd` (потомка класса `CFrameWnd`), а не напрямую от класса `CFrameWnd`. В MDI-приложениях главное окно содержит *не* единственное окно представления. Вместо этого оно содержит общую рабочую область приложения, где находятся отдельные *дочерние масштабируемые окна* для каждого открытого документа. Каждое дочернее окно имеет свое окно представления. Так как главное окно программы содержит общую рабочую область приложения, а не один открытый документ, его класс не включается в шаблон документа программы. (Вспомните: шаблон документа хранит информацию о классах и ресурсах, используемых для отображения и управления документом определенного типа.) Поскольку главное окно не создается автоматически при открытии первого документа (как в SDI-приложении), то для явного создания и отображения его вызывается функция `InitInstance`.

```
// создание главного окна MDI-приложения
CMainFrame* pMainFrame = new CMainFrame;
if (!pMainFrame->LoadFrame(IDR_MAINFRAME))
    return FALSE;
m_pMainWnd = pMainFrame;
// ...

// Показ и обновление единственного проинициализированного окна
pMainFrame->ShowWindow(m_nCmdShow);
pMainFrame->UpdateWindow();
return TRUE;
```

В начале приведенного фрагмента создается экземпляр класса `CMainFrame` для главного окна. Функция `LoadFrame` класса `CFrameWnd` создает собственно главное окно, используя ресурс с идентификатором `IDR_MAINFRAME` (меню, таблица горячих клавиш, строка заголовка и значок). Дескриптор окна сохраняется в переменной `m_pMainWnd` класса `CWinApp`. Далее вызов функции `CWnd::ShowWindow` в функции `InitInstance` делает окно видимым, а вызов функции `CWnd::UpdateWindow` приводит к перерисовке рабочей области окна. В программе `MyScribe` файл заголовков класса главного окна называется `MainFrm.h`, а файл реализации – `MainFrm.cpp`.

*Класс дочернего окна* в MDI-приложениях управляет дочерним окном. Каждое дочернее окно содержит окно представления для отображения открытого документа. В программах с однодокументным интерфейсом класс дочернего окна не используется. В программе `MyScribe` класс дочернего окна `CChildFrame` порождается от класса `CMDIChildWnd` порождаемого, в свою очередь, от класса

CFrameWnd. Его файл заголовков – ChildFrm.h, файл реализации – ChildFrm.cpp. Так как класс CChildFrame используется для создания и управления дочерним окном каждого открываемого документа, функция InitInstance вместо класса главного окна (как в SDI-приложении) включает этот класс в шаблон документа программы.

```
// Регистрация шаблона документа приложения. Шаблоны
// документов служат связью между документами, окнами
// документов и окнами приложений
CMultiDocTemplate* pDocTemplate;
pDocTemplate = new CMultiDocTemplate(IDR_TextTYPE,
    RUNTIME_CLASS(CMyScribeDoc),
    RUNTIME_CLASS(CChildFrame),           // дочернее окно
    RUNTIME_CLASS(CMyScribeView));       // MDI-приложения
AddDocTemplate(pDocTemplate);
```

Шаблон принадлежит классу CMultiDocTemplate, соответствующему MDI-приложениям, а не классу CSingleDocTemplate, который использовался в предыдущей версии программы MyScribe. Шаблону назначается идентификатор IDR\_TextTYPE для ресурсов (меню, строка описания и значок), соответствующих документу. В частности, меню IDR\_TextTYPE отображается каждый раз, когда открыт один или несколько документов. Строка IDR\_TextTYPE содержит стандартное расширение файла документа и описание типа документа, открываемого программой, значок IDR\_TextTYPE отображается в каждом дочернем окне, содержащем документ.

Класс представления в MDI-приложениях используется для создания и управления окнами представлений, отображающими открытые документы, расположенными в рабочей области дочерних окон документов. Так как класс представления программы MyScribe порождается от класса CEditView, то окно представления служит текстовым редактором. В программе MyScribe класс представления назван CMyScribeView, его файл заголовков – MyScribeView.h, а файл реализации – MyScribeView.cpp.

В разрабатываемой версии программы MyScribe обрабатываются документы только одного типа – текстового. Однако при помощи MFC можно писать программы, позволяющие открывать или создавать *документы различных типов*. Например, программа для работы с электронными таблицами позволяет открывать или создавать таблицы и диаграммы. Если добавить в программу документы более чем одного типа, то команда New в меню File будет отображать диалоговое окно, позволяющее выбрать тип документа. Можно открыть существующий документ любого из добавленных типов. Дополнительная информация о работе с документами некоторых типов содержится в справочной системе.

Для организации управления документом определенного типа укажите новый класс документа для управления его данными. Затем определите новый класс представления для управления окном представления, получения вводимой информации и отображения данных документа (используйте существующие классы в качестве образца). Для документа нового типа создайте набор ресурсов: меню, таблицу горячих клавиш (для SDI-приложения), строку и значок. Строка должна иметь такую же форму, как и созданная мастером Application Wizard для документа первоначального типа. Убедитесь, что для всех ресурсов задан один и тот же идентификатор, например, IDR\_ChartTYPE. В функции InitInstance класса приложения создайте объект-шаблон (объект CSingleDocTemplate для SDI-приложения или CMultiDocTemplate для MDI-приложения), указав идентификатор новых ресурсов, новые классы документа и представления, а также класс окна (класс главного окна для SDI-приложения или дочернего для MDI-приложения). Кроме того, в функции InitInstance вызовите функцию AddDocTemplate, чтобы добавить объект шаблона в объект приложения. Этот вызов дополнит шаблоны вашими данными.

## Код программы

Мастер Application Wizard при порождении класса представления от класса CEditView (как в программе MyScribe) генерирует код для чтения и записи текстовых документов из файлов на диске (см. гл. 12) и добавляется его в функцию Serialize класса документа. Для MDI-версии программы MyScribe *не* нужно добавлять в класс документа функцию DeleteContents, а в SDI-версию программы ее нужно добавлять, так как она удаляет существующий текст из окна представления в ответ на команду New меню File. В MDI-версии команда New создает *новое* окно представления вместо использования одного и того же существующего окна. Таким образом, средства удаления текста в MDI-программе не нужны.

При создании MDI-приложения мастер Application Wizard автоматически добавляет в функцию InitInstance класса приложения вызов функции DragAcceptFiles, позволяющей открыть файл, перетаскивая его значок с помощью мыши из папки Windows или окна Explorer в окно программы. Мастер Application Wizard автоматически включает в функцию InitInstance вызовы функций EnableShellOpen и RegisterShellFileTypes (см. параграфы “Открытие файлов” и “Регистрация drw-файлов” гл. 12). Обычно эти функции позволяют открыть файл, выполняя двойной щелчок на значке файла со стандартным расширением. Однако таковым для программы MyScribe является расширение .txt, которое, как правило, уже зарегистрировано для другой программы (например, Notepad.exe). Следовательно, двойной щелчок на файле с таким расширением приведет к запуску другой программы, а не MyScribe. Можно изменить запускаемое приложение, выбрав в Windows Explorer команды Options... или Folder Options... в меню View и открыв вкладку File Types.

К разрабатываемой MDI-версии программы необходимо добавить средства, ранее включенные в SDI-версию. Это связано с тем, что для создания MDI-приложения сгенерирован полностью новый набор исходных файлов (с помощью мастера Application Wizard невозможно преобразовать существующее SDI-приложение в MDI-приложение). Вообще, мастер Application Wizard нельзя использовать для непосредственного добавления новых средств в существующую программу. Например, одно из средств, поддерживаемых мастером Application Wizard – это панель инструментов программы. Если программа создана без панели инструментов, то позже *нельзя* использовать мастер Application Wizard для прямого добавления в программу панели инструментов. Чтобы добавить ее с помощью мастера Application Wizard, необходимо сгенерировать новую программу, а затем скопировать код для панели инструментов в уже существующую программу либо скопировать все средства, добавленные в существующую программу, в генерируемые исходные файлы.

Для упрощения и уменьшения объемов приводимых листингов, мы будем использовать только те средства мастера Application Wizard, которые требуются для текущей версии программы, лишь иногда добавляя средства для написания более сложной версии. Однако при создании собственного приложения можно сэкономить много сил в ходе дальнейших модификаций программы просто за счет предварительного планирования. При создании программы с помощью мастера Application Wizard необходимо предусмотреть средства, которые могут в дальнейшем понадобиться.

## Ресурсы

---

Выполним настройку ресурсов программы MyScribe:

- модифицируем меню;
- добавим сочетания клавиш;
- снабдим программу значком.

Для начала убедитесь, что проект MyScribe открыт. Затем откройте вкладку Resource View, чтобы отобразить список ресурсов. Откройте раздел Menu в списке Resource View. Обратите внимание: поскольку MyScribe является MDI-приложением, в этом разделе появляются два идентификатора:

IDR\_MAINFRAME и IDR\_TEXTTYPE. IDR\_MAINFRAME является идентификатором меню, отображаемого, если все документы закрыты. Это меню не нужно изменять. IDR\_TEXTTYPE – идентификатор меню, отображаемого в случае, если есть открытые документы. Выполните двойной щелчок на идентификаторе IDR\_TEXTTYPE, чтобы открыть редактор меню. Откройте меню Edit и под командой Paste добавьте команду Select All, разделитель и команды Find..., Find Next и Replace... Описание этих команд приведено в табл. 17.1.

Табл. 17.1. Новые команды, добавляемые в меню Edit

Идентификатор (ID)	Надпись (Caption)	Другие свойства
ID_EDIT_SELECT_ALL	Select &All	Separator (Разделитель)
ID_EDIT_FIND	&Find...	
ID_EDIT_REPEAT	Find &Next\<F3	
ID_EDIT_REPLACE	&Replace...	

Назначьте клавишу F3 команде меню Find Next.

Для завершения настройки меню, откройте меню Window программы MyScribe, щелкните на команде New Window и нажмите клавишу Del для удаления команды New Window. Входящие в меню Window команды реализуются Windows и библиотекой MFC. Команда New Window создает дополнительное дочернее окно и окно представления, используемое для отображения документа в активном (в данный момент – дочернем) окне. Эта команда предоставляет пользователю возможности просмотра и редактирования одного документа в нескольких окнах представления. В программе MyScribe команда New Window удалена по следующей причине: окно представления, порожаемое от класса CEditView, хранит текстовый документ. Если бы программа создала несколько окон представления, отображающих один документ, было бы сложно эффективно обновлять другие представления при внесении изменений в одно из них. В обычных MDI-приложениях данные документа централизовано хранятся в объекте документа. В таких программах можно создавать несколько представлений документа (командой New Window или с помощью вешки разбивки). Каждый раз, когда пользователь изменяет одно представление, объект класса представления вызывает функциональный UpdateAllViews класса документа, чтобы обновить остальные представления.

Если необходимо изменить один или оба значка программы, раскройте раздел Icon во вкладке Resource View. Появятся идентификаторы двух значков. Значок IDR\_MAINFRAME соответствует главному окну. Он отображается в строке заголовка главного окна, в панели задач Windows, а также там, где требуется отобразить значок программы. Значок IDR\_TEXTTYPE соответствует каждому дочернему окну. Он отображается в строке заголовка дочернего окна, а также в заголовке свернутого дочернего окна в рабочей области приложения. Для настройки любого из этих значков воспользуйтесь редактором Visual Studio.

## Текст программы MyScribe

Теперь можно построить и запустить программу, а также поэкспериментировать с функциями, описанными в начале главы. В листингах 17.1—17.10 приведен текст MDI-версии программы MyScribe, созданной в этой главе.

### Листинг 17.1.

```
// MyScribe.h : главный заголовочный файл приложения MyScribe
//
#pragma once
```

```

#ifndef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"          // основные символы

// CMyScribeApp:
// Смотрите реализацию этого класса в файле MyScribe.cpp
//

class CMyScribeApp : public CWinApp
{
public:
    CMyScribeApp();

    // Переопределения
public:
    virtual BOOL InitInstance();

    // Реализация
    afx_msg void OnAppAbout();
    DECLARE_MESSAGE_MAP()
};

extern CMyScribeApp theApp;

```

---

## Листинг 17.2.

```

// MyScribe.cpp : Определяет поведение классов приложения
//

#include "stdafx.h"
#include "MyScribe.h"
#include "MainFrm.h"

#include "ChildFrm.h"
#include "MyScribeDoc.h"
#include "MyScribeView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CMyScribeApp

BEGIN_MESSAGE_MAP(CMyScribeApp, CWinApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
    // Стандартные команды работы с файлами документов
    ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
    // Стандартная команда работы с печатью
    ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()

// Конструктор класса CMyScribeApp

CMyScribeApp::CMyScribeApp()

```

```

        // TODO: добавьте сюда код конструктора.
        // Поместите весь существенный код инициализации
        // в функцию InitInstance
    }

    // Единственный объект класса CMyScribeApp

    CMyScribeApp theApp;

    // Инициализация CMyScribeApp

    BOOL CMyScribeApp::InitInstance()
    {
        CWinApp::InitInstance();

        // Инициализация библиотек OLE
        if (!AfxOleInit())
        {
            AfxMessageBox(IDP_OLE_INIT_FAILED);
            return FALSE;
        }
        AfxEnableControlContainer();

        // Стандартная инициализация.
        // Если вы не используете какие-то из предоставленных
        // возможностей и хотите уменьшить размер конечного
        // исполняемого модуля, удалите из последующего кода
        // отдельные команды инициализации элементов, которые
        // вам не нужны.
        // Измените строку - аргумент функции (ключ, под которым
        // ваши установки хранятся в реестре).
        // TODO: Измените эту строку на что-нибудь подходящее,
        // например, на имя вашей компании или организации
        SetRegistryKey(_T("Local AppWizard-Generated Applications"));
        LoadStdProfileSettings(4); // Загрузка стандартных установок
                                   // из INI-файла (включая MRU)
        // Регистрация шаблона документа приложения. Шаблоны
        // документов служат связью между документами, окнами
        // документов и окнами приложений
        CMultiDocTemplate* pDocTemplate;
        pDocTemplate = new CMultiDocTemplate(IDR_TextTYPE,
            RUNTIME_CLASS(CMyScribeDoc),
            RUNTIME_CLASS(CChildFrame), // дочернее окно
            // MDI-приложения
            RUNTIME_CLASS(CMyScribeView));
        AddDocTemplate(pDocTemplate);
        // создание главного окна MDI-приложения
        CMainFrame* pMainFrame = new CMainFrame;
        if (!pMainFrame->LoadFrame(IDR_MAINFRAME))
            return FALSE;
        m_pMainWnd = pMainFrame;
        // Разрешить открытие документа методом drag/drop
        m_pMainWnd->DragAcceptFiles();
        // Разрешить DDE Execute open
        EnableShellOpen();
    }

```

```

        RegisterShellFileTypes(TRUE);
        // Просмотр командной строки для обнаружения стандартных
        // команд оболочки, DDE, открытия файлов
        CCommandLineInfo cmdInfo;
        ParseCommandLine(cmdInfo);
        // Выполнение команд, указанных в командной строке.
        // Вернет FALSE, если приложение было запущено с
        // /RegServer, /Register, /Unregserver или /Unregister.
        if (!ProcessShellCommand(cmdInfo))
            return FALSE;
        // Показ и обновление единственного проинициализированного окна
        pMainFrame->ShowWindow(m_nCmdShow);
        pMainFrame->UpdateWindow();
        return TRUE;
    }

    // CAboutDlg диалог, используемый в App About

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

        // Данные для диалога
        enum { IDD = IDD_ABOUTBOX };

    protected:
        virtual void DoDataExchange(CDataExchange* pDX);
        // поддержка DDX/DDV

        // Реализация
    protected:
        DECLARE_MESSAGE_MAP()
    };

    CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
    {
    }

    void CAboutDlg::DoDataExchange(CDataExchange* pDX)
    {
        CDialog::DoDataExchange(pDX);
    }

    BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    END_MESSAGE_MAP()

    // Команда приложения на выполнение диалога
    void CMyScribeApp::OnAppAbout()
    {
        CAboutDlg aboutDlg;
        aboutDlg.DoModal();
    }

    // Обработчики сообщений класса CMyScribeApp

```

---

### Листинг 17.3.

```
// MyScribeDoc.h : интерфейс класса CMyscribeDoc
//

#pragma once

class CMyscribeDoc : public CDocument
{
protected: // используется только для сериализации
    CMyscribeDoc();
    DECLARE_DYNCREATE(CMyscribeDoc)

// Атрибуты
public:

// Операции
public:

// Переопределения
public:
    virtual BOOL OnNewDocument();
    virtual void Serialize(CArchive& ar);

// Реализация
public:
    virtual ~CMyscribeDoc();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Сгенерированные функции карты сообщений
protected:
    DECLARE_MESSAGE_MAP()
};
```

---

### Листинг 17.4.

```
// MyScribeDoc.cpp : реализация класса CMyscribeDoc
//

#include "stdafx.h"
#include "MyScribe.h"

#include "MyScribeDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CMyscribeDoc

IMPLEMENT_DYNCREATE(CMyscribeDoc, CDocument)
```



```

BEGIN_MESSAGE_MAP(CMyScribeDoc, CDocument)
END_MESSAGE_MAP()

// Конструктор класса CMyScribeDoc

CMyScribeDoc::CMyScribeDoc()
{
    // TODO: добавьте сюда одноразовый код конструктора
}

CMyScribeDoc::~CMyScribeDoc()
{
}

BOOL CMyScribeDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;

    // TODO: добавьте сюда код реинициализации
    // (SDI-документы будут повторно использовать этот документ)

    return TRUE;
}

// Сериализация класса CMyScribeDoc

void CMyScribeDoc::Serialize(CArchive& ar)
{
    // CEditView содержит элемент управления, производящий сериализацию
    reinterpret_cast<CEditView*>
        (m_viewList.GetHead())->SerializeRaw(ar);
}

// Диагностика класса CMyScribeDoc

#ifdef _DEBUG
void CMyScribeDoc::AssertValid() const
{
    CDocument::AssertValid();
}

void CMyScribeDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}
#endif // _DEBUG

// Команды класса CMyScribeDoc

```

---

#### Листинг 17.5.

```

// MyScribeView.h : интерфейс класса CMyScribeView
//

```

```

#pragma once

class CMyScribeView : public CEditView
{
protected: // используется только для сериализации
    CMyScribeView();
    DECLARE_DYNCREATE(CMyScribeView)

    // Атрибуты
public:
    CMyScribeDoc* GetDocument() const;

    // Операции
public:

    // Переопределения
public:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
protected:
    virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
    virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
    virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);

    // Реализация
public:
    virtual ~CMyScribeView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

    // Сгенерированные функции карты сообщений
protected:
    DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG // отладочная версия в файле MyScribeView.cpp
inline CMyScribeDoc* CMyScribeView::GetDocument() const
{ return reinterpret_cast<CMyScribeDoc*>(m_pDocument); }
#endif

```

---

## Листинг 17.6.

```

// MyScribeView.cpp : реализация класса CMyScribeView
//

#include "stdafx.h"
#include "MyScribe.h"

#include "MyScribeDoc.h"
#include "MyScribeView.h"

#ifdef _DEBUG

```

```

#define new DEBUG_NEW
#endif

// CMyScribeView

IMPLEMENT_DYNCREATE(CMyScribeView, CEditView)

BEGIN_MESSAGE_MAP(CMyScribeView, CEditView)
    // Стандартные команды печати
    ON_COMMAND(ID_FILE_PRINT, CEditView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_DIRECT, CEditView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_PREVIEW, CEditView::OnFilePrintPreview)
END_MESSAGE_MAP()

// Конструктор класса CMyScribeView

CMyScribeView::CMyScribeView()
{
    // TODO: добавьте сюда собственный код конструктора
}

CMyScribeView::~CMyScribeView()
{
}

BOOL CMyScribeView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Измените класс или стили окна здесь,
    // добавляя и изменяя поля структуры cs

    BOOL bPreCreated = CEditView::PreCreateWindow(cs);
    cs.style &=
        ~(ES_AUTOHSCROLL|WS_HSCROLL);    // Разрешить перенос слов

    return bPreCreated;
}

// Печать в CMyScribeView

BOOL CMyScribeView::OnPreparePrinting(CPrintInfo* pInfo)
{
    // Стандартная подготовка в CEditView
    return CEditView::OnPreparePrinting(pInfo);
}

void CMyScribeView::OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo)
{
    // Стандартное начало печати в CEditView
    CEditView::OnBeginPrinting(pDC, pInfo);
}

void CMyScribeView::OnEndPrinting(CDC* pDC, CPrintInfo* pInfo)
{
    // Стандартное завершение печати в CEditView
    CEditView::OnEndPrinting(pDC, pInfo);
}

```

```

// Диагностика класса CMyScribeView

#ifdef _DEBUG
void CMyScribeView::AssertValid() const
{
    CEditView::AssertValid();
}

void CMyScribeView::Dump(CDumpContext& dc) const
{
    CEditView::Dump(dc);
}

CMyScribeDoc* CMyScribeView::GetDocument() const
// не отладочная версия встроена
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CMyScribeDoc)));
    return (CMyScribeDoc*)m_pDocument;
}
#endif // _DEBUG

// Обработчики сообщений класса CMyScribeView

```

---

### Листинг 17.7.

```

// MainFrm.h : интерфейс класса CMainFrame
//

#pragma once
class CMainFrame : public CMDIFrameWnd
{
    DECLARE_DYNAMIC(CMainFrame)
public:
    CMainFrame();

// Атрибуты
public:

// Операции
public:

// Переопределения
public:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);

// Реализация
public:
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected: // встроенные элементы панели управления
    CStatusBar  m_wndStatusBar;
    CToolBar    m_wndToolBar;

```

```
// Сгенерированные функции карты сообщений
protected:
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    DECLARE_MESSAGE_MAP()
};
```

---

### Листинг 17.8.

```
// MainFrm.cpp : реализация класса CMainFrame
//

#include "stdafx.h"
#include "MyScribe.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CMainFrame

IMPLEMENT_DYNAMIC(CMainFrame, CMDIFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CMDIFrameWnd)
    ON_WM_CREATE()
END_MESSAGE_MAP()

static UINT indicators[] =
{
    ID_SEPARATOR,           // индикатор строки состояния
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
};

// Конструктор класса CMainFrame

CMainFrame::CMainFrame()
{
    // TODO: добавьте сюда код инициализации элементов
}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CMDIFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD |
        WS_VISIBLE | CBRS_TOP | CBRS_GRIPPER | CBRS_TOOLTIPS |
        CBRS_FLYBY | CBRS_SIZE_DYNAMIC) ||
        !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
```

```

    {
        TRACE0("Failed to create toolbar\n");
        return -1;        // не удалось создать панель инструментов
    }

    if (!m_wndStatusBar.Create(this) ||
        !m_wndStatusBar.SetIndicators(indicators,
            sizeof(indicators)/sizeof(UINT)))
    {
        TRACE0("Failed to create status bar\n");
        return -1;        // не удалось создать строку состояния
    }
    // TODO: Удалите три следующие строки, если не хотите, чтобы
    // панель инструментов была паркующей
    m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
    EnableDocking(CBRS_ALIGN_ANY);
    DockControlBar(&m_wndToolBar);

    return 0;
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    if( !CMDIFrameWnd::PreCreateWindow(cs) )
        return FALSE;
    // TODO: Измените класс или стили окна здесь, изменяя
    // и добавляя поля в структуру cs

    return TRUE;
}

// Диагностика CMainFrame

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
    CMDIFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
    CMDIFrameWnd::Dump(dc);
}

#endif // _DEBUG

// Обработчики сообщений класса CMainFrame

```

---

### Листинг 17.9.

```

// ChildFrm.h : интерфейс класса CChildFrame
//

#pragma once

class CChildFrame : public CMDIChildWnd

```

```

{
    DECLARE_DYNCREATE(CChildFrame)
public:
    CChildFrame();

    // Атрибуты
public:

    // Операции
public:

    // Переопределения
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);

    // Реализация
public:
    virtual ~CChildFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

    // Сгенерированные функции карты сообщений
protected:
    DECLARE_MESSAGE_MAP()
};

```

---

#### Листинг 17.10.

```

// ChildFrm.cpp : реализация класса CChildFrame
//

#include "stdafx.h"
#include "MyScribe.h"

#include "ChildFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CChildFrame

IMPLEMENT_DYNCREATE(CChildFrame, CMDIChildWnd)

BEGIN_MESSAGE_MAP(CChildFrame, CMDIChildWnd)
END_MESSAGE_MAP()

// Конструктор класса CChildFrame

CChildFrame::CChildFrame()
{
    // TODO: добавьте сюда код инициализации элементов
}

CChildFrame::~CChildFrame()
{
}

```

```

BOOL CChildFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Измените класс или стили окна здесь,
    // изменяя и добавляя поля в структуре cs
    if( !CMDIChildWnd::PreCreateWindow(cs) )
        return FALSE;

    return TRUE;
}

// Диагностика CChildFrame

#ifdef _DEBUG
void CChildFrame::AssertValid() const
{
    CMDIChildWnd::AssertValid();
}

void CChildFrame::Dump(CDumpContext& dc) const
{
    CMDIChildWnd::Dump(dc);
}

#endif // _DEBUG

// Обработчики сообщений класса CChildFrame

```

## Резюме

---

Мы рассмотрели технику проектирования программ, соответствующих модели приложения с мультидокументным интерфейсом:

- *Приложения с однодокументным интерфейсом* (или SDI-приложениями), описанные в предыдущих главах, открывают одновременно только один документ.
- *Программы с мультидокументным интерфейсом* (или MDI-приложения) открывают несколько документов одновременно, позволяют просматривать и редактировать каждый из них в отдельном дочернем масштабируемом окне. Windows и библиотека MFC предоставляют большую часть кода, необходимого для поддержки мультидокументного интерфейса.
- *Все требуемые классы* и исходные файлы MDI-приложения проще всего создать, используя мастер Application Wizard. Выберите опцию Multiple Documents в диалоговом окне мастера Application Wizard на вкладке Application Type. Как и в SDI-приложении класс главного окна MDI-приложения управляет главным окном программы. Однако в MDI-программе главное окно *содержит не единственное* окно представления, служащее для просмотра документа. Вместо этого оно содержит рабочую область приложения. Таким образом, класс главного окна не связан с определенным типом документа и не включен в шаблон документа. В дополнение к четырем классам, используемым в SDI-приложениях, MDI-приложения используют класс дочернего масштабируемого окна. Этот класс управляет дочерними окнами, создаваемыми для каждого открытого документа. Каждое дочернее окно отображается в рабочей области приложения и содержит окно представления для отображения документа. Объект класса дочернего окна создается каждый раз при открытии документа, и этот класс включен в шаблон документа программы.
- *Набор ресурсов.* MDI-приложение содержит набор ресурсов (меню, строку заголовка, значок и таблицу горячих клавиш), имеющих идентификатор IDR\_MAINFRAME и связанных с главным



окном. Это меню отображается, когда *нет* открытых документов. Значок отображается в заголовке главного окна, в панели задач Windows при выполнении программы, а также в других случаях, когда требуется его отобразить. MDI-приложение имеет еще один набор ресурсов (меню, строку заголовка и значок), соответствующих каждому дочернему окну, отображающему документ. Все эти ресурсы имеют одинаковый идентификатор, имя которого основывается на имени программы или на типовом имени документа (если оно указано при создании программы). Данное меню со значком в заголовке дочернего окна отображается, когда открыт один или несколько документов.

- Создавая программу с помощью мастера Application Wizard, следует указать *все* элементы, которые могут потребоваться, так как с помощью Application Wizard свойства существующей программы нельзя “перенастроить”.

## Глава 18

### Ввод/вывод символов

---

- Отображение текста в окне представления
- Ввод символов с клавиатуры
- Текстовый курсор
- Текст программы FontInfo

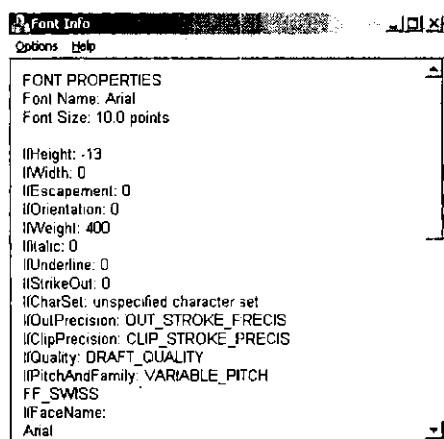
В предыдущих главах были разработаны программы, которые основывались, прежде всего, на стандартных элементах пользовательского интерфейса: строках заголовков, меню, панелях инструментов, строках состояния, окнах редактирования и надписях, используемых для отображения информации и получения данных, вводимых пользователем. С помощью этих элементов Windows и библиотека MFC предоставляют код для отображения текста и для чтения вводимых символов. В данной главе вы узнаете, как это делать самостоятельно при написании программы текстового редактора, текстового процессора или программы какого-либо другого типа. В частности:

- как *отобразить* текст непосредственно в окне представления;
- как *читать* отдельные символы, вводимые пользователем;
- как управлять мигающим *курсором*, отмечающим позицию документа в месте вставки символов.

### Отображение текста в окне представления

---

Создадим программу FontInfo, демонстрирующую отображение строк текста внутри окна представления. Основные методы, описанные в данной главе, можно использовать и для отображения текста на печатаемой странице (см. гл. 21). Программа FontInfo позволяет выбирать шрифт с помощью команды Font... меню Options, открывающей диалоговое окно Font, в котором можно выбрать имя шрифта (например, Courier), его стиль (например, **bold**), размер, эффекты (зачеркнутый или подчеркнутый), а также цвет текста. После установки всех свойств шрифта и закрытия диалогового окна Font, программа отобразит полную информацию о шрифте. Кроме того, содержащие эту информацию строки будут отображены с *использованием* описанных свойств шрифта. На рисунке показано окно программы FontInfo, отображаемое после закрытия диалогового окна Font щелчком на кнопке ОК.



Исходные файлы программы FontInfo генерируются с помощью мастера Application Wizard по методике, рассмотренной в гл. 9. В диалоговом окне New Project в поле Name введите имя программы FontInfo, а в поле Location – имя папки проекта. На вкладках диалогового окна мастера Application Wizard задайте *те же* установки, что и в программе WinHello в гл. 9, но отключите создание панели инструментов, строки состояния и поддержку печати, а на вкладке Generated Classes в качестве базового класса для класса представления выберите CScrollView.

## Отображение строк

Настройку сгенерированной программы FontInfo следует начать с написания кода для отображения текста в окне представления. Обычно этот текст сохраняется в классе документа (позже вы увидите, как это делается) и отображается функцией OnDraw класса представления. Перед добавлением кода в функцию OnDraw откройте файл FontInfoView.h и задайте значение ширины поля между текстом и верхней границей, а также между текстом и левой границей окна представления, указав следующее определение константы MARGIN.

```
const int MARGIN = 10;           // поле, отображаемое в левом
                                // верхнем углу окна представления
```

Добавьте код для отображения текста функцией OnDraw в файл FontInfoView.cpp:

```
void CFontInfoView::OnDraw(CDC* pDC)
{
    CFontInfoDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    // TODO: добавьте сюда код отображения собственных данных

    // вернуться, если не создан шрифт
    if (pDoc->m_Font.m_hObject == NULL) return;

    RECT ClipRect;
    int LineHeight;
    TEXTMETRIC TM;
    int Y = MARGIN;

    // выбор шрифта в объекте контекста устройства
    pDC->SelectObject (&pDoc->m_Font);

    // получение параметров текста
    pDC->GetTextMetrics (&TM);
    LineHeight = TM.tmHeight + TM.tmExternalLeading;

    // установка атрибутов текста
    pDC->SetTextColor (pDoc->m_Color);
    pDC->SetBkMode (TRANSPARENT);

    // получение координат недействительного участка
    pDC->GetClipBox (&ClipRect);

    pDC->TextOut (MARGIN, Y, "FONT PROPERTIES");

    // отображение параметров текста
```

```

for (int Line = 0; Line < TOTALLINES; Line++)
{
    Y += LineHeight;
    if (Y + LineHeight >= ClipRect.top && Y <=
        ClipRect.bottom)
        pDC->TextOut (MARGIN, Y, pDoc->m_LineTable
            [Line]);
}
}

```

Добавленный в функцию OnDraw фрагмент, иллюстрирует основные этапы отображения текста внутри окна представления.

1. Для случая, когда фрагмент программы, выполняющий отображение окна, *отличен* от функции OnDraw, необходимо получить объект контекста устройства для окна представления (в добавленном коде это не используется). Для завершения этих действий функция OnDraw ничего не выполняет, поэтому программе передается указатель на уже созданный объект контекста устройства. Вспомните: для отображения текста или графики необходимо иметь *объект контекста устройства*, который:

- связан с определенным устройством (например, окном на экране или принтером), сохраняющим информацию о шрифте и других атрибутах рисования;
- предоставляет функцию для рисования графических изображений на контекстно-связанном устройстве.

Если программа отображает выводимую информацию с помощью функции класса представления, отличной от функции OnDraw, создается собственный объект контекста устройства.

2. Далее следует выбрать шрифт, если не хотите использовать стандартный шрифт System. Чтобы задать шрифт, используемый для отображения текста, функция OnDraw вызывает функцию SelectObject класса CDC. Функции SelectObject передается адрес объекта шрифта, содержащий полное описание шрифта. (Как мы увидим в гл. 19, функция CDC::SelectObject может использоваться для выбора не только шрифта, но и других объектов, воздействующих на отображение графики.) Как показано ниже, при выборе нового шрифта класс документа инициализирует объект его описанием. Как только объект шрифта будет передан объекту контекста устройства, весь выводимый текст отображается с использованием шрифта, описание которого соответствует сохраняемому в объекте шрифту (в случае отсутствия выбранного шрифта используется наиболее близкий по описанию). Если шрифт не выбран, текст отображается с использованием стандартного системного шрифта System.
3. При необходимости, определите размеры текста. В функцию OnDraw передается размер шрифта посредством вызова функции GetTextMetrics класса CDC. Функция GetTextMetrics предоставляет полное описание используемого шрифта, применяемого при отображении текста. Эта информация хранится в структуре TEXTMETRIC. Как показано ниже, программа FontInfo отображает значения полей данных этой структуры для каждого выбираемого шрифта. Для определения общей высоты строки текста, функция OnDraw вычисляет сумму значения поля tmHeight структуры TEXTMETRIC (высота самого высокого символа) и поля tmExternalLeading (рекомендуемый междустрочный интервал). Результат сложения, хранящийся в переменной LineHeight, позже используется для вычисления начальной позиции по вертикали каждой строки. На следующем рисунке показаны другие поля структуры TEXTMETRIC, содержащие размеры символов.



4. При необходимости, установите желаемые атрибуты текста. Вызов функции `CDC::SetTextColor` устанавливает цвет текста, заданный при выборе шрифта, и сохраненный в переменной `m_Color` класса документа. Если вы не задали цвет текста, он будет отображаться черным. Вызов функции `CDC::SetBkMode` устанавливает режим фона текста, который относится к области, окружающей символы внутри символьных ячеек. При передаче параметра `TRANSPARENT` в функцию `SetBkMode` символы отображаются прямо поверх существующих в устройстве цветов без окраски фона. Если в функцию `SetBkMode` передать значение `OPAQUE`, то при отображении символов фон текста будет нарисован поверх существующих цветов на отображающей поверхности устройства (это стандартный режим фона). В режиме `OPAQUE` по умолчанию используется белый цвет. Вызывая функцию `CDC::SetBkColor` можно установить различные цвета фона. Функция `OnDraw` определяет режим `TRANSPARENT` так, чтобы символы отображались поверх цвета фона окна представления, поэтому в программе не нужно устанавливать цвет фона. Обратите внимание: фон окна окрашивается с использованием системы цветов "Window", устанавливаемой утилитой `Display` панели управления `Windows`.  
В табл. 18.1 приведены функции класса `CDC` для установки атрибутов текста, в табл. 18.2 – функции класса `CDC` для получения текущей установки каждого атрибута. Заметим: функции `SetMapMode` и `GetMapMode` устанавливают и обеспечивают режим отображения, воздействующий на вывод текста и графики; (см. параграф "Графические атрибуты" гл. 19). Более полную информацию о функциях, описанных в табл. 18.1 и 18.2, смотрите в справочной системе.

Табл. 18.1. Функции класса `CDC` для установки атрибутов текста

Функция	Назначение
<code>SetBkColor</code>	Определяет цвет фона текста ( <i>фоном</i> считается область вокруг символов)
<code>SetBkMode</code>	Либо разрешает, либо исключает возможность окраски фона текста
<code>SetMapMode</code>	Устанавливает текущий режим отображения, определяющий систему координат и единицы измерения для позиционирования текста и графики
<code>SetTextAlign</code>	Определяет выравнивание текста
<code>SetTextCharacterExtra</code>	Устанавливает величину межсимвольного интервала (разреженного или уплотненного)
<code>SetTextColor</code>	Определяет цвет шрифта

5. Для отображения текста функцией `OnDraw` получите размеры недействительной области окна представления (т. е. части окна, отмеченной для обновления). Эта задача решается вызовом функции `GetClipBox` класса `CDC`, которая передает размеры недействительной области окна

представления функции OnDraw. Вспомним (см. гл. 13): термин *недействительная область* относится к части окна, отмеченной для рисования или перерисовки в связи с каким-то внешним событием, например, перемещением пользователем перекрывающего окна. В этом случае на экран выводится только часть рисунка, попавшая в недействительную область, вывод вне этой области не производится (в ней остается старое содержимое).

Табл. 18.2. Функции класса CDC для определения установок атрибутов текста

Функция	Что задает
GetBkColor	Цвет фона текста
GetBkMode	Режим для фона текста
GetMapMode	Текущий режим отображения
GetTextAlign	Стиль выравнивания текста
GetTextCharacterExtra	Величина межсимвольного интервала
GetTextColor	Цвет текста

- Вызовите функцию TextOut класса CDC для отображения текста. (Если эта функция – OnDraw, отобразите только попадающий в недействительную область текст.) Функция TextOut – самая простая и широко распространенная функция отображения текста. Класс CDC предоставляет и другие функции вывода текста, поддерживающие дополнительные возможности (табл. 18.3).

Табл. 18.3. Функции класса CDC для отображения текста

Функция	Назначение
DrawText	Отображение текста в пределах заданного прямоугольника. Используется для изменения величины отступа табуляции, выравнивания текста по левому краю, центрирования или выравнивания по правому краю прямоугольника, а также для разрыва строк между словами с целью их подгонки к размерам прямоугольника
ExtTextOut	Вывод текста в пределах заданного прямоугольника. Используется для усечения текста, который не попадает в прямоугольник, заполнения прямоугольника цветом фона текста или изменения интервала между символами
GrayString	Вывод затененного текста. Обычно используется для указания недоступных опций или пунктов
TabbedTextOut	Отображает текст подобно функции TextOut, но увеличивает отступ табуляции с использованием заданного шага
TextOut	Отображает строку с заданной начальной позиции

Соответствующий фрагмент нашего примера выводит только строки текста, частично или полностью попадающие в пределы недействительной области окна представления. Windows не производит вывод любого текста, который программа пытается вывести вне этой области.

- Передаваемые в функцию TextOut *первый и второй параметры* задают координаты верхнего левого угла первого символа в отображаемой строке, т.е. координаты *точки выравнивания* символа (см. предыдущий рисунок). Заметьте: позицию точки выравнивания в пределах строки текста можно изменить, вызывая функцию CDC::SetTextAlign. Величину интервала между строками хранит переменная LineHeight.
- Передаваемый в функцию TextOut *третий параметр* является отображаемой строкой (или объектом CString, содержащим эту строку). Все остальные строки текста, кроме первой, сохранены в элементе m\_LineTable класса документа (генерация текста описана ниже).

7. Если команда Font... меню для выбора шрифта еще не вызывалась, то функция OnDraw сразу завершится, потому что текст недоступен. (Текст описывает шрифт и генерируется сразу после выбора шрифта, как описано ниже.)

## Сохранение текста и объект Font

Теперь можно дополнить класс документа FontInfo фрагментом для отображения диалогового окна Font (после выбора команды меню Font...), строками инициализации объекта шрифта в соответствии с установками, заданными в диалоговом окне Font, а также строками для генерации и сохранения текста, отображаемого в окне представления. В редакторе меню откройте меню IDR\_MAINFRAME, чтобы создать команду открытия диалогового окна Font. Удалите меню File и Edit, а затем добавьте слева от меню Help новое меню Options. В табл. 18.4 перечислены свойства пунктов этого меню.

Табл. 18.4. Свойства пунктов меню Options программы FontInfo

Идентификатор (ID)	Надпись (Caption)	Другие свойства
	&Options	Всплывающее меню
ID_OPTIONS_FONT	&Font...	
		Разделитель
ID_APP_EXIT	E&xit	

При необходимости, измените значок программы, открыв значок IDR\_MAINFRAME в графическом редакторе. Теперь сохраните изменения, сделанные в программных ресурсах, и закройте редактор ресурсов. Создайте обработчик команды Font, добавленной в меню программы FontInfo, в классе CFontInfoDoc. Перед реализацией функции OnOptionsFont необходимо открыть файл заголовков FontInfoDoc.h и определить некоторые новые переменные. В начале файла определите константу NUMLINES, содержащую число отображаемых (без учета строки заголовка) строк текста.

```
const int TALLINES = 42;
```

В начало определения класса CFontInfoDoc добавьте определения новых переменных.

```
class CFontInfoDoc : public CDocument
{
public:
    COLORREF m_Color;
    CString m_LineTable [TALLINES];
    CFont m_Font;
```

Определенная в этом фрагменте переменная m\_Color сохраняет цвет текста, выбранный в диалоговом окне Font, а m\_LineTable содержит массив объектов CString, используемый для хранения строк текста, отображенного в окне представления. Переменная m\_Font MFC-класса CFont содержит объект шрифта, используемый для установки шрифта текста.

Добавьте следующий фрагмент в сгенерированную функцию OnOptionsFont в файле TextDemoDoc.cpp.

```
void CFontInfoDoc::OnOptionsFont()
{
    // TODO: Добавьте сюда собственный код обработчика

    // отображение диалогового окна Font
    CFontDialog FontDialog;
    if (FontDialog.DoModal () != IDOK)
        return;
```

```

// установка выбранного цвета шрифта
m_Color = FontDialog.GetColor ();

// инициализация объекта шрифта
m_Font.DeleteObject ();
m_Font.CreateFontIndirect (&FontDialog.m_lf);

// получение данных о шрифте
int Num = 0;

m_LineTable [Num++] = "Font Name: " + FontDialog.GetFaceName ();

m_LineTable [Num] = "Font Size: ";
char NumBuf [18];
sprintf (NumBuf, "%d.%d points",
        FontDialog.GetSize () / 10, FontDialog.GetSize () % 10);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num++] = " ";

m_LineTable [Num] = "lfHeight: ";
sprintf (NumBuf, "%d", FontDialog.m_lf.lfHeight);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "lfWidth: ";
sprintf (NumBuf, "%d", FontDialog.m_lf.lfWidth);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "lfEscapement: ";
sprintf (NumBuf, "%d", FontDialog.m_lf.lfEscapement);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "lfOrientation: ";
sprintf (NumBuf, "%d", FontDialog.m_lf.lfOrientation);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "lfWeight: ";
sprintf (NumBuf, "%d", FontDialog.m_lf.lfWeight);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "lfItalic: ";
sprintf (NumBuf, "%d", FontDialog.m_lf.lfItalic);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "lfUnderline: ";
sprintf (NumBuf, "%d", FontDialog.m_lf.lfUnderline);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "lfStrikeOut: ";
sprintf (NumBuf, "%d", FontDialog.m_lf.lfStrikeOut);
m_LineTable [Num++] += NumBuf;

```



```

m_LineTable [Num] = "lfCharSet: ";
switch (FontDialog.m_lf.lfCharSet)
{
case ANSI_CHARSET:
    m_LineTable [Num++] += "ANSI_CHARSET";
    break;

case OEM_CHARSET:
    m_LineTable [Num++] += "OEM_CHARSET";
    break;

case SYMBOL_CHARSET:
    m_LineTable [Num++] += "SYMBOL_CHARSET";
    break;

default:
    m_LineTable [Num++] += "unspecified character set";
    break;
}

m_LineTable [Num] = "lfOutPrecision: ";
switch (FontDialog.m_lf.lfOutPrecision)
{
case OUT_CHARACTER_PRECIS:
    m_LineTable [Num++] += "OUT_CHARACTER_PRECIS";
    break;

case OUT_DEFAULT_PRECIS:
    m_LineTable [Num++] += "OUT_DEFAULT_PRECIS";
    break;

case OUT_STRING_PRECIS:
    m_LineTable [Num++] += "OUT_STRING_PRECIS";
    break;

case OUT_STROKE_PRECIS:
    m_LineTable [Num++] += "OUT_STROKE_PRECIS";
    break;

default:
    m_LineTable [Num++] += "unspecified output precision";
    break;
}

m_LineTable [Num] = "lfClipPrecision: ";
switch (FontDialog.m_lf.lfClipPrecision)
{
case CLIP_CHARACTER_PRECIS:
    m_LineTable [Num++] += "CLIP_CHARACTER_PRECIS";
    break;

case CLIP_DEFAULT_PRECIS:
    m_LineTable [Num++] += "CLIP_DEFAULT_PRECIS";
    break;
}

```

```

case CLIP_STROKE_PRECIS:
    m_LineTable [Num++] += "CLIP_STROKE_PRECIS";
    break;

default:
    m_LineTable [Num++] += "unspecified clipping precision";
    break;
}

m_LineTable [Num] = "lfQuality: ";
switch (FontDialog.m_lf.lfQuality)
{
case DEFAULT_QUALITY:
    m_LineTable [Num++] += "DEFAULT_QUALITY";
    break;

case DRAFT_QUALITY:
    m_LineTable [Num++] += "DRAFT_QUALITY";
    break;

case PROOF_QUALITY:
    m_LineTable [Num++] += "PROOF_QUALITY";
    break;

default:
    m_LineTable [Num++] += "unspecified output quality";
    break;
}

m_LineTable [Num] = "lfPitchAndFamily: ";
switch (FontDialog.m_lf.lfPitchAndFamily & 0x0003)
{
case DEFAULT_PITCH:
    m_LineTable [Num++] += "DEFAULT_PITCH";
    break;

case FIXED_PITCH:
    m_LineTable [Num++] += "FIXED_PITCH";
    break;

case VARIABLE_PITCH:
    m_LineTable [Num++] += "VARIABLE_PITCH";
    break;

default:
    m_LineTable [Num++] += "unspecified pitch";
    break;
}

switch (FontDialog.m_lf.lfPitchAndFamily & 0x00F0)
{
case FF_DECORATIVE:
    m_LineTable [Num++] += "FF_DECORATIVE; ";
    break;
}

```

```

case FF_DONTCARE:
    m_LineTable [Num++] += "FF_DONTCARE; ";
    break;

case FF_MODERN:
    m_LineTable [Num++] += "FF_MODERN; ";
    break;

case FF_ROMAN:
    m_LineTable [Num++] += "FF_ROMAN; ";
    break;

case FF_SCRIPT:
    m_LineTable [Num++] += "FF_SCRIPT; ";
    break;

case FF_SWISS:
    m_LineTable [Num++] += "FF_SWISS";
    break;

default:
    m_LineTable [Num++] += "unspecified family";
    break;
}

m_LineTable [Num++] += "lfFaceName: ";
m_LineTable [Num++] += FontDialog.m_lf.lfFaceName;
m_LineTable [Num++] += " ";

POSITION Pos = GetFirstViewPosition ();
CView *PView = GetNextView (Pos);
CClientDC ClientDC (PView);

ClientDC.SelectObject (&m_Font);
TEXTMETRIC TM;
ClientDC.GetTextMetrics (&TM);

m_LineTable [Num++] = "TEXTMETRIC fields:";

m_LineTable [Num] = "tmHeight: ";
sprintf (NumBuf, "%d", TM.tmHeight);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "tmAscent: ";
sprintf (NumBuf, "%d", TM.tmAscent);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "tmDescent: ";
sprintf (NumBuf, "%d", TM.tmDescent);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "tmInternalLeading: ";

```

```

sprintf (NumBuf, "%d", TM.tmInternalLeading);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "tmExternalLeading: ";
sprintf (NumBuf, "%d", TM.tmExternalLeading);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "tmAveCharWidth: ";
sprintf (NumBuf, "%d", TM.tmAveCharWidth);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "tmWeight: ";
sprintf (NumBuf, "%d", TM.tmWeight);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "tmOverhang: ";
sprintf (NumBuf, "%d", TM.tmOverhang);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "tmDigitizedAspectX: ";
sprintf (NumBuf, "%d", TM.tmDigitizedAspectX);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "tmDigitizedAspectY: ";
sprintf (NumBuf, "%d", TM.tmDigitizedAspectY);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "tmFirstChar: ";
sprintf (NumBuf, "%d", TM.tmFirstChar);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "tmLastChar: ";
sprintf (NumBuf, "%d", TM.tmLastChar);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "tmDefaultChar: ";
sprintf (NumBuf, "%d", TM.tmDefaultChar);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "tmBreakChar: ";
sprintf (NumBuf, "%d", TM.tmBreakChar);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "tmItalic: ";
sprintf (NumBuf, "%d", TM.tmItalic);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "tmUnderlined: ";
sprintf (NumBuf, "%d", TM.tmUnderlined);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "tmStruckOut: ";
sprintf (NumBuf, "%d", TM.tmStruckOut);
m_LineTable [Num++] += NumBuf;

```

```

m_LineTable [Num++] = "tmPitchAndFamily: ";

m_LineTable [Num] = "    Pitch Info: ";
if (TM.tmPitchAndFamily & TMPF_FIXED_PITCH)
    m_LineTable [Num] += "variable pitch ";
else
    m_LineTable [Num] += "fixed pitch ";

if (TM.tmPitchAndFamily & TMPF_VECTOR) m_LineTable [Num]
+= "vector font ";
if (TM.tmPitchAndFamily & TMPF_TRUETYPE) m_LineTable [Num]
+= "TrueType font ";
if (TM.tmPitchAndFamily & TMPF_DEVICE) m_LineTable [Num]
+= "device font ";
Num++;

m_LineTable [Num] = "    Family: ";
switch (TM.tmPitchAndFamily & 0x00F0)
{
case FF_DECORATIVE:
    m_LineTable [Num++] += "FF_DECORATIVE";
    break;

case FF_DONTCARE:
    m_LineTable [Num++] += "FF_DONTCARE";
    break;

case FF_MODERN:
    m_LineTable [Num++] += "FF_MODERN";
    break;

case FF_ROMAN:
    m_LineTable [Num++] += "FF_ROMAN";
    break;

case FF_SCRIPT:
    m_LineTable [Num++] += "FF_SCRIPT";
    break;

case FF_SWISS:
    m_LineTable [Num++] += "FF_SWISS";
    break;

default:
    m_LineTable [Num++] += "unspecified family";
    break;
}

m_LineTable [Num] = "tmCharSet: ";
switch (TM.tmCharSet)
{
case ANSI_CHARSET:

```

```

        m_LineTable [Num++] += "ANSI_CHARSET";
        break;

    case OEM_CHARSET:
        m_LineTable [Num++] += "OEM_CHARSET";
        break;

    case SYMBOL_CHARSET:
        m_LineTable [Num++] += "SYMBOL_CHARSET";
        break;

    default:
        m_LineTable [Num++] += "unspecified character set";
        break;
}

UpdateAllViews (NULL);
}

```

Функция OnOptionsFont получает управление при каждом выборе команды Font... из меню Options. С помощью фрагмента программы, добавленного в функцию OnOptionsFont, реализована следующая процедура.

1. Отображается обычное диалоговое окно Font. Диалоговое окно Font – это одно из обычных диалоговых окон, предоставляемых Windows (см. гл. 15). Диалоговое окно Font отображается при создании локального объекта MFC-класса CFontDialog и последующем вызове функции DoModal. Если отменить диалоговое окно, нажав в нем кнопку Cancel, то функция DoModal возвратит значение IDCANCEL и сразу же выполнится выход из функции OnOptionsFont. Если закрыть диалоговое окно Font щелчком на кнопке OK, то функция DoModal возвратит значение IDOK.

Диалоговое окно Font, отображаемое программой FontInfo, позволяет выбрать любые шрифты, доступные для экрана Windows. Альтернативное диалоговое окно Font используется для отображения шрифтов, доступных определенному принтеру. Шрифт в диалоговом окне Font можно настроить различными способами. Подробную информацию можно найти в справочной системе.

2. После этого функция OnOptionsFont вызывает функцию класса CFontDialog и обращается к переменным класса CFontDialog для получения информации о выбранном шрифте. Функция CFontDialog::GetColor обычно вызывается для получения значения выбранного цвета шрифта. При этом указанное значение сохраняется в элементе CFontInfoDoc::m\_Color, используемом функцией OnDraw для установки цвета шрифта. Несмотря на то, что цвет шрифта выбирается в диалоговом окне Font, все же он *не* является свойством окна. Скорее – это атрибут текста, присваиваемый объекту контекста устройства при вызове функции CDC::SetTextColor.
3. Функция OnOptionsFont вызывает функцию CFontDialog::GetFaceName для получения названия шрифта, выбранного в списке Font одноименного диалогового окна, а также функцию CFontDialog::GetSize для получения размера шрифта, выбранного в списке Size. Оба значения для отображения в окне представления записываются в массив m\_LineTable.

```

m_LineTable [Num++] = "Font Name: " + FontDialog.GetFaceName ();

m_LineTable [Num] = "Font Size: ";
char NumBuf [18];
sprintf
    (NumBuf, "%d.%d points",

```

```

        FontDialog.GetSize () / 10,
        FontDialog.GetSize () % 10);
    m_LineTable [Num++] += NumBuf;

```

4. Далее выполняется передача описания шрифта (на основании информации, введенной в диалоговое окно Font) в функцию `CFont::CreateFontIndirect` для инициализации объекта шрифта `m_Font`. Функция `OnOptionsFont` получает полное описание выбранного шрифта из переменной `CFontDialog::m_lf`, являющейся структурой `LOGFONT` (стандартная структура Windows). Структура `LOGFONT` используется для инициализации объекта шрифта `m_Font`:

```

    m_Font.DeleteObject ();
    m_Font.CreateFontIndirect (&FontDialog.m_lf);

```

Выполняемое предварительно обращение к функции `CFont::DeleteObject` удаляет существующую информацию о шрифте из объекта шрифта в том случае, если данный объект был ранее инициализирован предшествующим вызовом `OnOptionsFont`. Если объект шрифта *не* был предварительно инициализирован, то вызов `DeleteObject` бесполезен, но безопасен.

Каждый раз после инициализации объекта шрифта вызовом функции `CFont::CreateFontIndirect` (или `CFont::CreateFont`) можно *получить* текущую информацию о шрифте, записанную в соответствующем объекте шрифта. Для этого следует вызвать функцию `CGdiObject::GetObject`, копирующую эту информацию в структуру `LOGFONT`. Передача структуры `LOGFONT` в функцию `CFont::CreateFontIndirect` инициализирует (возможно, повторно) объект шрифта описанием выбранного перед этим шрифта. После вызова функции `CreateFontIndirect` информация о шрифте хранится внутри объекта шрифта, который можно выбрать в объекте контекста устройства и, как следствие, отобразить текст с использованием соответствующего шрифта.

5. Выполняется запись описания шрифта (значений полей структуры `LOGFONT`) в последовательность строк, сохраняемую в массиве `m_lineTable`, в результате чего они становятся доступными для отображения в окне представления.

```

    m_LineTable [Num++] = "LOGFONT fields:";

    m_LineTable [Num] = "lfHeight: ";
    sprintf (NumBuf, "%d", FontDialog.m_lf.lfHeight);
    m_LineTable [Num++] += NumBuf;

    m_LineTable [Num] = "lfWidth: ";
    sprintf (NumBuf, "%d", FontDialog.m_lf.lfWidth);
    m_LineTable [Num++] += NumBuf;

```

6. Функция `OnOptionsFont` создает объект контекста устройства, связанный с окном представления, выбирает в нем предварительно инициализированный объект шрифта и вызывает функцию `CDC::GetTextMetrics` для получения информации о шрифте, фактически используемом для отображения текста в окне. Функция `GetTextMetrics` копирует эту информацию в поля структуры `TEXTMETRIC` (содержимое каждого поля структуры `LOGFONT` и `TEXTMETRIC` описано в справочной системе).

```

    POSITION Pos = GetFirstViewPosition ();
    Cview *PView = GetNextView (Pos);
    CClientDC ClientDC (PView);

    // выбор шрифта из объекта контекста устройства
    ClientDC.SelectObject (&m_Font);
    TEXTMETRIC TM;
    ClientDC.GetTextMetrics (&TM);

```

Структуры LOGFONT и TEXTMETRIC похожи (имеют множество полей, хранящих одинаковую информацию), но имеют и важное теоретическое отличие. Структура LOGFONT используется для инициализации объекта шрифта и хранит описание *требуемого* шрифта, не гарантируя того, что шрифт, соответствующий описанию, действительно доступен для любого конкретного устройства вывода. Если же выбрать объект шрифта из контекста устройства, то при отображении текста будет использоваться шрифт, который более всего близок к описанию. Значения, которые функция GetTextMetrics присваивает структуре TEXTMETRIC, задают шрифт, используемый для отображения текста, который *действительно* доступен. Так как в диалоговом окне Font выбираются только фактически доступные шрифты, то структуры LOGFONT и TEXTMETRIC отображаются средствами FontInfo почти аналогично.

Мы уже знаем, что функция OnDraw выбирает объект шрифта из объекта контекста устройства, чтобы использовать его при отображении текста. Однако функция OnOptionsFont выбирает этот объект только для получения *информации* о шрифте, вызывая функцию GetTextMetrics. (Функции GetFirstViewPosition и GetNextView рассмотрены в параграфе “Реализация команд” гл. 12.)

7. Затем функция OnOptionsFont записывает содержимое каждого поля структуры TEXTMETRIC в переменную m\_LineTable.
8. В завершение выполняется вызов функции UpdateAllViews для принудительного отображения функцией OnDraw класса представления строк текста, содержащихся в массиве m\_LineTable, с использованием нового шрифта.

## Стандартные шрифты

Открыв диалоговое окно Font и использовав объект шрифта, можно выбрать *любой* из доступных шрифтов для экрана или для другого устройства, выбранного перед отображением диалогового окна. Можно выбрать *стандартный шрифт*, не отображая диалоговое окно Font и не используя объект шрифта. Стандартным является шрифт, принадлежащий небольшому набору типовых шрифтов Windows, обычно используемых для отображения информации на экране. Чтобы выбрать стандартный шрифт, вызовите функцию SelectStockObject класса CDC. Ее синтаксис выглядит так:

```
virtual CGdiObject* SelectStockObject (int nIndex);
```

Единственный параметр функции – nIndex – это индекс требуемого шрифта. Ему можно присвоить одно из значений, приведенных в табл. 18.5. Шрифты, соответствующие этим значениям, показаны на рисунке следом за таблицей.

**Табл. 18.5.** Значения, присваиваемые параметру nIndex функции SelectStockObject для выбора стандартного шрифта

Значение параметра	Стандартный шрифт
SYSTEM_FONT	Системный шрифт System с переменным питчем. Используется для отображения текста на экране, если шрифт не выбран в объекте контекста устройства. (В ранних версиях Windows использовался для отображения заголовков, меню и других текстов в окне)
SYSTEM_FIXED_FONT	Системный шрифт с фиксированным питчем Fixedsys. Наиболее разборчивый шрифт, пригодный для программ редактирования и других приложений, в которых применяется шрифт с фиксированным питчем. Используется редактором Windows Notepad
ANSI_VAR_FONT	Шрифт с переменным питчем, более мелкий, чем заданный значением SYSTEM_FONT
ANSI_FIXED_FONT	Шрифт с фиксированным питчем, более мелкий, чем заданный значением SYSTEM_FIXED_FONT



DEVICE_DEFAULT_FONT	Шрифт устройства, заданный по умолчанию (например, для окна – System, для принтера HP Laser Jet II – Courier)
OEM_FIXED_FONT	Шрифт с фиксированным питчем Terminal. Соответствует набору символов, используемых основными аппаратными средствами. (В ранних версиях Windows использовался для отображения текста в окне MS-DOS)

```

This is the SYSTEM_FONT stock font.
This is the SYSTEM_FIXED_FONT stock font.
This is the ANSI_VAR_FONT stock font.
This is the ANSI_FIXED_FONT stock font.
This is the DEVICE_DEFAULT_FONT stock font.
This is the OEM_FIXED_FONT stock font.

```

Вот пример функции OnDraw, которая перед отображением текста в окне выбирает системный шрифт с фиксированным питчем:

```

void CFontInfoView::OnDraw(CDC* pDC)
{
    CFontInfoDoc* pDoc = GetDocument();
    // TODO: здесь добавьте код отображения
    pD->SelectStockObject (SYSTEM_FIXED_FONT);
    // установка атрибутов текста ...
    // отображение текста в окне представления ...
}

```

## Средства прокрутки

Во время генерации программы FontInfo с помощью мастера AppWizard необходимо указать, что класс представления порождается от класса CScrollView, чтобы окно представления поддерживало средства прокрутки. Прокрутка нужна, если строки текста не могут полностью поместиться внутри окна представления. Особенно часто это случается при выборе крупного шрифта. В этом параграфе добавлен фрагмент программы, сообщающий MFC текущий размер документа. (Общее описание методов, применяемых для обеспечения средств прокрутки, приведено в гл. 13.) Создайте обработчик OnUpdate в классе CFontInfoView. Добавьте в него код, выделенный полужирным в следующем листинге. Виртуальная функция OnUpdate вызывается при первоначальном создании окна представления, а также всякий раз, когда функция CFontInfoDoc::OnOptionsFont обращается к CDocument::UpdateAllViews после выбора нового шрифта (о функции OnUpdate рассказывалось в гл. 13). Функция OnUpdate устанавливает полученный исходя из размеров нового шрифта размер прокручиваемого документа (т.е. полный размер основной части текста).

```

void CFontInfoView::OnUpdate(CView* pSender, LPARAM lHint,
CObject* pHint)
{
    // TODO: Добавьте сюда собственный код или вызов базового класса

    CFontInfoDoc* pDoc = GetDocument();

    if (pDoc->m_Font.m_hObject == NULL)
        SetScrollSizes (MM_TEXT, CSize (0, 0));
}

```

```

else
{
    CClientDC ClientDC (this);
    int LineWidth = 0;
    SIZE Size;
    TEXTMETRIC TM;

    ClientDC.SelectObject (&PDoc->m_Font);
    ClientDC.GetTextMetrics (&TM);

    for (int Line = 0; Line < TOTALLINES; ++Line)
    {
        Size = ClientDC.GetTextExtent
            (PDoc->m_LineTable [Line],
            PDoc->m_LineTable [Line].GetLength ());
        if (Size.cx > LineWidth)
            LineWidth = Size.cx;
    }
    Size.cx = LineWidth + MARGIN;
    Size.cy = (TM.tmHeight + TM.tmExternalLeading) *
        (TOTALLINES + 1) + MARGIN;

    SetScrollSizes (MM_TEXT, Size);
    ScrollToPosition (CPoint (0, 0));
}
CScrollView::OnUpdate (pSender, lHint, pHint);
}

```

Чтобы установить новый размер документа функция OnUpdate вызывает функцию CScrollView::SetScrollSizes. Если шрифт не выбран, то задается нулевой размер документа, а полоса прокрутки скрывается, так как отображаемый текст отсутствует. Если шрифт уже выбран, то функция OnUpdate устанавливает общую ширину и высоту текста. Ширина текста вычисляется вызовом функции CDC::GetLength для получения длины каждой строки. В LineWidth хранится ширина текста, равная максимальной длине строки.

```
Size.cx = LineWidth + MARGIN;
```

Обратите внимание, что функция GetTextMetrics возвращает *среднюю* ширину символа в поле tmAveCharWidth структуры TEXTMETRIC. В шрифте с переменным питчем для получения ширины символа или длины строки эту информацию использовать нельзя, так как ширина символов непостоянна. Действительную длину конкретной строки символов возвращает функция GetLength.

Для вычисления высоты текста создается объект контекста устройства окна представления, затем выбирается объект шрифта в объекте контекста устройства и вызывается функция GetTextMetrics для получения размеров символов. Высота текста равна сумме высоты верхнего поля и высоты одной строки, умноженной на количество строк.

```
Size.cy = (TM.tmHeight + TM.tmExternalLeading)
    * (NUMLINES + 1) + MARGIN;
```

Определив размер документа, функция OnUpdate вызывает функцию CScrollView::ScrollToPosition, чтобы прокрутить к началу текста окно представления.

```
SetScrollSizes (MM_TEXT, Size);
ScrollToPosition (CPoint (0, 0));
```

В завершение функция OnUpdate вызывает версию функции OnUpdate из класса CScrollView, чтобы она выполнила свою стандартную операцию: для перерисовки все окно объявляется недействительным.

```
CScrollView::OnUpdate (pSender, lHint, pHint);
```

## Модификация функции InitInstance

Добавьте обращение функции SetWindowText к функции InitInstance в файл FontInfo.cpp.

```
m_pMainWnd->ShowWindow(SW_SHOW);
m_pMainWnd->UpdateWindow();

m_pMainWnd->SetWindowText ("Font Info");
return TRUE;
}
```

## Ввод символов с клавиатуры

Рассмотрим, как воспринимается нажатие клавиши, когда окно программы активно. Подобно обработке сообщений мыши (см. гл. 10), при чтении кодов клавиш задействуется соответствующий обработчик сообщения в классе представления программы. При нажатии *любой* клавиши, кроме системной, посылается сообщение WM\_KEYDOWN, а при нажатии символьной клавиши посылается сообщение WM\_CHAR. Рассмотрим соответствующие обработчики.

### Сообщение WM\_KEYDOWN

Каждый раз, как только нажимается клавиша, система посылает сообщение WM\_KEYDOWN окну, в котором в данный момент находится *фокус ввода*, принадлежащий либо *активному окну*, либо наследуемому из активного окна. Активное диалоговое окно имеет выделенный заголовок или рамку. В программе MFC, сгенерированной мастером Application Wizard, при активном главном окне фокус содержится в окне представления, следовательно, при нажатии клавиши именно это окно получает сообщения WM\_KEYDOWN. В MDI-приложении фокус находится в активном окне представления. Использование обработчика сообщения WM\_KEYDOWN полезно, прежде всего, при визуализации сообщений клавиш, которые не генерируют печатные символы, например, стрелок и функциональных клавиш.

Нажатие *системной клавиши* приводит к тому, что окну с фокусом посылается сообщение WM\_SYSKEYDOWN, а не WM\_KEYDOWN. Обычно системные клавиши обрабатывает Windows, а не программа приложения. Системными клавишами являются Prt Scr, Alt и любые другие, нажатые одновременно с Alt.

Проиллюстрируем использование средств обработки сообщения WM\_KEYDOWN, добавив в программу FontInfo клавиатурный интерфейс, позволяющий прокручивать текст, используя и клавиши, и полосы прокрутки. Откройте в Developer Studio проект FontInfo, в класс CFontInfoView добавьте обработчик сообщения WM\_KEYDOWN. Добавьте в функцию OnKeyDown фрагмент программы:

```
void CFontInfoView::OnKeyDown(UINT nChar, UINT nRepCnt,
UINT nFlags)
{
    // TODO: Добавьте сюда собственный обработчик или
    // вызов базового класса

    CSize DocSize = GetTotalSize ();
```

```

RECT ClientRect;
GetClientRect (&ClientRect);
switch(nChar)
{
case VK_LEFT:
    if (ClientRect.right < DocSize.cx)
        SendMessage (WM_HSCROLL, SB_LINELEFT);
    break;

case VK_RIGHT:
    if (ClientRect.right < DocSize.cx)
        SendMessage (WM_HSCROLL, SB_LINERIGHT);
    break;

case VK_UP:
    if (ClientRect.bottom < DocSize.cy)
        SendMessage (WM_VSCROLL, SB_LINEUP);
    break;

case VK_DOWN:
    if (ClientRect.bottom < DocSize.cy)
        SendMessage (WM_VSCROLL, SB_LINEDOWN);
    break;

case VK_HOME:
    if (::GetKeyState (VK_CONTROL) & 0x8000)
    {
        if (ClientRect.bottom < DocSize.cy)
            SendMessage (WM_VSCROLL, SB_LEFT);
    }
    else
    {
        if (ClientRect.right < DocSize.cx)
            SendMessage (WM_HSCROLL, SB_TOP);
    }
    break;

case VK_END:
    if (::GetKeyState (VK_CONTROL & 0x8000))
    {
        if (ClientRect.bottom < DocSize.cy)
            SendMessage (WM_VSCROLL, SB_BOTTOM);
    }
    else
    {
        if (ClientRect.right < DocSize.cx)
            SendMessage (WM_HSCROLL, SB_RIGHT);
    }
    break;
}

```

```

case VK_PRIOR:
    if (ClientRect.bottom < DocSize.cy)
        SendMessage (WM_VSCROLL, SB_PAGEUP);
    break;

case VK_NEXT:
    if (ClientRect.bottom < DocSize.cy)
        SendMessage (WM_VSCROLL, SB_PAGEDOWN);
    break;
}

CScrollView::OnKeyDown(nChar, nRepCnt, nFlags);
}

```

В функцию OnKeyDown передается ряд параметров. Первый из них – это nChar. Он содержит значение, называемое *виртуальным кодом клавиши*, определяющим нажатую клавишу. Функция OnKeyDown использует этот код, чтобы выполнить переход к соответствующей подпрограмме. В табл. 18.6 приведен список виртуальных кодов клавиш, которые не генерируют сообщения WM\_CHAR. Клавиши, генерирующие сообщения WM\_CHAR, описаны в следующем параграфе. Они обрабатываются функцией обработки сообщения WM\_CHAR, которая по коду символа идентифицирует их.

Табл. 18.6. Виртуальные коды клавиш, не генерирующих сообщений WM\_CHAR

Значение (десятичное)	Обозначение константы	Клавиша
12	VK_CLEAR	Цифра 5 на дополнительной клавиатуре (режим Num Lock выключен)
16	VK_SHIFT	Shift
17	VK_CONTROL	Ctrl
19	VK_PAUSE	Pause
20	VK_CAPITAL	Caps Lock
33	VK_PRIOR	PgUp
34	VK_NEXT	PgDn
35	VK_END	End
36	VK_HOME	Home
37	VK_LEFT	←
38	VK_UP	↑
39	VK_RIGHT	→
40	VK_DOWN	↓
45	VK_INSERT	Ins
46	VK_DELETE	Delete
112	VK_F1	F1
113	VK_F2	F2
114	VK_F3	F3
115	VK_F4	F4
116	VK_F5	F5
117	VK_F6	F6

118	VK_F7	F7
119	VK_F8	F8
120	VK_F9	F9
121	VK_F10	F10
122	VK_F11	F11
123	VK_F12	F12
144	VK_NUMLOCK	Num Lock
145	VK_SCROLL	Scroll Lock

Нажатие клавиши Home или End приведет к тому, что функция OnKeyDown вызовет функцию Win32 API ::GetKeyState, чтобы определить, была ли одновременно с Home или End нажата клавиша Ctrl.

```
case VK_HOME:
    if (::GetKeyState (VK_CONTROL) & 0x8000)
    {
        if (ClientRect.bottom < DocSize.cy)
            SendMessage (WM_VSCROLL, SB_LEFT);
    }
    else
    {
        if (ClientRect.right < DocSize.cx)
            SendMessage (WM_HSCROLL, SB_TOP);
    }
    break;
```

Функция ::GetKeyState при вызове получает виртуальный код проверяемой клавиши (табл. 18.6) и возвращает значение, указывающее состояние соответствующей клавиши при генерации сообщения WM\_KEYDOWN. Если состояние клавиши изменилось, то старший бит значения, возвращаемого функцией ::GetKeyState, устанавливается в 1 (функция ::GetKeyState возвращает результат типа SHORT, являющийся 16-битовым значением). Проверить значение старшего бита можно так:

```
if (::GetKeyState (VK_SHIFT) & 0x8000)
    // то была нажата клавиша SHIFT
```

Если клавиша фиксируется, и она переключалась во время генерации сообщения WM\_KEYDOWN, то бит младшего разряда значения результата функции ::GetKeyState устанавливается в 1. Если клавиши Caps Lock, Num Lock и Scroll Lock *переключаются* во включенное состояние, то на клавиатуре обработчиком *зажигается* индикатор (если он имеется). Переключение клавиши Caps Lock можно проверить так:

```
if (::GetKeyState (VK_CAPITAL) & 0x0001)
    // то переключается клавиша Caps Lock
```

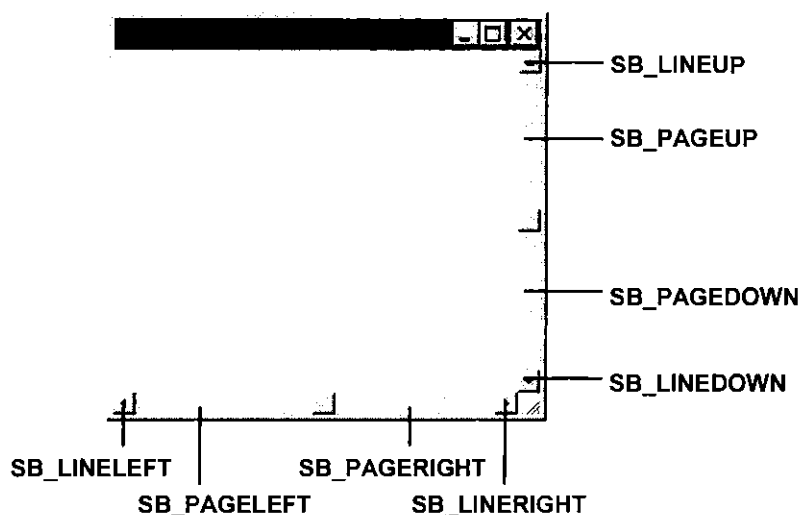
Текущее состояние клавиши можно получить, вызывая функцию Win32 API ::GetAsyncKeyState. Функция ::GetKeyState указывает, была ли данная клавиша нажата или переключена *при генерации сообщения WM\_KEYDOWN*. Она *не* возвращает текущее состояние клавиши, которое могло измениться за время обработки сообщения.

Разработанная функция OnKeyDown обрабатывает каждое нажатие клавиши, при необходимости передавая одно из тех сообщений, которые обычно передаются полосой прокрутки, когда на ней выполняется некоторое действие мышью. Сообщение передается окну представления вызовом функции CWnd::SendMessage. Класс CScrollView предоставляет обработчики для каждого из сообщений,

которые прокручивают окно и изменяют положение бегунка полосы прокрутки так же, как при щелчке на полосе прокрутки. Например, при нажатии клавиши "↓", функция OnKeyDown передает сообщение, *идентичное* передаваемому вертикальной полосой прокрутки при щелчке на кнопке под ее бегунком. Это сообщение имеет идентификатор WM\_VSCROLL, отображающий его связь с вертикальной полосой прокрутки. Специальное действие полосы прокрутки отображает посылаемый с сообщением код SB\_LINEDOWN. Когда обработчик сообщения WM\_VSCROLL, предоставляемый классом CScrollView, обрабатывает это сообщение, он прокручивает текст на одну строку вниз.

```
case VK_DOWN:
    if (ClientRect.bottom < DocSize.cy)
        SendMessage (WM_VSCROLL, SB_LINEDOWN);
    break;
```

Для каждого сообщения полосы прокрутки предусмотрен код с идентификатором, обозначение которого начинается с символов SB\_, указывающих, что произошло специальное действие прокрутки. Коды, которые передаются, если щелкнуть на полосе прокрутки, показаны на следующем рисунке.



Список клавиш, нажатие на которые обрабатывается функцией OnKeyDown приведен в табл. 18.7. Указаны действия, выполняемые после нажатия клавиши, и сообщения полосы прокрутки, передаваемые при этом функцией OnKeyDown окну представления. Термин *страница* относится к прокручиваемому расстоянию, равному  $1/10$  размера документа в направлении прокрутки, термин *строка* – к расстоянию, равному  $1/10$  размера страницы. Эти расстояния можно изменять, вызывая функцию CScrollView::SetScrollSizes (см. гл. 13).

До передачи сообщения полосы прокрутки функция OnKeyDown проверяет, видима ли соответствующая полоса прокрутки, так как при получении сообщения от скрытой полосы прокрутки код CScrollView работает неправильно. MFC скрывает горизонтальную полосу прокрутки, если окно представления имеет такую же ширину, как текст (или большую), и скрывает вертикальную полосу прокрутки, если окно представления имеет такую же высоту, как текст (или большую). Для получения размеров текста функция OnView вызывает функцию CScrollView::GetTotalSize, а для получения размера окна представления – GetClientRect.

```
CSize DocSize = GetTotalSize ();
RECT ClientRect;
GetClientRect (&ClientRect);
```

Табл. 18.7. Нажатия клавиш прокрутки, обрабатываемые функцией OnKeyDown программы FontInfo

Клавиша	Прокрутка	Сообщение, передаваемое окну представления
←	На один символ влево	WM_HSCROLL, SB_LINEUP
→	На один символ вправо	WM_HSCROLL, SB_LINEDOWN
↑	На одну строку вверх	WM_HSCROLL, SB_LINEUP
↓	На одну строку вниз	WM_HSCROLL, SB_LINEDOWN
Home	На первую позицию строки	WM_HSCROLL, SB_TOP
Ctrl+Home	На первую позицию первой строки	WM_HSCROLL, SB_TOP
End	На последнюю позицию строки	WM_HSCROLL, SB_BOTTOM
Ctrl+End	На последнюю позицию последней строки	WM_HSCROLL, SB_BOTTOM
PgUp	На одну страницу вверх	WM_HSCROLL, SB_PAGEUP
PgDn	На одну страницу вниз	WM_HSCROLL, SB_PAGEDOWN

Проверка ширины текста перед передачей сообщения горизонтальной полосы прокрутки производится так:

```
if (ClientRect.right < DocSize.cx)
    // если горизонтальная полоса прокрутки видима,
    // передается сообщение ...
```

Проверка высоты текста перед передачей сообщения вертикальной полосы прокрутки производится так:

```
if (ClientRect.bottom < DocSize.cy)
    // если вертикальная полоса прокрутки видима,
    // передается сообщение ...
```

Модификация программы FontInfo завершены. Теперь ее можно скомпилировать и выполнить. В конце главы помещены листинги исходного текста FontInfo.

## Сообщение WM\_CHAR

Сообщение WM\_CHAR передается окну с фокусом ввода при нажатии большинства клавиш. Клавиши, которые *не* передают сообщения WM\_CHAR, перечислены в табл. 18.6. Если программа позволяет отображать текст, самый простой способ чтения кода клавиши состоит в предоставлении обработчика сообщения WM\_CHAR, более удобного, чем WM\_KEYDOWN, так как он передает стандартный код ANSI печатаемого символа, а не виртуальный код клавиши, транслирующийся в код символа. Обычно при вводе текста в программу символы отображаются на экране внутри окна программы. Обработчик сообщения WM\_CHAR получает управление при каждом нажатии символьной клавиши, а параметр nChar содержит код ANSI символа.

Если нажимается клавиша, генерирующая код символа со значением *меньше* 32, то отображение на экране полученного символа невозможно. Такие коды генерируются при нажатии клавиш управления, например Ctrl+A, Enter или Tab вместо клавиш отображаемых символов. Программа либо игнорирует нажатия управляющих клавиш, либо использует их для выполнения некоторого управляющего воздействия. Например, в ответ на нажатие клавиши Backspace (код символа 8), программа может удалять предыдущий символ, а в ответ на нажатие клавиши Enter (код символа 13) – генерировать новую строку. Наиболее часто используемые клавиши управления перечислены в табл. 18.8. Если передать символьный код нажатой управляющей клавиши такой функции, как TextOut, то Windows отобразит маленький прямоугольник, указывая, что никакой печатный символ данному коду не соответствует.



Табл. 18.8. Наиболее используемые управляющие клавиши

Клавиша	Действие	Десятичное значение кода
Backspace	Возврат на одну позицию	8
Tab	Табуляция	9
Ctrl+↵	Перевод строки	10
↵	Возврат каретки	13
Esc	Выход	27

Если окно представления поддерживает средства прокрутки, т.е. порождается от класса CScrollView (см. гл. 13), то *перед* отображением текста или графики необходимо передать созданный объект контекста устройства CClientDC в функцию CView::OnPrepareDC. Функция OnPrepareDC согласует объект контекста устройства с текущей позицией прокрутки документа таким образом, чтобы выводимые символы появились в правильных позициях. Однако функцию OnPrepareDC вызывать не нужно для объекта контекста устройства, переданного функции OnDraw.

## Текстовый курсор

*Курсор* – это общедоступный ресурс Windows. Средства Visual C++ и Windows позволяют управлять курсором, отмечающим место вставки данных или внесения изменений в текст. Для управления курсором нужно определить несколько новых функций обработки сообщений. Это сообщения класса представления WM\_CREATE, WM\_KILLFOCUS и WM\_SETFOCUS. В примерах, рассматриваемых далее в этой главе, переменная m\_CaretPos типа POINT хранит текущую позицию курсора, m\_XCaret – его ширину, m\_YCaret – высоту.

Если в конструктор класса представления приложения добавить следующую строку:

```
m_CaretPos.x = m_CaretPos.y = 0;
```

то курсор первоначально появится в левом верхнем углу окна представления.

- Функция OnCreate – обработчик сообщения WM\_CREATE – вызывается после первоначального создания окна представления, но перед тем, как оно станет видимым. Добавленный в приведенном ниже примере код вычисляет и сохраняет размер отображаемого курсора.

```
int CSampleView::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CView::OnCreate(lpCreateStruct) == -1)
        return -1;

    // TODO: здесь добавьте специальный код

    CClientDC ClientDC(this);
    TEXTMETRIC TM;
    ClientDC.GetTextMetrics (&TM);
    m_XCaret = TM.tmAveCharWidth / 3;
    m_YCaret = TM.tmHeight + TM.tmExternalLeading;

    return 0;
}
```

Вместо установки некоторой произвольной ширины курсора функция OnCreate рассчитывает ее по текущей ширине рамки окна так, чтобы курсор имел подходящую ширину, независимо от

видеорежима, т. е. ширина курсора масштабируется в соответствии с размерами других стандартных элементов окна. Функция `OnCreate` делает высоту курсора равной высоте символов в стандартном системном шрифте, используемом для отображения текста. Функция `OnCreate` создает объект контекста устройства и вызывает функцию `GetTextMetrics`. На основании полей структуры `TM`, заполненной функцией `GetTextMetrics`, функция `OnCreate` вычисляет высоту курсора.

- Рассмотрим функцию `OnSetFocus` – обработчик сообщения `WM_SETFOCUS`. Функция `OnSetFocus` вызывается всякий раз, когда окно представления получает фокус ввода, в частности, при первоначальном создании окна представления и при каждом переключении на данную программу после работы в другой.

```
void CSampleView::OnSetFocus(CWnd* pOldWnd)
{
    CView::OnSetFocus(pOldWnd);

    // TODO: здесь добавьте собственный код обработчика

    CreateSolidCaret (m_XCaret, m_YCaret);
    SetCaretPos (m_CaretPos);
    ShowCaret ();
}
```

Функция `OnSetFocus` вызывает функцию `CreateSolidCaret` класса `CWnd` для создания курсора, передавая этой функции значения его ширины и высоты. Затем она вызывает функцию `CWnd::SetCaretPos`, чтобы поместить курсор в нужную позицию. Так как только что созданный курсор невидим, вызывается функция `CWnd::ShowCaret`, чтобы его отобразить. Обратите внимание: когда окно представления создано, функция `OnSetFocus` вызывается *после* функции `OnCreate` (таков порядок событий). Следовательно, можно использовать размеры курсора, установленные функцией `OnCreate`.

Курсор необходимо создавать до создания окна представления. Но его также необходимо создавать *каждый раз* при получении окном представления фокуса, поскольку курсор *удаляется* всякий раз, когда окно представления теряет фокус, функцией `::DestroyCaret` класса Win32 API, которую необходимо добавить в функцию `OnKillFocus`.

- Функция `OnKillFocus` вызывается каждый раз, когда окно представления теряет фокус ввода.

```
void CSampleView::OnKillFocus(CWnd* pNewWnd)
{
    CView::OnKillFocus(pNewWnd);

    // TODO: здесь добавьте собственный код обработчика

    ::DestroyCaret ();
}
```

Курсор необходимо удалять, поскольку это общедоступный ресурс Windows. Одновременно в рабочей области Windows может отображаться только один курсор, и он должен отображаться внутри окна с текущим фокусом ввода, чтобы показать место ввода текста (фактически, даже если приложение имеет несколько окон, ему доступен только *один* курсор).

- Функция `CWnd::HideCaret()` делает курсор невидимым.
- Функция `CWnd::ShowCaret()` делает курсор видимым.

Рассмотрим их применение на примере следующей функции `OnChar`:

```
void CSampleView::OnChar(UINT nChar, UINT nRepCnt, UINT nFlags)
```

```

{
    // TODO: здесь добавьте собственный код обработчика
    // и/или вызов стандартного обработчика

    if (nChar < 32)
    {
        ::MessageBeep (MB_OK); // генерация стандартного звука
        return;
    }

    CEchoDoc* PDoc = GetDocument();
    // сюда вставьте обработку получаемого символа

    CClientDC ClientDC (this);
    ClientDC.SetTextColor (::GetSysColor (COLOR_WINDOWTEXT));
    ClientDC.SetBkMode (TRANSPARENT);
    HideCaret ();
    ClientDC.TextOut (0, 0, PDoc->m_TextLine);
    CSize Size = ClientDC.GetTextExtent
        (PDoc->m_TextLine ,
         PDoc->m_TextLine.GetLength ());
    m_CaretPos.x = Size.cx;
    SetCaretPos (m_CaretPos);
    ShowCaret ();

    CView::OnChar(nChar, nRepCnt, nFlags);
}

```

- После сокрытия курсора и вывода измененной строки на экран добавленный код вызывает функцию `GetTextExtent` для определения новой длины строки, а затем – функцию `SetCaretPos`, чтобы переместить курсор в конец строки (в место вставки следующего символа). Функции `HideCaret ()` и `ShowCaret ()` применяются потому, что запись в окно при видимом курсоре может вызывать искажение экрана в позиции курсора. Обратите внимание: вы *не* должны скрывать курсор при рисовании функцией `OnDraw`, потому что Windows автоматически скрывает курсор до вызова этой функции и восстанавливает после возврата из нее. По той же причине не нужно скрывать курсор в функции `OnPaint`, которая рисует в окне, не являющемся окном представления.

Обратите внимание: вызовы функции `HideCaret` кумулятивные, т.е. при вызове функции `HideCaret` более одного раза подряд без вызовов функции `ShowCaret` будет необходимо столько же раз вызвать функцию `ShowCaret`, чтобы снова сделать курсор видимым.

## Текст программы FontInfo

---

Исходный код программы FontInfo приведен в листингах с 18.1 по 18.8.

### Листинг 18.1.

```

// FontInfo.h : главный файл заголовков программы FontInfo
//

#pragma once

#ifndef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCN

```

```

#endif

#include "resource.h"          // основные символы

// CFontInfoApp:
// Смотрите реализацию этого класса в файле FontInfo.cpp
//

class CFontInfoApp : public CWinApp
{
public:
    CFontInfoApp();

    // Переопределения
public:
    virtual BOOL InitInstance();

    // Реализация
    afx_msg void OnAppAbout();
    DECLARE_MESSAGE_MAP()
};

extern CFontInfoApp theApp;

```

---

## Листинг 18.2.

```

// FontInfo.cpp : Определяет поведение классов приложения
//

#include "stdafx.h"
#include "FontInfo.h"
#include "MainFrm.h"

#include "FontInfoDoc.h"
#include "FontInfoView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CFontInfoApp

BEGIN_MESSAGE_MAP(CFontInfoApp, CWinApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
    // Стандартные команды работы с файлами документов
    ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
END_MESSAGE_MAP()

// Конструктор класса CFontInfoApp

CFontInfoApp::CFontInfoApp()
{
    // TODO: добавьте сюда собственный код конструктора
    // Поместите весь существенный код инициализации
    // в функцию InitInstance
}

```

```

// Единственный объект класса CFontInfoApp
CFontInfoApp theApp;

// Инициализация CFontInfoApp
BOOL CFontInfoApp::InitInstance()
{
    CWinApp::InitInstance();

    // Инициализация библиотек OLE
    if (!AfxOleInit())
    {
        AfxMessageBox(IDP_OLE_INIT_FAILED);
        return FALSE;
    }
    AfxEnableControlContainer();
    // Стандартная инициализация.
    // Если вы не используете какие-то из возможностей и хотите
    // уменьшить размер оконечного исполняемого модуля,
    // удалите отдельные процедуры инициализации из последующего
    // кода.
    // Измените строку-аргумент функции (ключ в реестре,
    // под которым хранятся ваши установки).
    // TODO: измените эту строку на что-нибудь подходящее,
    // например, название вашей компании или организации
    SetRegistryKey(_T("Local AppWizard-Generated Applications"));
    LoadStdProfileSettings(4); // Загрузка установок из INI-файла
                               // (включая MRU)

    // Регистрация шаблонов документов приложения. Шаблоны документов
    // служат связью между документами, окнами документов и окнами
    // представлений
    CSingleDocTemplate* pDocTemplate;
    pDocTemplate = new CSingleDocTemplate(
        IDR_MAINFRAME,
        RUNTIME_CLASS(CFontInfoDoc),
        RUNTIME_CLASS(CMainFrame), // главное окно
                                   // SDI-приложения
        RUNTIME_CLASS(CFontInfoView));
    AddDocTemplate(pDocTemplate);
    // Поиск в командной строке команд управления, DDE,
    // открытия файлов
    CCommandLineInfo cmdInfo;
    ParseCommandLine(cmdInfo);
    // Выполнение команд, указанных в командной строке. Вернет
    // FALSE, если приложение запускалось с /RegServer, /Register,
    // /Unregserver или /Unregister.
    if (!ProcessShellCommand(cmdInfo))
        return FALSE;
    // Прорисовка и обновление единственного
    // проинициализированного окна
    m_pMainWnd->ShowWindow(SW_SHOW);
    m_pMainWnd->UpdateWindow();

    m_pMainWnd->SetWindowText ("Font Info");
    return TRUE;
}

```

```

// Диалог CAboutDlg, используемый в App About

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

// Данные для диалога
    enum { IDD = IDD_ABOUTBOX };

protected:
    virtual void DoDataExchange(CDataExchange* pDX);
// Поддержка DDX/DDV

// Реализация
protected:
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
END_MESSAGE_MAP()

// Команда приложения на запуск диалога
void CFontInfoApp::OnAppAbout()
{
    CAboutDlg aboutDlg;
    aboutDlg.DoModal();
}

// Обработчики сообщений класса CFontInfoApp

```

---

### Листинг 18.3.

```

// FontInfoDoc.h : интерфейс класса CFontInfoDoc
//

#pragma once

const int TOTALLINES = 42;

class CFontInfoDoc : public CDocument
{
public:
    COLORREF m_Color;
    CString m_LineTable [TOTALLINES];
    CFont m_Font;

```

```

protected: // используется только для сериализации
    CFontInfoDoc();
    DECLARE_DYNCREATE(CFontInfoDoc)

// Атрибуты
public:

// Операции
public:

// Переопределения
public:
    virtual BOOL OnNewDocument();
    virtual void Serialize(CArchive& ar);

// Реализация
public:
    virtual ~CFontInfoDoc();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Сгенерированные функции карты сообщений
protected:
    DECLARE_MESSAGE_MAP()
public:
    afx_msg void OnOptionsFont();
};

```

---

#### Листинг 18.4.

```

// FontInfoDoc.cpp : реализация класса CFontInfoDoc
//

#include "stdafx.h"
#include "FontInfo.h"

#include "FontInfoDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CFontInfoDoc

IMPLEMENT_DYNCREATE(CFontInfoDoc, CDocument)

BEGIN_MESSAGE_MAP(CFontInfoDoc, CDocument)
    ON_COMMAND(ID_OPTIONS_FONT, OnOptionsFont)
END_MESSAGE_MAP()

```

```

// Конструктор/деструктор класса CFontInfoDoc

CFontInfoDoc::CFontInfoDoc()
{
    // TODO: добавьте сюда собственный код конструктора
}

CFontInfoDoc::~CFontInfoDoc()
{
}

BOOL CFontInfoDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;

    // TODO: добавьте сюда код реинициализации
    // (SDI-документы будут повторно использовать этот документ)

    return TRUE;
}

// Сериализация класса CFontInfoDoc

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

// Диагностика класса CFontInfoDoc

#ifdef _DEBUG
void CFontInfoDoc::AssertValid() const
{
    CDocument::AssertValid();
}

void CFontInfoDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}
#endif // _DEBUG

// Команды класса CFontInfoDoc

void CFontInfoDoc::OnOptionsFont()
{
    // TODO: Добавьте сюда собственный код обработчика
    // отображение диалогового окна Font
    CFontDialog FontDialog;
}

```



```

if (FontDialog.DoModal () != IDOK)
    return;

// установка выбранного цвета шрифта
m_Color = FontDialog.GetColor ();

// инициализация объекта шрифта
m_Font.DeleteObject ();
m_Font.CreateFontIndirect (&FontDialog.m_lf);

// получение данных о шрифте
int Num = 0;

m_LineTable [Num++] = "Font Name: " + FontDialog.GetFaceName ();

m_LineTable [Num] = "Font Size: ";
char NumBuf [18];
sprintf (NumBuf, "%d.%d points",
        FontDialog.GetSize () / 10, FontDialog.GetSize () % 10);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num++] = " ";

m_LineTable [Num] = "lfHeight: ";
sprintf (NumBuf, "%d", FontDialog.m_lf.lfHeight);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "lfWidth: ";
sprintf (NumBuf, "%d", FontDialog.m_lf.lfWidth);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "lfEscapement: ";
sprintf (NumBuf, "%d", FontDialog.m_lf.lfEscapement);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "lfOrientation: ";
sprintf (NumBuf, "%d", FontDialog.m_lf.lfOrientation);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "lfWeight: ";
sprintf (NumBuf, "%d", FontDialog.m_lf.lfWeight);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "lfItalic: ";
sprintf (NumBuf, "%d", FontDialog.m_lf.lfItalic);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "lfUnderline: ";
sprintf (NumBuf, "%d", FontDialog.m_lf.lfUnderline);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "lfStrikeOut: ";
sprintf (NumBuf, "%d", FontDialog.m_lf.lfStrikeOut);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "lfCharSet: ";

```

```

switch (FontDialog.m_lf.lfCharSet)
{
case ANSI_CHARSET:
    m_LineTable [Num++] += "ANSI_CHARSET";
    break;

case OEM_CHARSET:
    m_LineTable [Num++] += "OEM_CHARSET";
    break;

case SYMBOL_CHARSET:
    m_LineTable [Num++] += "SYMBOL_CHARSET";
    break;

default:
    m_LineTable [Num++] += "unspecified character set";
    break;
}

m_LineTable [Num] = "lfOutPrecision: ";
switch (FontDialog.m_lf.lfOutPrecision)
{
case OUT_CHARACTER_PRECIS:
    m_LineTable [Num++] += "OUT_CHARACTER_PRECIS";
    break;

case OUT_DEFAULT_PRECIS:
    m_LineTable [Num++] += "OUT_DEFAULT_PRECIS";
    break;

case OUT_STRING_PRECIS:
    m_LineTable [Num++] += "OUT_STRING_PRECIS";
    break;

case OUT_STROKE_PRECIS:
    m_LineTable [Num++] += "OUT_STROKE_PRECIS";
    break;

default:
    m_LineTable [Num++] += "unspecified output precision";
    break;
}

m_LineTable [Num] = "lfClipPrecision: ";
switch (FontDialog.m_lf.lfClipPrecision)
{
case CLIP_CHARACTER_PRECIS:
    m_LineTable [Num++] += "CLIP_CHARACTER_PRECIS";
    break;

case CLIP_DEFAULT_PRECIS:
    m_LineTable [Num++] += "CLIP_DEFAULT_PRECIS";
    break;

case CLIP_STROKE_PRECIS:
    m_LineTable [Num++] += "CLIP_STROKE_PRECIS";
    break;
}

```

```

default:
    m_LineTable [Num++] += "unspecified clipping precision";
    break;
}

m_LineTable [Num] = "lfQuality: ";
switch (FontDialog.m_lf.lfQuality)
{
case DEFAULT_QUALITY:
    m_LineTable [Num++] += "DEFAULT_QUALITY";
    break;

case DRAFT_QUALITY:
    m_LineTable [Num++] += "DRAFT_QUALITY";
    break;

case PROOF_QUALITY:
    m_LineTable [Num++] += "PROOF_QUALITY";
    break;

default:
    m_LineTable [Num++] += "unspecified output quality";
    break;
}

m_LineTable [Num] = "lfPitchAndFamily: ";
switch (FontDialog.m_lf.lfPitchAndFamily & 0x0003)
{
case DEFAULT_PITCH:
    m_LineTable [Num++] += "DEFAULT_PITCH";
    break;

case FIXED_PITCH:
    m_LineTable [Num++] += "FIXED_PITCH";
    break;

case VARIABLE_PITCH:
    m_LineTable [Num++] += "VARIABLE_PITCH";
    break;

default:
    m_LineTable [Num++] += "unspecified pitch";
    break;
}

switch (FontDialog.m_lf.lfPitchAndFamily & 0x00F0)
{
case FF_DECORATIVE:
    m_LineTable [Num++] += "FF_DECORATIVE; ";
    break;

case FF_DONTCARE:
    m_LineTable [Num++] += "FF_DONTCARE; ";
    break;

case FF_MODERN:

```

```

        m_LineTable [Num++] += "FF_MODERN; ";
        break;

case FF_ROMAN:
        m_LineTable [Num++] += "FF_ROMAN; ";
        break;

case FF_SCRIPT:
        m_LineTable [Num++] += "FF_SCRIPT; ";
        break;

case FF_SWISS:
        m_LineTable [Num++] += "FF_SWISS";
        break;

default:
        m_LineTable [Num++] += "unspecified family";
        break;
}

m_LineTable [Num++] += "lfFaceName: ";
m_LineTable [Num++] += FontDialog.m_lf.lfFaceName;
m_LineTable [Num++] += " ";

POSITION Pos = GetFirstViewPosition ();
CView *PView = GetNextView (Pos);
CClientDC ClientDC (PView);

ClientDC.SelectObject (&m_Font);
TEXTMETRIC TM;
ClientDC.GetTextMetrics (&TM);

m_LineTable [Num++] = "TEXTMETRIC fields:";

m_LineTable [Num] = "tmHeight: ";
sprintf (NumBuf, "%d", TM.tmHeight);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "tmAscent: ";
sprintf (NumBuf, "%d", TM.tmAscent);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "tmDescent: ";
sprintf (NumBuf, "%d", TM.tmDescent);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "tmInternalLeading: ";
sprintf (NumBuf, "%d", TM.tmInternalLeading);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "tmExternalLeading: ";
sprintf (NumBuf, "%d", TM.tmExternalLeading);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "tmAveCharWidth: ";
sprintf (NumBuf, "%d", TM.tmAveCharWidth);

```

```

m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "tmWeight: ";
sprintf (NumBuf, "%d", TM.tmWeight);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "tmOverhang: ";
sprintf (NumBuf, "%d", TM.tmOverhang);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "tmDigitizedAspectX: ";
sprintf (NumBuf, "%d", TM.tmDigitizedAspectX);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "tmDigitizedAspectY: ";
sprintf (NumBuf, "%d", TM.tmDigitizedAspectY);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "tmFirstChar: ";
sprintf (NumBuf, "%d", TM.tmFirstChar);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "tmLastChar: ";
sprintf (NumBuf, "%d", TM.tmLastChar);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "tmDefaultChar: ";
sprintf (NumBuf, "%d", TM.tmDefaultChar);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "tmBreakChar: ";
sprintf (NumBuf, "%d", TM.tmBreakChar);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "tmItalic: ";
sprintf (NumBuf, "%d", TM.tmItalic);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "tmUnderlined: ";
sprintf (NumBuf, "%d", TM.tmUnderlined);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num] = "tmStruckOut: ";
sprintf (NumBuf, "%d", TM.tmStruckOut);
m_LineTable [Num++] += NumBuf;

m_LineTable [Num++] = "tmPitchAndFamily: ";

m_LineTable [Num] = "    Pitch Info: ";
if (TM.tmPitchAndFamily & TMPF_FIXED_PITCH)
    m_LineTable [Num] += "variable pitch ";
else
    m_LineTable [Num] += "fixed pitch ";

if (TM.tmPitchAndFamily & TMPF_VECTOR) m_LineTable [Num]
+= "vector font ";

```

```

if (TM.tmPitchAndFamily & TMPF_TRUETYPE) m_LineTable [Num]
+= "TrueType font ";
if (TM.tmPitchAndFamily & TMPF_DEVICE) m_LineTable [Num]
+= "device font ";
Num++;
\
m_LineTable [Num] = " Family: ";
switch (TM.tmPitchAndFamily & 0x00F0)
{
case FF_DECORATIVE:
    m_LineTable [Num++] += "FF_DECORATIVE";
    break;

case FF_DONTCARE:
    m_LineTable [Num++] += "FF_DONTCARE";
    break;

case FF_MODERN:
    m_LineTable [Num++] += "FF_MODERN";
    break;

case FF_ROMAN:
    m_LineTable [Num++] += "FF_ROMAN";
    break;

case FF_SCRIPT:
    m_LineTable [Num++] += "FF_SCRIPT";
    break;

case FF_SWISS:
    m_LineTable [Num++] += "FF_SWISS";
    break;

default:
    m_LineTable [Num++] += "unspecified family";
    break;
}

m_LineTable [Num] = "tmCharSet: ";
switch (TM.tmCharSet)
{
case ANSI_CHARSET:
    m_LineTable [Num++] += "ANSI_CHARSET";
    break;

case OEM_CHARSET:
    m_LineTable [Num++] += "OEM_CHARSET";
    break;

case SYMBOL_CHARSET:
    m_LineTable [Num++] += "SYMBOL_CHARSET";
    break;

default:
    m_LineTable [Num++] += "unspecified character set";
    break;
}

```

```

    }

    UpdateAllViews (NULL);
}

```

---

### Листинг 18.5.

```

// FontInfoView.h : интерфейс класса CFontInfoView
//

#pragma once

const int MARGIN = 10;          // поле, отображаемое в левом верхнем углу
                                // окна представления

class CFontInfoView : public CScrollView
{
protected: // используется только для сериализации
    CFontInfoView();
    DECLARE_DYNCREATE(CFontInfoView)

// Атрибуты
public:
    CFontInfoDoc* GetDocument() const;

// Операции
public:

// Переопределения
public:
    virtual void OnDraw(CDC* pDC);          // переопределена для
                                            // прорисовки этого вида
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
protected:
    virtual void OnInitialUpdate();         // впервые вызывается после
                                            // вызова конструктора

// Реализация
public:
    virtual ~CFontInfoView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Сгенерированные функции карты сообщений
protected:
    DECLARE_MESSAGE_MAP()
    virtual void OnUpdate(CView* /*pSender*/, LPARAM /*lHint*/,
        CObject* /*pHint*/);
public:
    afx_msg void OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags);
};

#ifdef _DEBUG // отладочная версия в файле FontInfoView.cpp

```

```

inline CFontInfoDoc* CFontInfoView::GetDocument() const
{ return reinterpret_cast<CFontInfoDoc*>(m_pDocument); }
#endif

```

---

### Листинг 18.6.

```

// FontInfoView.cpp : реализация класса CFontInfoView
//

#include "stdafx.h"
#include "FontInfo.h"

#include "FontInfoDoc.h"
#include "FontInfoView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CFontInfoView

IMPLEMENT_DYNCREATE(CFontInfoView, CScrollView)

BEGIN_MESSAGE_MAP(CFontInfoView, CScrollView)
    ON_WM_KEYDOWN()
END_MESSAGE_MAP()

// Конструктор/деструктор класса CFontInfoView

CFontInfoView::CFontInfoView()
{
    // TODO: добавьте сюда собственный код конструктора
}

CFontInfoView::~CFontInfoView()
{
}

BOOL CFontInfoView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Измените класс или стили окна здесь,
    // добавляя или изменяя поля структуры cs

    return CScrollView::PreCreateWindow(cs);
}

// Отображение класса CFontInfoView

void CFontInfoView::OnDraw(CDC* pDC)
{
    CFontInfoDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    // TODO: добавьте сюда код отображения собственных данных
}

```



```

// вернуться, если не создан шрифт
if (pDoc->m_Font.m_hObject == NULL) return;

RECT ClipRect;
int LineHeight;
TEXTMETRIC TM;
int Y = MARGIN;

// выбор шрифта в объекте контекста устройства
pDC->SelectObject (&pDoc->m_Font);

// получение параметров текста
pDC->GetTextMetrics (&TM);
LineHeight = TM.tmHeight + TM.tmExternalLeading;

// установка атрибутов текста
pDC->SetTextColor (pDoc->m_Color);
pDC->SetBkMode (TRANSPARENT);

// получение координат недействительного участка
pDC->GetClipBox (&ClipRect);

pDC->TextOut (MARGIN, Y, "FONT PROPERTIES");

// отображение параметров текста
for (int Line = 0; Line < TOTALLINES; Line++)
{
    Y += LineHeight;
    if (Y + LineHeight >= ClipRect.top && Y
        <= ClipRect.bottom)
        pDC->TextOut (MARGIN, Y, pDoc->
            m_LineTable [Line]);
}

void CFontInfoView::OnInitialUpdate()
{
    CScrollView::OnInitialUpdate();
    CSize sizeTotal;
    // TODO: вычислить общий размер изображения
    sizeTotal.cx = sizeTotal.cy = 100;
    SetScrollSizes(MM_TEXT, sizeTotal);
}

// Диагностика CFontInfoView

#ifdef _DEBUG
void CFontInfoView::AssertValid() const
{
    CScrollView::AssertValid();
}

void CFontInfoView::Dump(CDumpContext& dc) const
{
    CScrollView::Dump(dc);
}

```

```

CFontInfoDoc* CFontInfoView::GetDocument() const
// не отладочная версия встроена
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CFontInfoDoc)));
    return (CFontInfoDoc*)m_pDocument;
}
#endif // _DEBUG

// Обработчики сообщений класса CFontInfoView

void CFontInfoView::OnUpdate(CView* pSender, LPARAM lHint, CObject* pHint)
{
    // TODO: Добавьте сюда собственный код или вызов базового класса

    CFontInfoDoc* PDoc = GetDocument();

    if (PDoc->m_Font.m_hObject == NULL)
        SetScrollSizes (MM_TEXT, CSize (0, 0));
    else
    {
        CClientDC ClientDC (this);
        int LineWidth = 0;
        SIZE Size;
        TEXTMETRIC TM;

        ClientDC.SelectObject (&PDoc->m_Font);
        ClientDC.GetTextMetrics (&TM);

        for (int Line = 0; Line < TOTALLINES; ++Line)
        {
            Size = ClientDC.GetTextExtent
                (PDoc->m_LineTable [Line],
                PDoc->m_LineTable [Line].GetLength ());
            if (Size.cx > LineWidth)
                LineWidth = Size.cx;
        }
        Size.cx = LineWidth + MARGIN;
        Size.cy = (TM.tmHeight + TM.tmExternalLeading)
            * (TOTALLINES + 1) + MARGIN;

        SetScrollSizes (MM_TEXT, Size);
        ScrollToPosition (CPoint (0, 0));
    }
    CScrollView::OnUpdate (pSender, lHint, pHint);
}

void CFontInfoView::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)
{
    // TODO: Добавьте сюда собственный обработчик
    // или вызов базового класса

    CSize DocSize = GetTotalSize ();
    RECT ClientRect;
    GetClientRect (&ClientRect);
    switch(nChar)

```

```

{
case VK_LEFT:
    if (ClientRect.right < DocSize.cx)
        SendMessage (WM_HSCROLL, SB_LINELEFT);
    break;

case VK_RIGHT:
    if (ClientRect.right < DocSize.cx)
        SendMessage (WM_HSCROLL, SB_LINERIGHT);
    break;

case VK_UP:
    if (ClientRect.bottom < DocSize.cy)
        SendMessage (WM_VSCROLL, SB_LINEUP);
    break;

case VK_DOWN:
    if (ClientRect.bottom < DocSize.cy)
        SendMessage (WM_VSCROLL, SB_LINEDOWN);
    break;

case VK_HOME:
    if (::GetKeyState (VK_CONTROL) & 0x8000)
    {
        if (ClientRect.bottom < DocSize.cy)
            SendMessage (WM_VSCROLL, SB_LEFT);
    }
    else
    {
        if (ClientRect.right < DocSize.cx)
            SendMessage (WM_HSCROLL, SB_TOP);
    }
    break;

case VK_END:
    if (::GetKeyState (VK_CONTROL & 0x8000))
    {
        if (ClientRect.bottom < DocSize.cy)
            SendMessage (WM_VSCROLL, SB_BOTTOM);
    }
    else
    {
        if (ClientRect.right < DocSize.cx)
            SendMessage (WM_HSCROLL, SB_RIGHT);
    }
    break;

case VK_PRIOR:
    if (ClientRect.bottom < DocSize.cy)
        SendMessage (WM_VSCROLL, SB_PAGEUP);
    break;

```

```

        case VK_NEXT:
            if (ClientRect.bottom < DocSize.cy)
                SendMessage (WM_VSCROLL, SB_PAGEDOWN);
            break;
    }

    CScrollView::OnKeyDown(nChar, nRepCnt, nFlags);
}

```

---

### Листинг 18.7.

```

// MainFrm.h : интерфейс класса CMainFrame
//

#pragma once
class CMainFrame : public CFrameWnd
{

protected: // используется только для сериализации
    CMainFrame();
    DECLARE_DYNCREATE(CMainFrame)

// Атрибуты
public:

// Операции
public:

// Переопределения
public:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);

// Реализация
public:
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

// Сгенерированные функции карты сообщений
protected:
    DECLARE_MESSAGE_MAP()
};

```

---

### Листинг 18.8.

```

// MainFrm.cpp : реализация класса CMainFrame
//

#include "stdafx.h"
#include "FontInfo.h"

#include "MainFrm.h"

#ifdef _DEBUG

```

```

#define new DEBUG_NEW
#endif

// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
END_MESSAGE_MAP()

// Конструктор/деструктор класса CMainFrame

CMainFrame::CMainFrame()
{
    // TODO: добавьте сюда собственный код инициализации
}

CMainFrame::~CMainFrame()
{
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    if( !CFrameWnd::PreCreateWindow(cs) )
        return FALSE;
    // TODO: Измените класс или стили окна, изменяя и добавляя
    // поля структуры cs

    return TRUE;
}

// Диагностика класса CMainFrame

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
    CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
    CFrameWnd::Dump(dc);
}

#endif //_DEBUG

// Обработчики сообщений класса CMainFrame

```

## Резюме

Мы рассмотрели, как можно отобразить строки текста в окне представления, как читать символные и несимвольные клавиши, как отображать курсор и управлять им для отметки указателя вставки текста.

- *Объект контекста устройства.* При отображении текста в окне необходимо получить объект контекста устройства. Если текст отображается функцией `OnDraw`, то используется переданный в эту функцию адрес объекта контекста устройства.
- *Шрифты.* Шрифт выбирается в обычном диалоговом окне `Font`. Для этого создайте объект MFC-класса `CFontDialog`, а затем вызовите функцию `DoModal`. После возврата из функции можно получить полное описание выбранного шрифта, обращаясь к переменным и вызывая функцию класса `CFontDialog`. Для инициализации объекта шрифта, являющегося экземпляром класса `CFont`, можно использовать информацию о шрифте из объекта класса `CFontDialog`. Затем объект шрифта можно выбрать в объекте контекста устройства и, используя этот шрифт, отобразить текст. Для получения информации о размере или других характеристиках выбранного шрифта вызовите функцию `CDC::GetTextMetrics`. Если вы не хотите отображать текст, используя заданный по умолчанию шрифт `System`, то выберите альтернативный шрифт, вызвав функцию `CDC::SelectObject` (шрифт, описанный в классе `CFont`) или функцию `CDC::SelectStockObject` (стандартный шрифт). Определите любые текстовые атрибуты, стандартные значения которых хотите изменить. Вызовите функцию `CDC::SetTextCharacterExtra` для изменения межсимвольного интервала и `CDC::SetTextColor` для установки цвета текста.
- *Прорисовка текста.* При отображении текста функцией `OnDraw` необходимо вызвать функцию `CDC::GetClipBox` для получения размеров недействительной области, что позволяет увеличить эффективность работы программы, так как текст отображается только в той области окна, которая нуждается в перерисовке (в других областях он и так виден). Для завершения процедуры отображения текста используются функции класса `CDC` – `TextOut` или `DrawText`. Функция `OnDraw` отображает текст, частично или полностью попадающий в недействительную область или окно представления.
- *Обработчики клавиш.* Создав обработчик сообщения `WM_KEYDOWN`, посылаемого при нажатии какой-либо клавиши (кроме системной), можно читать данные, вводимые с клавиатуры. Обработчик передает виртуальный код клавиши, идентифицирующий ее. Обработчик сообщения `WM_KEYDOWN` предназначен для обработки несимвольных клавиш (например, `Home`, `End`, стрелки или функциональные клавиши). Наиболее удобный способ чтения символьных клавиш (т.е. клавиш, которым соответствуют печатные символы) состоит в создании обработчика сообщения `WM_CHAR`, передающего для печатного символа фактический код ANSI. В программе, сгенерированной мастером `Application Wizard`, функции обработки сообщений `WM_KEYDOWN`, `WM_CHAR` необходимо строить как элементы класса представления, так как эти сообщения передаются окну представления.
- *Курсор.* Чтобы создать и отобразить указатель, необходимо вызвать функции `CWnd::CreateSolidCaret` и `CWnd::ShowCaret` в обработчике сообщения `WM_SETFOCUS`, получающем управление при каждом помещении фокуса ввода в окно. Чтобы удалить указатель внутри обработчика сообщения `WM_KILLFOCUS`, посылаемого при потере окном фокуса ввода, необходимо вызвать функцию `DestroyCaret`. Причиной уничтожения указателя является то, что указатель может иметь только то окно, в котором находится фокус ввода. Чтобы переместить указатель на нужное место, следует вызвать функцию `CWnd::SetCaretPos`. Перед рисованием в окне с помощью функции, отличной от `OnDraw` или `OnPaint`, необходимо вызвать функцию `CWnd::HideCaret`, чтобы скрыть указатель, а после завершения рисования – вызвать функцию `CWnd::ShowCaret` для восстановления указателя. В программе, сгенерированной мастером `Application Wizard`, функции обработки сообщений `WM_SETFOCUS` или `WM_KILLFOCUS` необходимо строить как элементы класса представления, так как эти сообщения передаются окну представления.

# Глава 19

## Средства рисования

---

- Объект контекста устройства
- Инструменты рисования
- Графические атрибуты
- Рисование
- Текст программы ScratchBook

Различные подходы к созданию и манипулированию графическими изображениями рассматриваются в этой и следующей главах. Мы также рассмотрим технику вызова функций рисования в процессе выполнения программы. Эти функции предназначены для создания рисунков, состоящих из отдельных геометрических фигур, таких, как прямые линии, дуги и прямоугольники. В гл. 20 рассмотрено создание и отображение растровых изображений (называемых точечными рисунками), хранящих коды пикселей, используемых для воспроизведения образов на устройстве. Растровые изображения удобны для создания более сложных рисунков, которые нелегко разделить на отдельные геометрические фигуры.

Средства, описанные в этих главах, взаимосвязаны. Функции рисования используются для изменения узоров пикселей внутри растровых изображений, а битовые операции применяются для манипулирования изображениями, созданными с помощью функций рисования, например, для перемещения или растягивания изображения. Также рассматривается, как используются функции рисования, предоставляемые системой Windows и библиотекой MFC. Эти функции в сочетании с растровыми средствами (гл. 20) составляют полный набор инструментов создания графических образов внутри окна представления или для какого-либо другого устройства (например, принтера). (В гл. 21 описаны специальные технические приемы печати графических изображений или текстов.)

В текущей главе описаны основные действия, выполняемые при создании графических изображений, т.е. создание объекта контекста устройства, выбор средств рисования внутри объекта, установка атрибутов рисования для объекта, вызов функций-членов объекта для рисования графики. В последнем параграфе главы для иллюстрации применения описанных в этой главе средств приведен текст программы ScratchBook, выполняющей рисование различных фигур.

### Объект контекста устройства

---

Чтобы отобразить текст или графику необходим объект контекста устройства, соответствующий окну или устройству вывода данных. При рисовании этот объект сохраняет выбранные средства и установленные атрибуты и предоставляет функции-члены для рисования точек, линий, прямоугольников и других фигур. Для отображения графического объекта с помощью функции `OnDraw` класса представления используется объект контекста устройства и в функцию передается его адрес. Функция `OnDraw` вызывается при рисовании или перерисовке окна представления. Если класс представления поддерживает прокрутку (т.е. порожден от класса `CScrollView`), то переданный в него объект контекста устройства настраивается на текущую позицию прокрутки документа.

```
void CMyView::OnDraw (CDC* pDC)
{
    // отобразите графику, используя 'pDC->'
}
```

Для отображения графики не в окне представления, а в каком-то *другом* окне (например, в диалоговом), класс окна для рисования или перерисовки предоставляет обработчик сообщений WM\_PAINT, называемый OnPaint, который создает объект контекста устройства, порождаемый от MFC-класса CPaintDC. Пример функции OnPaint для диалогового окна приведен в параграфе “Управление диалоговым окном” в гл. 15.

```
void CMyDialog::OnPaint()
{
    CPaintDC PaintDC (this);

    // отобразите графику, используя 'PaintDC'...
}
```

Чтобы нарисовать или перерисовать окно представления, вызывают обработчик сообщения WM\_PAINT. Класс CView предоставляет функцию OnPaint, которая создает и подготавливает объект контекста устройства, а затем передает его в функцию OnDraw. Класс окна, который *не* порождается от CView, должен предоставлять собственную функцию OnPaint, выполняющую рисование содержимого окна. Чтобы отобразить графику в окне представления или другом окне из функции, которая *не обрабатывает* сообщения OnDraw или OnPaint, нужно создать объект контекста устройства, являющийся членом MFC-класса CClientDC. Если окно представления поддерживает прокрутку, то перед использованием объекта необходимо для настройки объекта на текущую позицию документа вызвать функцию CScrollView::OnPrepareDC. (Если необходимо отобразить графику на экране, не привязываясь к рабочей области окна, создайте объект класса CWindowDC.)

```
void CMyView::OtherFunction ()
{
    CClientDC ClientDC (this)

    // Если графика отображается в окне представления,
    // поддерживающем прокрутку:
    OnPrepareDC (&ClientDC);

    // для отображения графики используется 'ClientDC' ...
}
```

Рассмотренные в этой главе функции рисования являются членами класса CDC. Так как CDC – базовый класс по отношению к остальным классам объекта контекста устройства, то эти функции вызываются при использовании объекта контекста устройства произвольного типа. В данной главе основной акцент сделан на рисовании внутри окна (прежде всего окна представления). Однако перечисленные функции и способы не зависят от типов устройств и используются для отображения рисунков на других устройствах, например принтерах или плоттерах.

## Инструменты рисования

---

Замкнутая фигура состоит из двух отдельных элементов:

- *границы;*
- *внутренней области.*

В распоряжение пользователя, соответственно, предоставлены два инструмента:

- *перо;*
- *кисть.*



Выбор инструмента отражается на работе функций класса CDC. Задание параметров пера влияет на способ рисования линии. Они действуют как на прямые и кривые линии (например, нарисованные с использованием функции LineTo или Arc), так и на границы замкнутых фигур (например, прямоугольников и эллипсов). Задание параметров кисти воздействует на способ рисования (заливки) внутренней области замкнутых фигур. *Инструменты рисования*, определенные в этом разделе, т.е. перья и кисти, принадлежат к категории объектов, называемых *графическими* или *объектами GDI* (термин *объект* относится к структуре данных Windows, а не к объекту C++; *GDI* означает *графический интерфейс устройства* – *graphics device interface*). Существуют другие графические объекты: шрифты (см. гл. 18), растровые изображения (гл. 20), области, контуры и палитры. Хотя области, контуры и палитры также имеют отношение к рисованию, их изучение выходит за пределы этой книги. Общее описание областей, контуров и палитр приведено в справочной системе.

В момент создания объект контекста устройства содержит заданные по умолчанию перо и кисть. Перо рисует сплошную черную линию шириной в 1 пиксель независимо от текущего режима отображения (который будет рассмотрен далее). Кисть заливает внутреннюю область фигуры с замкнутым контуром непрозрачным белым цветом. Для каждого из этих инструментов в табл. 19.1 приведены функции рисования, на которые влияет выбор инструмента, и указан выбранный по умолчанию идентификатор инструмента. В таблице приведены только рассмотренные в этой главе функции рисования.

Табл. 19.1. Инструменты рисования

Инструмент рисования	Инструмент, заданный по умолчанию	Функции рисования, на которые он действует
Перо	BLACK_PEN	Arc
		Chord
		Ellipse
		LineTo
		Pie
		PolyBezier
		PolyBezierTo
		Polygon
		PolyLine
		PolyLineTo
		PolyPolygon
		PolyPolyLine
		Rectangle
		RoundRect
Кисть	WHITE_BRUSH	Chord
		Ellipse
		ExtFloodFill
		FloodFill
		Pie
		Polygon
		PolyPolygon
		Rectangle
		RoundRect

Если нужно выбрать инструмент, идентификатор передается в функцию `SelectStockObject` класса `CDC`. (Как показано в гл. 18, можно вызвать функцию `SelectStockObject` для выбора стандартного шрифта.) Чтобы изменить текущее перо или кисть, выберите стандартное перо или кисть или создайте пользовательские, а затем выберите их в объекте контекста устройства. Выбранные перо или кисть используются до следующего явного выбора других инструментов рисования. Параметр `nIndex` является кодом отдельного стандартного объекта, который передается в объект контекста устройства. Значения этого параметра для выбора стандартных перьев и кистей перечислены в табл. 19.2. Стандартное перо рисует сплошные линии шириной в один пиксель независимо от заданного режима отображения. Стандартная кисть закрашивает сплошным цветом, а не узорами (которые будут описаны далее).

```
CGdiObject* SelectStockObject (int nIndex);
```

Табл. 19.2. Значения, передаваемые в функцию `SelectStockObject` для выбора стандартных перьев и кистей

Значение	Встроенный объект
BLACK_BRUSH	Черная кисть
DKGRAY_BRUSH	Темно-серая кисть
GRAY_BRUSH	Серая кисть
LTGRAY_BRUSH	Светло-серая кисть
NULL_BRUSH	Нулевая кисть (область не закрашивается)
WHITE_BRUSH	Белая кисть (по умолчанию)
BLACK_PEN	Черное перо (по умолчанию)
NULL_PEN	Нулевое перо (не рисует линию или границу)
WHITE_PEN	Белое перо

В приведенном ниже примере выбирается серая кисть и белое перо.

```
void CMYView::OnDraw(CDC* pDC)
{
    pDC->SelectStockObject (WHITE_PEN);
    pDC->SelectStockObject (GRAY_BRUSH);

    // Вызов других графических функций и рисование графики ...
    // (линии и границы будут белыми, внутренние области
    // фигур с замкнутыми контурами - серыми)
}
```

Перо `NULL_PEN` не рисует линий. Аналогично при выборе `NULL_BRUSH` внутренняя часть фигуры не закрашивается. Этот вид кисти удобен при рисовании фигур, состоящих только из границы (прямоугольник), если необходимо оставить неизменным существующее на экране графическое изображение внутри границы. В `ScratchBook` выбирается `NULL_BRUSH` для рисования незакрашенных фигур с замкнутым контуром.

Чтобы сформировать перо или кисть, необходимо создать экземпляр класса `CPen` для пера или `CBrush` для кисти и вызвать соответствующую функцию класса `CPen` или `CBrush` для инициализации пера или кисти. Далее следует выбрать перо или кисть в объекте контекста устройства, сохраняя указатель на предыдущее перо или кисть, и вызвать функции рисования для выполнения графического вывода. Нарисовав все, что нужно, снова выберите старое перо или кисть в объекте контекста устройства. Для создания временного пера или кисти можно объявить экземпляр класса `CPen` или `CBrush` как локальный объект внутри функции, генерирующей графический вывод. Этот метод продемонстрирован в примере фрагмента программы, приведенном в конце этого раздела. При многократном использовании в программе выбранного пера или кисти объект удобнее объявить как

переменную класса представления или любого класса, управляющего окном вывода. Для инициализации пера вызовите функцию `CreatePen` класса `CPen`. Параметр `nPenStyle` описывает стиль линии, нарисованной пером.

```
BOOL CreatePen (int nPenStyle, int nWidth, COLORREF crColor);
```

Присваивание стиля `PS_NULL` создает перо, совпадающее со стандартным пером `NULL_PEN`. Стиль `PS_INSIDEFRAME` выбирает перо для рисования границы вокруг фигуры с замкнутым контуром, расположенной внутри ограничивающего прямоугольника. Стили `PS_DASH`, `PS_DOT`, `PS_DASHDOT` и `PS_DASHDOTDOT` используются, если ширина пера равна 1 пикселю. Если ширина пера превышает этот размер, то перечисленные стили генерируют сплошные линии.

Табл. 19.3. Типы перьев (Pen) и их свойства

Тип пера	Свойства
<code>PS_NULL</code>	стандартное перо
<code>PS_INSIDEFRAME</code>	рисует границу вокруг области с замкнутым контуром, расположенной внутри ограничивающего прямоугольника
<code>PS_DASH</code>	штриховая линия
<code>PS_DOT</code>	пунктирная линия
<code>PS_DASHDOT</code>	штрихпунктирная линия (точка – штрих)
<code>PS_DASHDOTDOT</code>	штрихпунктирная линия (две точки – штрих)
<code>PS_SOLID</code>	непрерывная линия

Ширину линии в *логических единицах*, используемых в текущем режиме отображения, описывает параметр `nWidth`. Если ширина пера – 0, то ширина линии – 1 пиксель, независимо от текущего режима отображения. Такая ширина генерируется и стандартным пером, и заданным по умолчанию. Параметр `crColor` задает цветовой код линии. Легче всего описать цвет, используя макрос `Win32 RGB`. Параметры `bRed`, `bGreen` и `bBlue` показывают относительную интенсивность красного, зеленого и синего цветов. Каждому параметру можно присвоить значение в диапазоне от 0 до 255. В табл. 19.4 приведены значения, которые передаются в макрос `RGB` для описания 16 чистых цветов, доступных в стандартном графическом режиме VGA.

```
ColorRef RGB (bRed, bGreen, bBlue)
```

Перу присваивается только *чистый* цвет. Чистый цвет – это цвет, генерируемый аппаратными средствами, который может быть получен установкой интенсивности RGB составляющих для отдельного пикселя в данном видеорежиме (если это невозможно, данный цвет будет симитирован узором из чистых цветов и не может использоваться для отдельного пикселя). В настоящее время технические характеристики распространенных видеокарт позволяют использовать режим `TrueColor`, при котором любое сочетание RGB-компонент даст чистый цвет. Режимы с меньшей глубиной цвета используются довольно редко (например, в случаях, когда требуется быстрая регенерация изображения). Если присвоить перу цветовой код, который не относится ни к одному из чистых цветов, то линия будет нарисована с использованием ближайшего чистого цвета. Но если перо имеет стиль `PS_INSIDEFRAME`, ширину более 1 пикселя и присвоенный цвет не является чистым, то Windows использует полутона.

В классе `CPen` предусмотрена более совершенная функция инициализации пера, называемая `ExtCreatePen`. В среде Windows NT эта функция задает способ изменения и объединения широких перьев, что позволяет создавать перья с пользовательским стилем. Однако Windows 95 не поддерживает большинство из этих средств. Заметим также, что вместо вызова функции `CPen::CreatePen`, объект пера можно инициализировать при его создании, передавая конструктору `CPen` соответствующие параметры. Информация о конструкторах `CPen` и `ExtCreatePen` содержится в справочной системе.

Табл. 19.4. Значения, передаваемые в макрос RGB для генерации стандартных чистых цветов VGA

Цвет VGA	Значения параметров bRed, bGreen и bBlue
Темно-красный (dark red)	128, 0, 0
Светло-красный (light red)	255, 0, 0
Темно-зеленый (dark green)	0, 128, 0
Светло-зеленый (light green)	0, 255, 0
Темно-синий (dark blue)	0, 0, 128
Светло-синий (light blue)	0, 0, 255
Темно-желтый (dark yellow)	128, 128, 0
Светло-желтый (light yellow)	255, 255, 0
Темно-бирюзовый (dark cyan)	0, 128, 128
Светло-бирюзовый (light cyan)	0, 255, 255
Темно-сиреневый (dark magenta)	128, 0, 128
Светло-сиреневый (light magenta)	255, 0, 255
Черный (black)	0, 0, 0
Темно-серый (dark gray)	128, 128, 128
Светло-серый (light gray)	192, 192, 192
Белый (white)	255, 255, 255

Чтобы кисть окрашивала однородным цветом внутреннюю область фигур, ее следует инициализировать, вызывая функцию `CreateSolidBrush` класса `CBrush` с параметром `crColor`, описывающим цвет заливки.

```
BOOL CreateSolidBrush (COLORREF crColor);
```

Можно задать любой цвет. Если присвоенный цвет не является чистым, то Windows генерирует полученный имитацией полутонов псевдополутоновый цвет. Кроме того, вызвав функцию `CreateHatchBrush` класса `CBrush`, можно инициализировать кисть для заливки внутренней области фигур.

```
BOOL CreateHatchBrush (int nIndex, COLORREF crColor);
```

Параметр `nIndex` задает узор. Параметр `crColor` описывает цвет линий штриховки.

Табл. 19.5. Параметры функции `CreateHatchBrush`

HS_BDIAGONAL	линии, направленные слева снизу направо вверх
HS_CROSS	прямая клетчатая заливка
HS_DIAGCROSS	косая клетчатая заливка
HS_FDIAGONAL	линии, направленные справа снизу налево вверх
HS_HORIZONTAL	горизонтальные линии
HS_VERTICAL	вертикальные линии

Для заполнения фигуры заданным узором функция `CreatePatternBrush` класса `CBrush` вызывает кисть. Параметр `pBitmap` является указателем на объект растрового изображения. Если фигура рисуется с помощью кисти, то ее внутренняя область полностью заполняется копиями растрового изображения, размещаемыми одна возле другой. Объект растрового изображения создается и инициализируется по методике, описанной в гл. 20. Задайте размер растрового изображения равным  $8 \times 8$

пикселей. Если растровое изображение монохромное, то Windows использует текущие цвета текста и фона.

```
BOOL CreatePatternBrush (CBitmap* pBitmap);
```

Кисть (как и перо) можно инициализировать при создании, передавая конструктору CBrush соответствующие параметры. Информация об этом – в справочной системе. Как только перо или кисть инициализированы, их выбирают в объекте контекста устройства с помощью функции SelectObject класса CDC. Для выбора пера вызовите функцию SelectObject, где pPen – указатель на объект-перо. Функция SelectObject возвращает указатель на *предыдущий* объект-перо, выбранный в объекте контекста устройства. Если перо ранее не выбиралось, это будет временный объект пера, заданного по умолчанию. Для выбора кисти вызывается функция SelectObject, где pBrush – указатель на объект-кисть. Функция SelectObject возвращает указатель на ранее выбранную кисть. Если она ранее не выбиралась, то это будет временный объект для заданной по умолчанию кисти.

```
CPen* SelectObject (CPen* pPen);
```

```
CBrush* SelectObject (CBrush* pBrush);
```

Вызывая функцию SelectObject для выбора пера или кисти, нужно сохранить возвращаемый указатель. После вызова графических функций для отображения выводимой информации с использованием пера или кисти (что описано далее в этой же главе) *удалите* перо или кисть из объекта контекста устройства и вызовите функцию SelectObject для выбора предыдущего объекта. Перо или кисть необходимо удалить из объекта контекста устройства, чтобы объект контекста устройства не хранил некорректный дескриптор после удаления объекта. При инициализации пера или кисти Windows добавляет дескриптор, сохраняемый внутри объекта. При выборе пера или кисти объект контекста устройства также сохраняет этот дескриптор. Когда объекты выходят за пределы области видимости или удаляются, деструктор объекта уничтожает дескриптор. Однако этот шаг не нужно выполнять, если объект контекста устройства удаляется до удаления объекта пера или кисти. Пример функции OnDraw иллюстрирует описанные действия.

```
void CMyView::OnDraw(CDC* pDC)
{
    CBrush Brush;           // объявить объект кисть;
    CPen Pen;               // объявить объект перо;
    CBrush *PtrOldBrush;    // сохранить указатель на предыдущую кисть;
    CPen *PtrOldPen;        // сохранить указатель на предыдущее перо;

    // инициализировать сплошное синее перо шириной 3 пикселя:
    Pen.CreatePen (PS_SOLID, 3, RGB (0, 0, 255));

    // инициализировать сплошную желтую кисть:
    Brush.CreateSolidBrush (RGB (255, 255, 0));

    // передать перо объекту контекста устройства:
    PtrOldPen = pDC->SelectObject (&Pen);

    // передать кисть объекту контекста устройства:
    PtrOldBrush = pDC->SelectObject (&Brush);

    // установить любые требуемые атрибуты рисования ...

    // вызвать функции рисования для создания
    // графического вывода ...
    // (линии и границы будут синими, внутренняя
    // площадь фигур будет желтой);
```

```
// удалить новые перо и кисть из объекта контекста устройства:
pDC -> SelectObject (Ptr.OldPen);
pDC -> SelectObject (Ptr.OldBrush);
```

## Графические атрибуты

Объект контекста устройства при первичном создании имеет набор стандартных атрибутов, определяющих работу функций рисования. Класс CDC содержит функции для изменения этих атрибутов, а также получения их текущих значений. В табл. 19.6 приведены наиболее подходящие для средств этой главы функции и атрибуты рисования. В первой колонке перечислены атрибуты, во второй – их стандартные значения в созданном объекте контекста устройства. Третья колонка содержит функции класса CDC для изменения набора атрибутов и функции для получения текущих установок, а в четвертой – перечислены функции рисования или функции, выполняющие только отображение текста, на которые действуют эти атрибуты (см. гл. 18, табл. 18.1, 18.2).

Рассмотрим атрибут режима отображения. Текущий режим отображения (mapping mode):

- действует на все функции рисования графики и текста;
- определяет единицы измерений и направление увеличения значений координат, используемых для отображения графики и текстов;
- воздействует на способ интерпретации координат, передаваемых в функции графического вывода и другие функции, принимающие *логические координаты*. Однако текущий режим отображения не влияет на функции, которым передают *координаты устройства*. Основное различие между логическими координатами и координатами устройства показано в параграфе “Логические и фактические координаты” гл. 13. Координаты устройства описывают размещение объекта указанием расстояния до объекта в пикселях (называемых также единицами устройства) по горизонтали и по вертикали от левого верхнего угла рабочей области окна (или левого верхнего угла доступной для печати области страницы). Горизонтальные координаты возрастают при перемещении вправо, а вертикальные – увеличиваются при перемещении вниз. Для координат устройства *начало* (т. е. точка с координатами 0, 0) всегда находится в верхнем левом углу.

В этом разделе рассматриваются *логические координаты* и *координаты устройства*. Значения координат указывают на положение точки на поверхности экрана. Однако иногда используются простые значения *измерений*, показывающие размер некоторого элемента. Значения измерения можно представить в логических единицах, т. е. в единицах, определяемых текущим режимом отображения, например, в единицах ширины пера, переданных в функцию CPen::CreatePen или в размерах графического блока, переданных в функцию CDC::BitBlt (гл. 20). Значения измерения могут также задаваться в единицах устройства (например, в пикселях) для величины прокрутки, передаваемой в CWnd::ScrollWindow, или размера окна, передаваемого сообщением WM\_SIZE. В документации на функцию или сообщение должно быть указано, какие единицы в них используются.

Логические координаты в стандартном режиме отображения (с идентификатором MM\_TEXT) приведены в пикселях (горизонтальные координаты увеличиваются при перемещении вправо, а вертикальные – вниз). В стандартном режиме отображения MM\_TEXT логические координаты совпадают с координатами устройства, если программа не меняет положение начала системы логических координат (см. ниже). Создание альтернативного режима отображения может изменить и логические единицы, и направление увеличения логических координат. Чтобы назначить альтернативный режим отображения, вызовите функцию SetMapMode класса CDC.

```
virtual int SetMapMode (int nMapMode);
```

Табл. 19.6. Основные атрибуты рисования

Атрибуты рисования	Стандартное значение	Функции, используемые для установки (получения) значения	Воздействуют на функции рисования
Направление рисования дуги	По часовой стрелке	SetArcDirection (GetArcDirection)	Arc  Chord Ellipse Pie
Цвет фона	Белый	SetBkColor (GetBkColor)	Функции режима рисования
Режим фона	OPAQUE	SetBkMode (GetBkMode)	Функции режима рисования
Первоначальная кисть	0,0 (координаты экрана)	SetBrushOrg (GetBrushOrg)	Chord  Ellipse Pie Polygon PolyPolygon Rectangle RoundRect LineTo
Текущая позиция	0,0 (логические координаты)	MoveTo (GetCurrentPosition)	PolyBezierTo PolylineTo Arc
Режим рисования	R2_COPYPEN	SetROP2 (GetROP2)	Chord Ellipse LineTo Pie PolyBezier PolyBezierTo Polygon Polyline PolylineTo PolyPolygon PolyPolyline Rectangle RoundRect
Режим отображения	MM_TEXT	SetMapMode (GetMapMode)	Все функции рисования
Режим заливки	ALTERNATE	SetPolyFillMode (GetPolyFillMode)	Polygon  PolyPolygon

В качестве параметра задается `nMapMode` – индекс, описывающий новый режим отображения. В табл. 19.7 приведены значения, которые можно присвоить параметру `nMapMode`, и соответствующие им размеры логических единиц в результирующем режиме отображения. Для режимов отображения `MM_HIENGLISH`, `MM_HIMETRIC`, `MM_LOENGLISH`, `MM_LOMETRIC` и `MM_TWIPS` горизонтальные координаты увеличиваются при перемещении вправо, вертикальные – *при передвижении вверх* (в отличие от стандартного режима отображения). Заметим, что режимы отображения `MM_ANISOMETRIC` или `MM_ISOMETRIC`, описывая размер логических единиц и направление увеличения координат, создают настраиваемый режим отображения. Вертикальные и горизонтальные координаты для режима `MM_ISOMETRIC` имеют одинаковый размер, а для режима `MM_ANISOMETRIC` могут иметь разные размеры. Для их описания в классе `CDC` предусмотрены функции `SetWindowExt` и `SetViewportExt`.

Табл. 19.7. Режимы отображения

Значение, присваиваемое параметру <code>nMapMode</code> функции <code>SetMapMode</code>	Размер логической единицы
<code>MM_ANISOTROPIC</code>	Определяется самостоятельно
<code>MM_HIENGLISH</code>	0,001 дюйма
<code>MM_HIMETRIC</code>	0,01 мм
<code>MM_ISOTROPIC</code>	Определяется самостоятельно
<code>MM_LOENGLISH</code>	0,01 дюйма
<code>MM_LOMETRIC</code>	0,1 мм
<code>MM_TEXT</code> (режим отображения по умолчанию)	1 единица устройства (пиксель)
<code>MM_TWIPS</code>	1/1440 дюйма (1/20 точки)

Начало системы логических координат (т.е. точка с логическими координатами 0, 0) для всех режимов отображения изначально расположено в верхнем левом углу рабочей области окна. Однако если класс окна представления порожден от `CScrollView` (гл. 13), MFC согласует относительное положение логического начала с прокруткой документа. Изменяется атрибут, называемый *началом представления* – *viewport origin*. Вспомните гл. 13: для порождаемого от `CScrollView` класса окна представления необходимо передать идентификатор режима отображения, используемого при вызове функции `CScrollView::SetScrollSizes` для установки размера прокрутки. Кроме того, в гл. 21 вы узнаете, что MFC обычно изменяет начало представления, чтобы печатать данную страницу в многостраничном документе.

При вызове любой из функций отображения текста (см. гл. 18), функций рисования или функций битовых операций (например, `BitBlt`, гл. 20) необходимо описывать логические координаты. Однако многие функции Windows и уведомляющие сообщения используют координаты устройства, например, функции `CWnd::GetClientRect` и `CWnd::MoveWindow` и сообщение `WM_MOUSEMOVE`. Если используется стандартный режим отображения и окно представления *не* поддерживает прокрутку, то не стоит беспокоиться о типе координат, используемых отдельной функцией или сообщением, так как логические координаты будут совпадать с координатами устройства. Но если класс представления поддерживает прокрутку или используется альтернативный режим отображения, уделите внимание типам применяемых координат. Вызвав функцию `LPtoDP` класса `CDC`, можно преобразовать логические координаты в координаты устройства, а, вызвав функцию `DPtoLP` класса `CDC`, преобразовать координаты устройства – в логические.

Использование альтернативных режимов отображения (например, `MM_HIENGLISH`, `MM_HIMETRIC`, `MM_LOENGLISH`, `MM_LOMETRIC` или `MM_TWIPS`, а также `MM_ANISOMETRIC` или `MM_ISOMETRIC` при соответствующем задании единиц) удобно тем, что размер рисуемого изображения не зависит от устройства, на котором оно отображается. И, наоборот, при использовании стандартного режима



отображения `MM_TEXT` размер зависит от разрешения устройства. Соответственно, альтернативные режимы отображения полезны для программ, которые согласуются с принципами WYSIWYG (What You See Is What You Get – что вы видите, то и получаете), особенно программ, в которых размер объекта на экране такой же, как его размер на любом принтере или на другом устройстве вывода. Однако в примерах программ, приведенных в оставшейся части книги, для простоты используется стандартный режим отображения. Информация об определении и использовании альтернативных режимов отображения – в документации на функцию `SetMapMode` класса `CDC` и другие функции отображения этого класса. Полный список этих функций приведен в справочной системе.

## Рисование

---

Создав объект контекста устройства и выбрав инструменты рисования и атрибуты, можно приступить непосредственно к рисованию. В класс `CDC` входят функции, позволяющие рисовать:

- *точки*;
- *прямые и кривые линии*;
- *фигуры с замкнутыми контурами*.

Размещение фигуры в этих функциях задается логическими координатами. В паре координат, описывающих точку и передаваемых в функцию, *горизонтальная координата передается перед вертикальной*. Горизонтальная координата часто называется координатой *x*, а вертикальная – *y*. Координаты передаются во все функции рисования как значения типа `int`. Поскольку разрядность данных типа `int` составляет 32 бита, в них может храниться значение в диапазоне от `-2147483648` до `+2147483647`. Программам в среде Windows NT можно передавать значения координат внутри полного диапазона. Однако для Windows 95 можно передавать значения координат только в диапазоне от `-32768` до `+32767`, представляющем собой диапазон 16-битовых целых значений (ограничение значения типа `int`). Все функции, рассмотренные в оставшейся части главы, являются членами класса `CDC`. Рассмотренные и кратко упоминаемые функции, можно изучить более подробно в справочной системе.

### Точка

Один пиксель можно закрасить, вызывая функцию `SetPixelV` класса `CDC`, как в следующем примере. Пример вызова функции `SetPixelV`, как и функций, приведенных в следующем параграфе, предполагает, что `pDC` является указателем на конкретный объект контекста устройства, переданный в функцию `OnDraw`, или явно определен.

```
pDC->SetPixelV (10, 15, RGB (255, 0, 0));
```

Горизонтальные и вертикальные координаты пикселя в логических единицах описываются первыми двумя параметрами, а третий параметр задает его цвет. Если заданный цвет не является чистым, доступным в текущем видеорежиме, то функция `SetPixelV` будет использовать ближайший чистый цвет. Очевидно, что для единственного пикселя невозможно использовать псевдополутон. Если нужно прочитать текущий цветовой код точки, то вместо вызова функции `SetPixelV` вызовите более медленную функцию `CDC::SetPixel`. Функция `SetPixel` устанавливает пикселу указанный цвет и возвращает прежний. Кроме того, можно прочитать текущее значение цвета пикселя, вызывая функцию `CDC::GetPixel`. MFC поддерживает альтернативные версии большинства функций рисования, описывающих координаты путем передачи соответствующей структуры вместо передачи отдельных координат. Например, в функцию `SetPixelV` можно передать одну структуру `POINT` вместо первых двух параметров. Синтаксис альтернативных вариантов функций смотрите в справочной системе.

## Программа FractalView

Программа FractalView, описанная ниже, иллюстрирует использование функции SetPixelV, а также другие способы создания графических изображений. Она рисует рекурсивный узор (множество Мандельброта), который полностью заполняет рабочую область в окне представления. При изменении размера окна или удалении перекрывающего окна, программа удаляет недействительную область окна представления и перерисовывает узор.

Обратитесь к мастеру Application Wizard, чтобы сгенерировать исходные файлы для программы FractalView. На вкладках диалогового окна Application Wizard отключите создание строки состояния и панели инструментов в создаваемой однодокументной программе. После создания исходных файлов начните настройку программы. Переопределите виртуальную функцию OnIdle класса приложения.

```
BOOL CFractalViewApp::OnIdle(LONG lCount)
{
    // TODO: Добавьте сюда собственный код обработчика
    // или вызов базового класса

    CFractalViewView *PView =
        (CFractalViewView *) ((CFrameWnd *)m_pMainWnd)
        ->GetActiveView ();
    PView->DrawCol ();
    return TRUE;
}
```

Во время простоя программы, т.е. когда она не занята обработкой сообщения, функция OnIdle вызывается периодически (из главного цикла обработки сообщения MFC).

1. Вначале OnIdle вызывает вариант функции OnIdle из базового класса для выполнения стандартных действий программы.
2. Далее добавленный к функции OnIdle код вызывает функцию CFrameWnd::GetActiveView, чтобы получить указатель на объект представления.
3. Переменная m\_pMainWnd класса приложения сохраняет адрес объекта главного окна и используется для вызова функции GetActiveView.
4. Вызывается функция-член DrawColor класса представления (определена ниже), чтобы нарисовать единственный столбец рекурсивного узора.
5. Наконец, обработчик OnIdle возвращает значение TRUE, так что MFC будет продолжать периодически ее вызывать. Если функция OnIdle возвращает значение FALSE, то MFC не будет ее вызывать до получения программой следующего сообщения.

На обычной аппаратуре рисование рекурсивного узора занимает много времени. Но программа FractalView не рисует полную фигуру из функции OnDraw. Блокирование обработки сообщений программой FractalView на протяжении всего времени рисования узора запретило бы пользователю выбирать команды меню или других программ. (Функция обработки сообщений, например, OnDraw, должна возвращать управление перед обработкой следующего сообщения.) Программа FractalView рисует только *единственный столбец пикселей* внутри узора каждый раз при вызове функции OnIdle. После рисования этого столбца функция OnIdle возвращает управление, позволяя обработать любое ожидаемое сообщение. Затем функция OnIdle получает управление и рисует следующий столбец. Этот процесс повторяется до завершения рисования узора. При создании рекурсивного узора программа FractalView обрабатывает следующий столбец узора и проверяет ожидаемое сообщение для его обработки, если оно имеется. В гл. 22 описана альтернативная версия программы FractalView, использующая более простой и эффективный метод выполнения длительных графических операций:

запуск отдельного потока управления, рисующего весь графический узор, пока основной поток продолжает обрабатывать сообщения.

Далее следует добавить к классу представления обработчик сообщений WM\_SIZE. Откройте вкладку Class View и откройте окно Properties для класса CFractalViewView. Создайте обработчик сообщения WM\_SIZE. В начало определения класса добавьте выделенные полужирным шрифтом определения. Для сохранения номера следующего столбца пикселей внутри рекурсивного узора используется переменная m\_Col. Переменные m\_ColMax и m\_RowMax сохраняют номера последней строки и столбца. Эти значения зависят от размера окна представления. Далее вы узнаете, как они устанавливаются. Переменные m\_CR, m\_CDI и m\_CDR используются при генерации рекурсивных узоров, а функция DrawColor – для рисования каждой колонки изображения.

```
// FractalViewView.h : интерфейс класса CFractalViewView
//

#pragma once

class CFractalViewView : public CView
{
private:
    int m_Col;
    int m_ColMax;
    float m_CR;
    float m_CDI;
    float m_CDR;
    int m_RowMax;

public:
    void DrawCol ();
};
```

Добавьте в начале файла в FractalViewView.cpp определения, выделенные полужирным шрифтом. Константы и массив ColorTable используются при генерации графического узора. Так как еще не все изменения внесены, оставьте файл FractalViewView.cpp открытым.

```
// FractalViewView.cpp : реализация класса CFractalViewView
//

#include "stdafx.h"
#include "FractalView.h"

#include "FractalViewDoc.h"
#include "FractalViewView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// определение набора констант Мандельброта
#define CIMAX 1.2
#define CIMIN -1.2
#define CRMAX 1.0
#define CRMIN -2.0
#define NMAX 128
```

```

// цвета, используемые для узора Мандельброта
DWORD ColorTable [6] =
{
    0x0000ff, // красный
    0x00ff00, // зеленый
    0xff0000, // синий
    0x00ffff, // желтый
    0xffff00, // бирюзовый
    0xff00ff, // фиолетовый
};

```

Выполните инициализацию конструктора класса CFractalViewView и определите функции DrawCol в конце файла.

```

CFractalViewView::CFractalViewView()
{
    // TODO: добавьте сюда код конструктора
    m_Col = 0;
}

void CFractalViewView::DrawCol ()
{
    CClientDC ClientDC (this);
    float CI;
    int ColorVal;
    float I;
    float ISqr;
    float R;
    float RSqr;
    int Row;

    if (m_Col >= m_ColMax || GetParentFrame ()->IsIconic ())
        return;

    CI = (float) CIMAX;
    for (Row = 0; Row < m_RowMax; Row++)
    {
        R = (float)0.0;
        I = (float)0.0;
        RSqr = (float)0.0;
        ISqr = (float)0.0;
        ColorVal = 0;
        while (ColorVal < NMAX && RSqr + ISqr < 4)
        {
            ++ColorVal;
            RSqr = R * R;
            ISqr = I * I;
            I *= R;
            I += I + CI;
            R = RSqr - ISqr + m_CR;
        }
        ClientDC.SetPixelV (m_Col, Row, ColorTable
            [ColorVal % 6]);
    }
}

```

```

        CI -= m_DCI;
    }
    m_Col++;
    m_CR += m_DCR;
}

```

Функция DrawCol при каждом вызове рисует следующий столбец пикселей внутри рекурсивного узора, продвигаясь слева направо. Функция DrawCol создает объект контекста устройства класса CClientDC, а потом использует уравнение Мандельброта для вычисления цветового кода каждого пикселя в столбце. Чтобы закрасить пиксель, она вызывает функцию SetPixelV класса CDC. (Описание метода вычисления цветового кода каждого пикселя во множестве Мандельброта в этой книге не приводится. Можете обратиться к книгам и статьям, посвященным рекурсивным методам, например, *Fractal Programming in C* Роджера Стивенса (Roger T. Stevens), издательство M&T Books.) Обратите внимание: функция DrawCol возвращает управление сразу, без рисования столбца, если главное окно минимизировано, т.е. если CWnd::IsIconic возвращает значение TRUE. Адрес объекта главного окна, используемый для вызова этой функции, получен вызовом функции CWnd::GetParentFrame. Рекурсивный узор полностью заполняет окно представления, следовательно, значения переменных, используемых для его создания, зависят от размера окна. Они устанавливаются в ответ на сообщение WM\_SIZE, передаваемое при первичном создании окна и при каждом изменении его размеров. Добавьте в обработчик OnSize этого сообщения следующий фрагмент программы в файл FractalViewView.cpp.

```

void CFractalViewView::OnSize(UINT nType, int cx, int cy)
{
    CView::OnSize(nType, cx, cy);

    // TODO: Добавьте сюда собственный код обработчика
    if (cx <= 1 || cy <= 1)    // проверка деления на ноль
        return;
    m_ColMax = cx;
    m_RowMax = cy;

    m_DCR = (float)((CRMAX - CRMIN) / (m_ColMax - 1));
    m_DCI = (float)((CIMAX - CIMIN) / (m_RowMax - 1));
}

```

Переданные в OnSize параметры cx и cy содержат текущие размеры окна представления в единицах устройства (пикселях). После вызова функции OnSize окно представления очищается, а для перерисовки окна вызывается функция OnDraw. Последняя функция вызывается также при перерисовке окна представления по какой-либо причине (например, из-за удаления перекрывающего окна). Функция OnDraw не рисует непосредственно узор. Она переустанавливает столбец в 0 таким образом, что функция DrawCol начинает перерисовывать рекурсивный узор (один столбец за один раз), начиная с первого столбца. Законченное определение функции OnDraw имеет такой вид:

```

void CFractalViewView::OnDraw(CDC* /*pDC*/)
{
    CFractalViewDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    // TODO: добавьте сюда собственный код прорисовки

    m_Col = 0;
    m_CR = (float)CRMIN;
}

```

Наконец, добавьте обычный вызов функции SetWindowText в функцию InitInstance в файл FractalView.cpp.

```
// Прорисовка и обновление единственного
// проинициализированного окна
m_pMainWnd->ShowWindow(SW_SHOW);
m_pMainWnd->UpdateWindow();

m_pMainWnd->SetWindowText ("Fractal Drawing");

return TRUE;
```

Программа FractalView готова.

## **Текст программы FractalView**

В следующих листингах (19.1—19.8) приведены исходные тексты программы FractalView.

---

### **Листинг 19.1**

```
// FractalView.h : главный заголовочный файл приложения FractalView
//

#pragma once

#ifndef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"          // основные символы

// CFractalViewApp:
// Смотрите реализацию этого класса в файле FractalView.cpp
//

class CFractalViewApp : public CWinApp
{
public:
    CFractalViewApp();

// Переопределения
public:
    virtual BOOL InitInstance();

// Реализация
    afx_msg void OnAppAbout();
    DECLARE_MESSAGE_MAP()
    virtual BOOL OnIdle(LONG lCount);
};

extern CFractalViewApp theApp;
```

---

### **Листинг 19.2**

```
// FractalView.cpp : Определяет поведение классов приложения
//

#include "stdafx.h"
```

```

#include "FractalView.h"
#include "MainFrm.h"

#include "FractalViewDoc.h"
#include "FractalViewView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CFractalViewApp

BEGIN_MESSAGE_MAP(CFractalViewApp, CWinApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
    // Стандартные команды работы с файлами документов
    ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
END_MESSAGE_MAP()

// Конструктор CFractalViewApp
CFractalViewApp::CFractalViewApp()
{
    // TODO: добавьте сюда код конструктора.
    // Поместите весь существенный код инициализации в
    // функцию InitInstance
}

// Единственный объект класса CFractalViewApp
CFractalViewApp theApp;

// Инициализация CFractalViewApp
BOOL CFractalViewApp::InitInstance()
{
    CWinApp::InitInstance();

    // Инициализация библиотек OLE
    if (!AfxOleInit())
    {
        AfxMessageBox(IDP_OLE_INIT_FAILED);
        return FALSE;
    }
    AfxEnableControlContainer();
    // Стандартная инициализация.
    // Если вы не используете какие-то из возможностей и хотите
    // уменьшить размер оконечного исполняемого модуля,
    // удалите отдельные процедуры инициализации из
    // последующего кода.
    // Измените строку-аргумент функции (ключ в реестре, под
    // которым хранятся в реестре ваши установки).
    // TODO: измените эту строку на что-нибудь подходящее,
    // например, название вашей компании или организации
    SetRegistryKey(_T("Local AppWizard-Generated Applications"));
    LoadStdProfileSettings(4); // Загрузка установок из INI-файла
                               // (включая MRU)

```

```

// Регистрация шаблонов документов приложения. Шаблоны
// документов служат связью между документами, окнами документов
// и окнами представлений
CSingleDocTemplate* pDocTemplate;
pDocTemplate = new CSingleDocTemplate(
    IDR_MAINFRAME,
    RUNTIME_CLASS(CFractalViewDoc),
    RUNTIME_CLASS(CMainFrame), // главное окно
                                // SDI-приложения
    RUNTIME_CLASS(CFractalViewView));
AddDocTemplate(pDocTemplate);
// Поиск в командной строке команд управления, DDE,
// открытия файлов
CCommandLineInfo cmdInfo;
ParseCommandLine(cmdInfo);
// Выполнение команд, указанных в командной строке. Вернет
// FALSE, если приложение запускалось с /RegServer, /Register,
// /Unregserver или /Unregister.
if (!ProcessShellCommand(cmdInfo))
    return FALSE;
// Прорисовка и обновление единственного
// проинициализированного окна
m_pMainWnd->ShowWindow(SW_SHOW);
m_pMainWnd->UpdateWindow();

m_pMainWnd->SetWindowText ("Fractal Drawing");

return TRUE;
}

// CAboutDlg диалог, используемый в App About
class CAboutDlg : public CDialog
{
public:
    CAboutDlg();

// Данные для диалога
    enum { IDD = IDD_ABOUTBOX };

protected:
    virtual void DoDataExchange(CDataExchange* pDX);
    // Поддержка DDX/DDV

// Реализация
protected:
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
}

```



```

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
END_MESSAGE_MAP()

// Команда приложения на запуск диалога
void CFractalViewApp::OnAppAbout()
{
    CAboutDlg aboutDlg;
    aboutDlg.DoModal();
}

// Обработчики сообщений класса CFractalViewApp

BOOL CFractalViewApp::OnIdle(LONG lCount)
{
    // TODO: Добавьте сюда собственный код обработчика
    // или вызов базового класса

    CFractalViewView *PView =
        (CFractalViewView *) ((CFrameWnd *)m_pMainWnd)
        ->GetActiveView ();
    PView->DrawCol ();
    return TRUE;
}

```

---

### Листинг 19.3

```

// FractalViewDoc.h : интерфейс класса CFractalViewDoc
//

#pragma once

class CFractalViewDoc : public CDocument
{
protected: // используется только для сериализации
    CFractalViewDoc();
    DECLARE_DYNCREATE(CFractalViewDoc)

// Атрибуты
public:

// Операции
public:

// Переопределения
public:
    virtual BOOL OnNewDocument();
    virtual void Serialize(CArchive& ar);

// Реализация
public:
    virtual ~CFractalViewDoc();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif
}

```

```
protected:

// Сгенерированные функции карты сообщений
protected:
    DECLARE_MESSAGE_MAP()
};
```

---

#### Листинг 19.4

```
// FractalViewDoc.cpp : реализация класса CFractalViewDoc class
//

#include "stdafx.h"
#include "FractalView.h"

#include "FractalViewDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CFractalViewDoc

IMPLEMENT_DYNCREATE(CFractalViewDoc, CDocument)

BEGIN_MESSAGE_MAP(CFractalViewDoc, CDocument)
END_MESSAGE_MAP()

// Конструктор CFractalViewDoc

CFractalViewDoc::CFractalViewDoc()
{
    // TODO: добавьте сюда собственный код конструктора
}

CFractalViewDoc::~CFractalViewDoc()
{
}

BOOL CFractalViewDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;

    // TODO: добавьте сюда код реинициализации
    // (SDI-документы будут повторно использовать этот документ)

    return TRUE;
}

// Сериализация CFractalViewDoc

void CFractalViewDoc::Serialize(CArchive& ar)
{
}
```

```

        if (ar.IsStoring())
        {
            // TODO: добавьте сюда код сохранения
        }
        else
        {
            // TODO: добавьте сюда код загрузки
        }
    }

// Диагностика CFractalViewDoc

#ifdef _DEBUG
void CFractalViewDoc::AssertValid() const
{
    CDocument::AssertValid();
}

void CFractalViewDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}
#endif // _DEBUG

// Команды CFractalViewDoc

```

---

## Листинг 19.5

```

// FractalViewView.h : интерфейс класса CFractalViewView
//

#pragma once

class CFractalViewView : public CView
{
private:
    int m_Col;
    int m_ColMax;
    float m_CR;
    float m_DCI;
    float m_DCR;
    int m_RowMax;

public:
    void DrawCol ();

protected: // используется только для сериализации
    CFractalViewView();
    DECLARE_DYNCREATE(CFractalViewView)

// Атрибуты
public:
    CFractalViewDoc* GetDocument() const;

// Операции
public:

```

```

// Переопределения
public:
    virtual void OnDraw(CDC* pDC);
// переопределена для прорисовки этого вида
virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
protected:

// Реализация
public:
    virtual ~CFractalViewView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Сгенерированные функции карты сообщений
protected:
    DECLARE_MESSAGE_MAP()
public:
    afx_msg void OnSize(UINT nType, int cx, int cy);
};

#ifdef _DEBUG // отладочная версия в файле FractalViewView.cpp
inline CFractalViewDoc* CFractalViewView::GetDocument() const
{ return reinterpret_cast<CFractalViewDoc*>(m_pDocument); }
#endif

```

---

## Листинг 19.6

```

// FractalViewView.cpp : реализация класса CFractalViewView
//

#include "stdafx.h"
#include "FractalView.h"

#include "FractalViewDoc.h"
#include "FractalViewView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// определение набора констант Мандельброта
#define CIMAX 1.2
#define CIMIN -1.2
#define CRMAX 1.0
#define CRMIN -2.0
#define NMAX 128

// цвета, используемые для узора Мандельброта
DWORD ColorTable [6] =
{
    0x0000ff, // красный

```

```

        0x00ff00, // зеленый
        0xff0000, // синий
        0x00ffff, // желтый
        0xffff00, // бирюзовый
        0xff00ff, // фиолетовый
};

// CFractalViewView

IMPLEMENT_DYNCREATE(CFractalViewView, CView)

BEGIN_MESSAGE_MAP(CFractalViewView, CView)
    ON_WM_SIZE()
END_MESSAGE_MAP()

// Конструктор CFractalViewView

CFractalViewView::CFractalViewView()
{
    // TODO: добавьте сюда код конструктора
    m_Col = 0;
}

CFractalViewView::~CFractalViewView()
{
}

BOOL CFractalViewView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Измените класс или стили окна, изменяя
    // и добавляя поля структуры cs

    return CView::PreCreateWindow(cs);
}

// Прорисовка CFractalViewView

void CFractalViewView::OnDraw(CDC* /*pDC*/)
{
    CFractalViewDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    // TODO: добавьте сюда собственный код прорисовки

    m_Col = 0;
    m_CR = (float)CRMIN;
}

// Диагностика CFractalViewView

#ifdef _DEBUG
void CFractalViewView::AssertValid() const
{
    CView::AssertValid();
}

```

```

void CFractalViewView::Dump(CDumpContext& dc) const
{
    CView::Dump(dc);
}

CFractalViewDoc* CFractalViewView::GetDocument() const
// неотладочная версия встроена
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CFractalViewDoc)));
    return (CFractalViewDoc*)m_pDocument;
}
#endif // _DEBUG

// Обработчики сообщений CFractalViewView

void CFractalViewView::OnSize(UINT nType, int cx, int cy)
{
    CView::OnSize(nType, cx, cy);

    // TODO: Добавьте сюда собственный код обработчика
    if (cx <= 1 || cy <= 1)
        // проверка деления на ноль
        return;
    m_ColMax = cx;
    m_RowMax = cy;

    m_DCR = (float)((CRMAX - CRMIN) / (m_ColMax - 1));
    m_DCI = (float)((CIMAX - CIMIN) / (m_RowMax - 1));
}

void CFractalViewView::DrawCol ()
{
    CClientDC ClientDC (this);
    float CI;
    int ColorVal;
    float I;
    float ISqr;
    float R;
    float RSqr;
    int Row;

    if (m_Col >= m_ColMax || GetParentFrame ()->IsIconic ())
        return;

    CI = (float) CIMAX;
    for (Row = 0; Row < m_RowMax; Row++)
    {
        R = (float)0.0;
        I = (float)0.0;
        RSqr = (float)0.0;
        ISqr = (float)0.0;
        ColorVal = 0;
        while (ColorVal < NMAX && RSqr + ISqr < 4)
        {
            ++ColorVal;

```

```

        RSqr = R * R;
        ISqr = I * I;
        I *= R;
        I += I + CI;
        R = RSqr - ISqr + m_CR;
    }
    ClientDC.SetPixelV (m_Col, Row, ColorTable
        [ColorVal % 6]);
    CI -= m_DCI;
}
m_Col++;
m_CR += m_DCR;
}

```

---

### Листинг 19.7

```

// MainFrm.h : интерфейс класса CMainFrame
//

#pragma once
class CMainFrame : public CFrameWnd
{

protected: // используются только для сериализации
    CMainFrame();
    DECLARE_DYNCREATE(CMainFrame)

// Атрибуты
public:

// Операции
public:

// Переопределения
public:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);

// Реализация
public:
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

// Сгенерированные функции карты сообщений
protected:
    DECLARE_MESSAGE_MAP()
};

```

---

### Листинг 19.8

```

// MainFrm.cpp : реализация класса CMainFrame
//

#include "stdafx.h"

```

```

#include "FractalView.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
END_MESSAGE_MAP()

// Конструктор CMainFrame

CMainFrame::CMainFrame()
{
    // TODO: добавьте сюда код инициализации элементов
}

CMainFrame::~CMainFrame()
{
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    if( !CFrameWnd::PreCreateWindow(cs) )
        return FALSE;
    // TODO: Измените класс или стили окна, изменяя или
    // добавляя поля структуре cs

    return TRUE;
}

// Диагностика CMainFrame

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
    CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
    CFrameWnd::Dump(dc);
}

#endif // _DEBUG

// Обработчики сообщений CMainFrame

```



## Отрезки линий

Рассмотрим, как рисуются следующие разновидности отрезков линий:

- *отрезки прямых линий* и их наборы;
- *отрезки регулярных кривых*, являющихся частями эллипсов;
- *отрезки лекальных кривых* (кривых Безье).

Будут рассмотрены также режимы рисования фона, влияющие на отрезки прямых и кривых линий.

### Отрезки прямых

Для рисования отрезка прямой линии:

1. Вызовите функцию `CDC::MoveTo` для описания начальной точки отрезка.
2. Затем вызовите `CDC::LineTo` для описания конечной точки и генерации самого отрезка. Например, следующий фрагмент программы рисует отрезок прямой, соединяющий точки (5, 15) и (30,75).

```
pDC->MoveTo (5, 15);  
pDC->LineTo (30, 75);
```

Передаваемые в функцию `MoveTo` параметры описывают горизонтальные и вертикальные координаты новой *текущей позиции*. Если объект контекста устройства создается впервые, то текущая позиция имеет логические координаты (0, 0). Функция `LineTo` рисует отрезок прямой из текущей позиции до конечной точки, заданной переданными в нее параметрами. Функция `LineTo` также изменяет текущую позицию, чтобы задать конечную точку. Таким образом, при рисовании последовательности соединенных между собой отрезков прямых, необходимо вызвать функцию `MoveTo` только перед первым вызовом функции `LineTo`. Например, следующий фрагмент программы рисует последовательность соединенных отрезков прямых, формирующих букву "W".

```
pDC->MoveTo (50, 50);  
pDC->LineTo (150, 225);  
pDC->LineTo (225, 150);  
pDC->LineTo (300, 225);  
pDC->LineTo (375, 75);
```

Существует альтернативный способ рисования последовательно соединенных отрезков прямых: посредством вызова функции `CDC::Polyline`. Первый параметр, передаваемый в `Polyline` – это указатель на массив структуры `POINT`, содержащий точки соединения, а второй – задает общее количество точек. Например, следующий фрагмент программы рисует последовательность соединенных отрезков прямых такую же, как в предыдущем примере.

```
POINT Points [5]  
  
Points [0].x = 30;  
Points [0].y = 30;  
Points [1].x = 85;  
Points [1].y = 120;  
Points [2].x = 120;  
Points [2].y = 100;  
Points [3].x = 180;  
Points [3].y = 180;  
Points [4].x = 230;  
Points [4].y = 55;  
  
pDC->Polyline (Points, 5);
```

Есть несколько разновидностей функции Polyline:

- Функция Polyline никогда не переустанавливает текущую позицию.
- Функция CDC::PolylineTo наоборот, рисует отрезок прямой от текущей позиции до первой указанной точки и перемещает текущую позицию в последнюю заданную точку.
- Можно вызвать функцию CDC::PolyPolyline для рисования совокупности отдельных наборов линий (в каждом наборе отрезки прямых соединены между собой, а сами наборы между собой не соединены). Подобно функции Polyline функция PolyPolyline не изменяет установки текущей позиции.

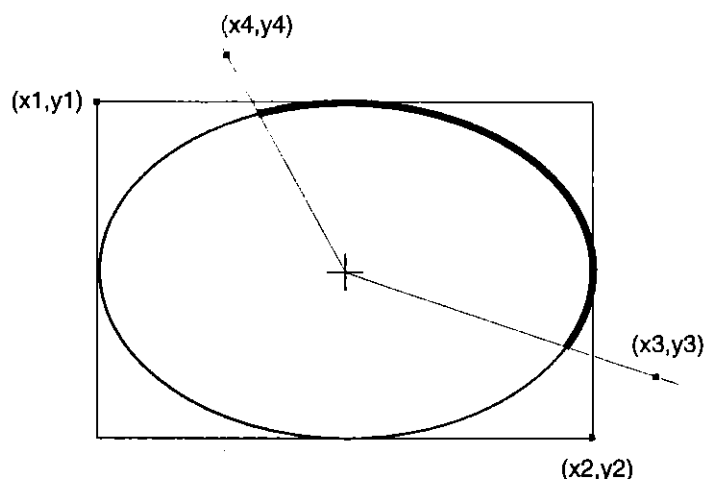
Наконец, если любая из этих функций используется для рисования фигуры с замкнутым контуром, то фигура не заливается цветом автоматически, в отличие от функций рисования таких фигур, рассмотренных в параграфе “Рисование замкнутых фигур”.

## Отрезки регулярных кривых (фрагменты эллипсов)

Отрезки дуг, являющиеся сегментами эллипсов, рисуют с помощью функции CDC::Arc. Обратите внимание: функции Arc передаются четыре пары координат. Функция Arc никогда не использует и не корректирует текущую позицию. Первая пара описывает левый верхний угол прямоугольника, который ограничивает эллипс (рисуемая дуга является сегментом этого эллипса); вторая – правый нижний угол ограничивающего прямоугольника. Третья – начальную точку дуги, а четвертая – конечную. По умолчанию дуга рисуется из начальной точки в конечную, против часовой стрелки. Для изменения направления рисования вызовите функцию CDC::SetArcDirection.

```
BOOL Arc
(int x1, int y1, // левый верхний угол ограничивающего
                      // прямоугольника;
 int x2, int y2, // правый нижний угол ограничивающего
                      // прямоугольника;
 int x3, int y3, // начальная точка дуги;
 int x4, int y4 ); // конечная точка дуги
```

Начальная точка необязательно должна принадлежать эллипсу. Она может быть где-либо на линии, проходящей через центр ограничивающего прямоугольника и начальную точку на эллипсе, задавая угол начала дуги. Это справедливо и для задания конечной точки. На следующем рисунке показаны координаты, заданные с результирующей дугой.



## Отрезки локальных кривых (кривых Безье)

Отрезок локальной кривой, которая *не* является дугой эллипса, можно нарисовать, вызывая функцию `CDC::PolyBezier`. Функция `PolyBezier` рисует последовательность соединенных отрезков кривых, называемых *сплайном Безье* (кривой Безье).

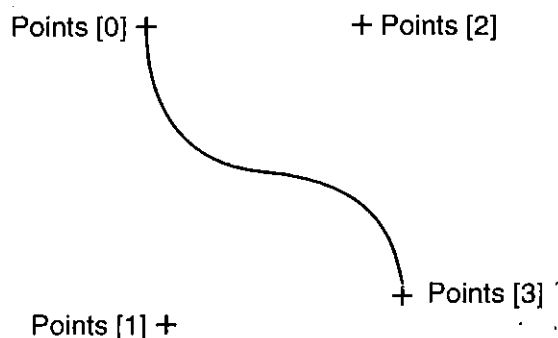
```
BOOL PolyBezier
(const POINT* lpPoints, // указатель на массив структур POINT
 int nCount);           // число структур в массиве
```

Рассмотрим рисование одиночного сплайна. Следующий фрагмент программы рисует одиночный сплайн. Массив, передаваемый в первом параметре, содержит четыре структуры `POINT` с координатами точек кривой: первая – начальная, вторая и третья – управляющие, задающие форму кривой, четвертая – конечная точка кривой.

```
// Рисование кривой, состоящей из одного сплайна:
POINT Points [4];

Points [0].x = 25; Points [0].y = 25;
Points [1].x = 35; Points [1].y = 170;
Points [2].x = 130; Points [2].y = 20;
Points [3].x = 150; Points [3].y = 150;

pDC->PolyBezier (Points, 4);
```



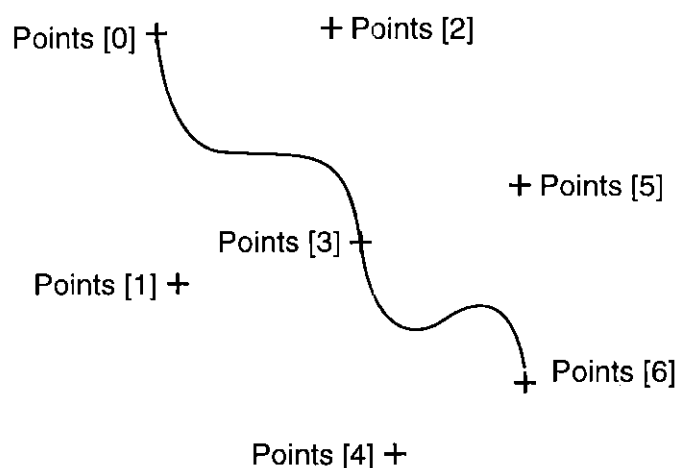
Для рисования следующего сплайна нужно прибавить к массиву только *три* дополнительные точки: первая и вторая – управляют точками нового сплайна, третья – конечная точка нового сплайна. Начальная точка нового сплайна совпадает с конечной точкой первого. Например, следующий фрагмент программы рисует отрезок кривой, состоящий из двух соединенных сплайнов (первый такой же, как в предыдущем примере).

```
// Рисование кривой, состоящей из двух сплайнов:
POINT Points [7];

Points [0].x = 25; Points [0].y = 25;
Points [1].x = 35; Points [1].y = 170;
Points [2].x = 130; Points [2].y = 20;
Points [3].x = 150; Points [3].y = 150;
Points [4].x = 170; Points [4].y = 280;
Points [5].x = 250; Points [5].y = 115;
Points [6].x = 250; Points [6].y = 225;

pDC->PolyBezier (HWinDC, Points, 7);
```

В массиве, присваиваемое параметру `nCount`, должно быть элементов `POINT` на единицу больше трехкратного числа требуемых сплайнов. Поскольку конечная точка данного сплайна используется как начальная точка следующего, отдельные сплайны, нарисованные с помощью функции `PolyBezier`, всегда соединены. Чтобы переход от одного сплайна к следующему был *плавным*, убедитесь, что последняя управляющая точка первого сплайна, конечная точка первого сплайна (являющаяся начальной точкой второго) и первая управляющая точка второго сплайна лежат на одной прямой. Например, на рисунке, изображающем сплайн, переход плавный потому, что точки `Points[2]`, `Points[3]` и `Points[4]` лежат на одной прямой. Функция `PolyBezier` никогда не использует и не изменяет текущую позицию. Функция `CDC::PolyBezierTo` меняет текущую позицию.



## Режим рисования

Для отрезков линий, нарисованных с использованием описанных выше функций, стиль, толщина и цвет определяются типом пера, выбранного в текущий момент в объекте контекста устройства. На рисование линий также влияет текущий *режим рисования*, который описывает способ комбинирования цвета пера с текущим цветом дисплея. Окончательный цвет каждого пикселя зависит от текущего цвета пикселя, цвета пера и режима рисования. В стандартном режиме Windows просто копирует цвет пера на дисплей (если перо красное, каждый пиксель нарисованной линии будет окрашен красным цветом, независимо от его текущего цвета). Вызвав функцию-член `SetROP2` класса `CDC`, режим рисования можно изменить. Параметр `nDrawMode` описывает желаемый режим рисования.

```
int SetROP2 (int nDrawMode);
```

Существует 16 возможных режимов рисования. Наиболее распространенные из них перечислены в табл. 19.8. Описание более сложных из них смотрите в документации на эту функцию. В таблице приведен результирующий цвет каждого пикселя линии, нарисованной в соответствии с используемым режимом рисования. Значение `RC_COPYPEN` задает стандартный режим рисования. При выборе режима рисования `R2_NOT` линия будет нарисована цветом, инверсным по отношению к экрану. Этот метод рисования имеет несколько преимуществ:

- линия видна на экране любого цвета;
- можно использовать режим `R2_NOT` для рисования видимой линии внутри области, содержащей смесь цветов.

При повторном рисовании этой же линии в режиме `R2_NOT` она автоматически удаляется, а цвет экрана восстанавливается. Этот режим можно использовать для рисования прямоугольников выборки,

создания анимации и других целей. Выбор пера NULL и кисти NULL можно смоделировать, задав режим R2\_NOT. Режим рисования *также* влияет на рисование границ и внутренних областей замкнутых фигур, рассматриваемых в следующем параграфе.

Табл. 19.8. Режимы рисования, устанавливаемые вызовом функции SetROP2

Значение параметра dDrawMode	Цвет каждого пикселя рисуемой фигуры
RC_COPYPEN (Стандартный режим)	Соответствует цвету пера
RC_NOTCOPYPEN	Инверсный цвету пера
R2_NOT	Инверсный цвету фона
R2_BLACK	Черный
R2_WHITE	Белый
R2_NOP	Не изменяется

## Фон и прерывистые линии

Для прерывистых линий (использующих стили PS\_DASH, PS\_DOT, PS\_DASHDOT или PS\_DASHDOTDOT) цвет, которым закрашиваются пробелы в линии, зависит от текущего режима фона и его цвета. Вспомните (гл. 18): режим фона устанавливается вызовом функции CDC::SetBkMode. Если аргументу nBkMode присвоено значение по умолчанию OPAQUE, пропуски внутри линий будут закрашены текущим цветом фона. Если присвоено значение TRANSPARENT, пробелы не закрашиваются (экранные цвета остаются неизменными).

```
int SetBkMode (int nBkMode);
```

Цвет фона устанавливается при вызове CDC::SetBkColor.

```
virtual COLORREF SetBkColor (COLORREF crColor);
```

В гл. 18 рассмотрено воздействие режима и цвета фона на рисование текста.

## Рисование замкнутых фигур

Режим и цвет фона также влияют на границу вокруг фигур с замкнутым контуром, нарисованных перьями для прерывистых линий, а также на внутренние области этих фигур, нарисованных кистями со штриховкой. Следующие функции класса CDC позволяют рисовать фигуры с замкнутым контуром, ограничивающие одну или несколько областей поверхности экрана:

Табл. 19.9. Функции рисования класса CDC для рисования замкнутых фигур

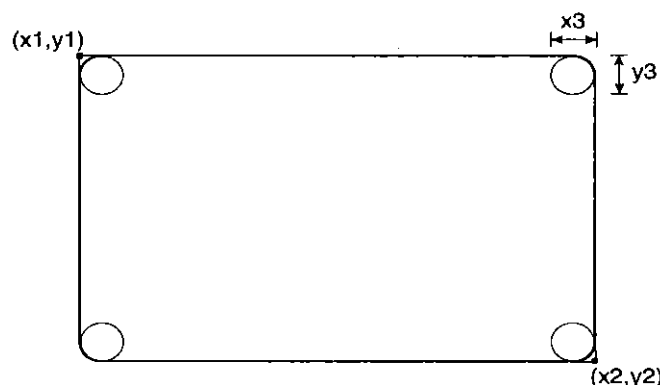
Функция	Действие
Rectangle	Прямоугольник
RoundRect	Закругленный прямоугольник
Ellipse	Эллипс
Chord	Сегмент эллипса
Pie	Сектор
Polygon	Многоугольник
PolyPolygon	Набор многоугольников

Простой прямоугольник нарисует функция `CDC::Rectangle`. В приведенном ниже примере верхний левый угол этого прямоугольника находится в точке с координатами (25, 50), а нижний правый угол – в точке (175, 225).

```
pDC->Rectangle (25, 50, 175, 225);
```

Для рисования закругленного прямоугольника обращаются к функции `CDC::RoundRect`. Первая пара координат, передаваемая в `RoundRect`, описывает позицию верхнего левого угла прямоугольника, вторая – правого нижнего угла; третья – задает ширину и высоту прямоугольника, ограничивающего эллипс, используемый для рисования закругленных углов.

```
BOOL RoundRect
(int x1, int y1, // левый верхний угол прямоугольника;
 int x2, int y2, // правый нижний угол прямоугольника;
 int x3, int y3); // размер закругляющих эллипсов
```

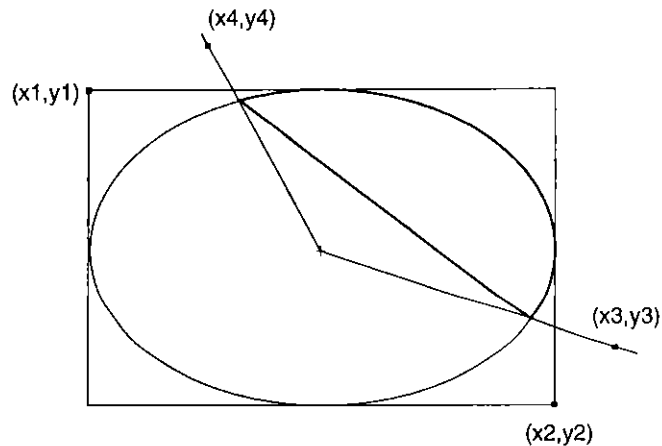


Круг или эллипс рисуют с помощью функции `CDC::Ellipse`. Первая пара координат описывает координаты верхнего левого угла прямоугольника, ограничивающего эллипс, вторая – координаты правого нижнего угла ограничивающего прямоугольника.

```
BOOL Ellipse
(int x1, int y1, // левый верхний угол ограничивающего
 // прямоугольника;
 int x2, int y2); // правый нижний угол ограничивающего
 // прямоугольника
```

Функция `CDC::Chord` позволяет нарисовать хорду. Хорда отсекает фигуру, образованную пересечением эллипса и сегмента линии. Первые две пары координат, передаваемые в функцию `Chord`, описывают прямоугольник, ограничивающий эллипс; третья – начальную точку хорды; четвертая – конечную точку. Как и в функции `Arc`, заданная хордой фигура рисуется из начальной точки в конечную против часовой стрелки, если для изменения направления не вызвана функция `SetArcDirection`, а заданные начальная и конечная точки не лежат на эллипсе.

```
BOOL Chord
(int x1, int y1, // левый верхний угол ограничивающего
 // прямоугольника;
 int x2, int y2, // правый нижний угол ограничивающего
 // прямоугольника;
 int x3, int y3, // начальная точка хорды;
 int x4, int y4); // конечная точка хорды
```



Сектор рисуется функцией `CDC::Pie`. Координаты, передаваемые в функцию `Pie`, аналогичны координатам, передаваемым в функции `Arc` и `Chord`.

```

BOOL Pie
    (int x1,  int y1,      // левый верхний угол ограничивающего
                                // прямоугольника;
    int x2,  int y2,      // правый нижний угол ограничивающего
                                // прямоугольника;
    int x3,  int y3,      // начальная точка сектора;
    int x4,  int y4);     // конечная точка сектора

```

Многоугольник, состоящий из двух или более вершин, соединенных линиями, рисует функция `CDC::Polygon`. Например, следующие строки рисуют треугольник. Первый параметр, передаваемый в функцию `Polygon`, является указателем на массив `POINT`. Элементы этого массива описывают координаты вершин. Второй параметр показывает число вершин, которые необходимо соединить.

```

POINT Points [3];

Points [0].x = 20;
Points [0].y = 10;
Points [1].x = 30;
Points [1].y = 30;
Points [2].x = 10;
Points [2].y = 30;

pDC->Polygon (Points, 3);

```

В отличие от функции `Polyline`, рассмотренной в предыдущем разделе, функция `Polygon` всегда создает замкнутую фигуру (кроме случая, когда соединяемых вершин всего две). Чтобы фигура имела замкнутый контур, при необходимости проводится линия из последней вершины в первую. В приведенном примере функция `Polygon` соединяет первую точку (`Point[0]`) со второй, вторую с третьей и третью с первой, образуя треугольник. Если этот массив точек передать функции `Polyline`, то она соединит первую точку со второй, а вторую точку с третьей. Вызвав функцию `CDC::PolyPolygon` можно нарисовать несколько отдельных многоугольников.

Границы каждой из замкнутых фигур, рассмотренных в этом параграфе, рисуются с помощью текущего пера, а внутренние области заполняются текущей кистью. Обратите внимание: при рисовании замкнутой фигуры пером, которому присвоен стиль `PS_INSIDEFRAME`, граница рисуется внутри ограничивающего прямоугольника. Если нужно нарисовать замкнутую фигуру без закрашивания (т.е. оставить внутреннюю область без изменений), то вызывается функция `CDC::SelectStockObject`,

позволяющая выбрать объект `NULL_BRUSH` перед рисованием. Текущий режим рисования, установленный при вызове функции `SetROP2`, влияет на способ формирования границ и внутренних областей замкнутых фигур так же, как на рисование линий (см. параграф “Режим рисования”).

Вызовы `CDC::SetBkMode` и `CDC::SetBkColor` задают режим и цвет фона, которые влияют на фигуры с замкнутыми контурами следующим образом. Если выбрано перо для прерывистой линии (созданное в стиле `PS_DASH`, `PS_DOT`, `PS_DASHDOT` или `PS_DASHDOTDOT`), то режим и цвет фона управляют рисованием пробелов в границах фигур. В режиме `OPAQUE` пробелы заполняются цветом фона, а в режиме `TRANSPARENT` – не заполняются. Если используется кисть с узорами (созданная вызовом `CreateHatchBrush` или `CreatePatternBrush`), то для выравнивания заполняющего узора вызывают функцию `SetBrushOrg` класса `CDC`. Если используется кисть со штриховкой (созданная вызовом `CreateHatchBrush`), то режим и цвет фона управляют заполнением пробелов между линиями штриховки, т.е. в режиме `OPAQUE` пробелы закрашиваются цветом фона, а в режиме `TRANSPARENT` пробелы не закрашиваются.

Ниже приведено описание нескольких дополнительных функций рисования, принадлежащих классу `CDC`.

Табл. 19.10. Дополнительные функции рисования

Функция	Цель
<code>DrawFocusRect</code>	Рисует границу прямоугольника, используя пунктирную линию, без заливки внутренней области. Граница рисуется цветом, инверсным цвету экрана. Повторный вызов функции с этими же координатами удаляет границу
<code>DrawIcon</code>	Рисует значок
<code>ExtFloodFill</code>	Заполняет область, ограниченную данным цветом, используя текущую кисть. Можно закрасить область, <i>заполненную</i> указанным цветом
<code>FillRect</code>	Заполняет прямоугольную область, используя указанную кисть без рисования границ
<code>FloodFill</code>	Заполняет область, ограниченную данным цветом, используя текущую кисть
<code>FrameRect</code>	Рисует прямоугольную границу, используя указанную кисть, без заполнения внутренней области
<code>InvertRect</code>	Инвертирует цвет внутри прямоугольной области
<code>PolyDraw</code>	Рисует фигуры, состоящие из комбинаций прямых и кривых линий, т.е. из сегментов прямых линий и сплайнов Безье

## Программа ScratchBook

Описана версия программы `ScratchBook`, в которую добавлена возможность рисования множества разных форм. Отдельная форма зависит от выбранного инструмента рисования, который выбирается щелчком на кнопке в панели инструментов или выбором команды в меню `Tools`. Предусмотрена также возможность задания толщины линий, используемых для рисования фигур (одинарная, двойная или тройная) щелчком на кнопке в панели инструментов или выбором команды в меню `Options`. Кроме того, разрешен выбор цвета фигуры с помощью команды в меню `Color`. Фрагмент, добавленный в программу `ScratchBook`, демонстрирует не только методы рисования, описанные в этой главе, но и способы конструирования иерархии классов и использования полиморфизма (см. гл. 5). Для достижения необходимых результатов следует внести изменения в файл программы `ScratchBook`, созданный в гл. 14.

Начнем с настройки меню программы. Для этого откройте в `Developer Studio` проект `ScratchBook`, затем откройте вкладку `Resource View` и меню `IDR_MAINFRAME` в редакторе меню. Создайте меню `Color` с элементами, перечисленными в таблице 19.11. Создайте обработчики сообщений `COMMAND` и



UPDATE\_COMMAND для всех команд меню. Примите имя функции, заданное по умолчанию. В табл. 19.12 для каждого из обработчиков сообщений приведены идентификаторы команды и сообщения, а также стандартное имя функции обработки сообщений.

Табл. 19.11. Элементы меню Color

Идентификатор (ID)	Надпись (Caption)	Другие свойства
ID_COLOR_BLACK	&Black	
ID_COLOR_WHITE	&White	
ID_COLOR_RED	&Red	
ID_COLOR_GREEN	&Green	
ID_COLOR_BLUE	&Blue	
ID_COLOR_YELLOW	&Yellow	
ID_COLOR_CYAN	&Cyan	
ID_COLOR_MAGENTA	&Magenta	
ID_COLOR_CUSTOM	&Custom...	

Табл. 19.12. Обработчики сообщений для команд меню Color

Идентификатор (ID)	Сообщение	Имя функции
ID_COLOR_BLACK	COMMAND	OnColorBlack
	UPDATE_COMMAND_UI	OnUpdateColorBlack
ID_COLOR_BLUE	COMMAND	OnColorBlue
	UPDATE_COMMAND_UI	OnUpdateColorBlack
ID_COLOR_CUSTOM	COMMAND	OnColorCustom
	UPDATE_COMMAND_UI	OnUpdateColorCustom
ID_COLOR_CYAN	COMMAND	OnColorCyan
	UPDATE_COMMAND_UI	OnUpdateColorCyan
ID_COLOR_GREEN	COMMAND	OnColorGreen
	UPDATE_COMMAND_UI	OnUpdateColorGreen
ID_COLOR_MAGENTA	COMMAND	OnColorMagenta
	UPDATE_COMMAND_UI	OnUpdateColorMagenta
ID_COLOR_RED	COMMAND	OnColor Red
	UPDATE_COMMAND_UI	OnUpdateColorRed
ID_COLOR_WHITE	COMMAND	OnColorWhite
	UPDATE_COMMAND_UI	OnUpdateColorWhite
ID_COLOR_YELLOW	COMMAND	OnColorYellow
	UPDATE_COMMAND_UI	OnUpdateColorYellow

Откройте файл ScratchBook.h и добавьте определения двух новых переменных в начало определения класса CScratchBookApp. Переменная m\_CurrentColor хранит значение активного цвета, используемого для рисования фигур, а m\_IdxCOLORCmd хранит идентификатор команды меню Color, выбранной для получения этого цвета.

```
class CScratchBookApp : public CWinApp
{
public:
    int m_CurrentWidth;
```

```

UINT m_CurrentTool;
COLORREF m_CurrentColor;
UINT m_IdxCmd;

```

Для инициализации этих переменных откройте файл ScratchBook.cpp и добавьте операторы, выделенные полужирным шрифтом, в конструктор класса CScratchBookApp. После этого начальный цвет будет установлен в черный.

```

CScratchBookApp::CScratchBookApp()
{
    // TODO: добавьте сюда код конструктора
    // Поместите весь существенный код инициализации
    // в функцию InitInstance

    m_CurrentColor = RGB (0, 0, 0);
    m_CurrentWidth = 1;
    m_CurrentTool=ID_TOOLS_LINE;
    m_IdxCmd = ID_COLOR_BLACK;
}

```

В файл ScratchBook.cpp добавьте обработчики новых команд меню. Приведенные обработчики сообщений меню работают так же, как обработчики, добавленные в программу в гл. 14. Пояснения смотрите в параграфе “Обработчики сообщений кнопок и команд” гл. 14.

```

void CScratchBookApp::OnColorBlack()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentColor = RGB (0, 0, 0);
    m_IdxCmd = ID_COLOR_BLACK;
}

void CScratchBookApp::OnUpdateColorBlack(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck (m_IdxCmd == ID_COLOR_BLACK ? 1 : 0);
}

void CScratchBookApp::OnColorWhite()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentColor = RGB (255, 255, 255);
    m_IdxCmd = ID_COLOR_WHITE;
}

void CScratchBookApp::OnUpdateColorWhite(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck(m_IdxCmd == ID_COLOR_WHITE ? 1 : 0);
}

void CScratchBookApp::OnColorRed()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentColor = RGB (255, 0, 0);
    m_IdxCmd = ID_COLOR_RED;
}

```

```

void CScratchBookApp::OnUpdateColorRed(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck(m_IdxColorCmd == ID_COLOR_RED ? 1 : 0);
}

void CScratchBookApp::OnColorGreen()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentColor = RGB (0, 255, 0);
    m_IdxColorCmd = ID_COLOR_GREEN;
}

void CScratchBookApp::OnUpdateColorGreen(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck(m_IdxColorCmd == ID_COLOR_GREEN ? 1 : 0);
}

void CScratchBookApp::OnColorBlue()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentColor = RGB (0, 0, 255);
    m_IdxColorCmd = ID_COLOR_BLUE;
}

void CScratchBookApp::OnUpdateColorBlue(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck(m_IdxColorCmd == ID_COLOR_BLUE ? 1 : 0);
}

void CScratchBookApp::OnColorYellow()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentColor = RGB (255, 255, 0);
    m_IdxColorCmd = ID_COLOR_YELLOW;
}

void CScratchBookApp::OnUpdateColorYellow(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck (m_IdxColorCmd == ID_COLOR_YELLOW ? 1 : 0);
}

void CScratchBookApp::OnColorCyan()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentColor = RGB (0, 255, 255);
    m_IdxColorCmd = ID_COLOR_CYAN;
}

void CScratchBookApp::OnUpdateColorCyan(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика

```

```

    pCmdUI->SetCheck(m_IdxColorCmd == ID_COLOR_CYAN ? 1 : 0);
}

void CScratchBookApp::OnColorMagenta()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentColor = RGB (255, 0, 255);
    m_IdxColorCmd = ID_COLOR_MAGENTA;
}

void CScratchBookApp::OnUpdateColorMagenta(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck(m_IdxColorCmd == ID_COLOR_MAGENTA ? 1 : 0);
}

void CScratchBookApp::OnColorCustom()
{
    // TODO: Добавьте сюда собственный код обработчика
    CColorDialog ColorDialog;

    if (ColorDialog.DoModal () == IDOK)
    {
        m_CurrentColor = ColorDialog.GetColor ();
        m_IdxColorCmd = ID_COLOR_CUSTOM;
    }
}

void CScratchBookApp::OnUpdateColorCustom(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI ->SetCheck(m_IdxColorCmd == ID_COLOR_CUSTOM ? 1 : 0);
}

```

Обратите внимание: функция ColorCustom, получающая управление, при выборе команды Custom... в меню Color отображает диалоговое окно Color, позволяющее выбрать настраиваемый цвет. Способы отображения обычного диалогового окна рассмотрены в параграфе “Создание текста и объект Font” гл. 18.

## Классы фигур

В файле заголовков ScratchBookDoc.h для класса документа *удалите* текущее определение класса CLine. Затем добавьте следующие определения классов.

```

class CFigure: public CObject
{
protected:
    COLORREF m_Color;
    DWORD m_X1, m_Y1, m_X2, m_Y2;
    CFigure () {}
    DECLARE_SERIAL (CFigure)

public:
    virtual void Draw (CDC *PDC) {}
}

```

```

    CRect GetDimRect ();
    virtual void Serialize (CArchive& ar);
};

class CLine: public CFigure
{
protected:
    DWORD m_Width;
    CLine () {}
    DECLARE_SERIAL (CLine)

public:
    CLine (int X1, int Y1, int X2, int Y2, COLORREF Color,
int Width);
    virtual void Draw (CDC *PDC);
    virtual void Serialize (CArchive& ar);
};

class CRectangle: public CFigure
{
protected:
    DWORD m_Width;
    CRectangle () {}
    DECLARE_SERIAL (CRectangle)

public:
    CRectangle (int X1, int Y1, int X2, int Y2, COLORREF Color,
int Width);
    virtual void Draw (CDC *PDC);
    virtual void Serialize (CArchive& ar);
};

class CFRectangle: public CFigure
{
protected:
    DWORD m_Width;
    CFRectangle () {}
    DECLARE_SERIAL (CFRectangle)

public:
    CFRectangle (int X1, int Y1, int X2, int Y2, COLORREF Color,
int Width);
    virtual void Draw (CDC *PDC);
    virtual void Serialize (CArchive& ar);
};

class CRRectangle: public CFigure
{
protected:
    DWORD m_Width;
    CRRectangle () {}
    DECLARE_SERIAL (CRRectangle)

```

```

public:
    CRRectangle (int X1, int Y1, int X2, int Y2, COLORREF Color,
int Width);
    virtual void Draw (CDC *PDC);
    virtual void Serialize (CArchive& ar);
};

class CFRRectangle: public CFigure
{
protected:
    DWORD m_Width;
    CFRRectangle () {}
    DECLARE_SERIAL (CFRRectangle)

public:
    CFRRectangle (int X1, int Y1, int X2, int Y2, COLORREF Color,
int Width);
    virtual void Draw (CDC *PDC);
    virtual void Serialize (CArchive& ar);
};

class CCircle: public CFigure
{
protected:
    DWORD m_Width;
    CCircle () {}
    DECLARE_SERIAL (CCircle)

public:
    CCircle (int X1, int Y1, int X2, int Y2, COLORREF Color,
int Width);
    virtual void Draw (CDC *PDC);
    virtual void Serialize (CArchive& ar);
};

class CFCircle: public CFigure
{
protected:
    DWORD m_Width;
    CFCircle () {}
    DECLARE_SERIAL (CFCircle)

public:
    CFCircle (int X1, int Y1, int X2, int Y2, COLORREF Color);
    virtual void Draw (CDC *PDC);
    virtual void Serialize (CArchive& ar);
};

```

В иерархии определяемых классов есть единый базовый класс CFigure, порожденный от CObject, и порожденный от CFigure класс для каждой фигуры, которую можно нарисовать. Класс CFigure содержит переменные и функции, используемые для *всех* типов фигур, а порожденный класс фигуры – переменные и функции, используемые для фигуры конкретного типа. В частности, класс CFigure содержит переменную m\_Color для хранения цвета, поскольку фигуры всех типов

имеют цвет. Аналогично, класс CFigure содержит координаты (m\_X1, m\_Y1, m\_X2, m\_Y2), потому что положение и размер всех типов фигур задается с использованием четырех координат. Напротив, только линии и незакрашенные фигуры имеют толщину линии (закрашенные фигуры ее не имеют). Следовательно, переменная m\_Width для хранения толщины содержится только в классах для незакрашенных фигур и линий.

Каждый порожденный класс имеет собственный конструктор для инициализации переменных. Класс CFigure также предоставляет функцию, вычисляющую размеры прямоугольника, ограничивающего фигуру (GetDimRect). Этот класс содержит виртуальную функцию Serialize для чтения с диска и записи на диск переменных класса CFigure. Классы, которые не имеют переменной m\_Width, могут полагаться на функцию CFigure::Serialize. Однако каждый класс, имеющий эту переменную, должен иметь собственную функцию Serialize для чтения и записи значения m\_Width. Каждая из функций Serialize явно вызывает функцию CFigure::Serialize для чтения и записи переменных, определенных в классе CFigure.

Класс CFigure содержит виртуальную функцию Draw. Определение этой функции позволяет программе использовать единственный указатель класса CFigure для вызова функции Draw применительно к фигуре любого типа. Класс фигуры каждого типа содержит переопределенную функцию Draw, выполняющую рисование с помощью подпрограммы, соответствующей определенному типу фигуры. Как видно, такое применение полиморфизма намного упрощает код в классе представления, управляющий рисованием фигур. Объяснение порождения классов, наследования, виртуальных функций и полиморфизма см. в гл. 5. Заметим, что можно (но не обязательно) определить функцию CFigure::Draw как *чисто виртуальную функцию*, сделав класс CFigure *абстрактным* базовым классом. Однако, поскольку чисто виртуальные функции не рассматриваются в части II этой книги, Draw представляет собой обыкновенную виртуальную функцию.

Откройте файл ScratchBookDoc.cpp и удалите код реализации класса CLine, а также код реализации функций CLine::Draw, CLine::Serialize и CLine::GetDimRect. Затем добавьте в конце файла следующие строки для реализации функций-членов классов фигур.

```
// реализация классов фигур

IMPLEMENT_SERIAL (CFigure, CObject, 2)

CRect CFigure::GetDimRect ()
{
    return CRect
        (min (m_X1, m_X2), min (m_Y1, m_Y2),
         (max (m_X1, m_X2))+1, (max (m_Y1, m_Y2))+1);
}

void CFigure::Serialize (CArchive& ar)
{
    if (ar.IsStoring ())
        ar << m_X1 << m_Y1 << m_X2 << m_Y2 << m_Color;
    else
        ar >> m_X1 >> m_Y1 >> m_X2 >> m_Y2 >> m_Color;
}

IMPLEMENT_SERIAL (CLine, CFigure, 2);

CLine::CLine (int X1, int Y1, int X2, int Y2,
              COLORREF Color, int Width)
{
    m_X1 = X1;
```

```

    m_Y1 = Y1;
    m_X2 = X2;
    m_Y2 = Y2;
    m_Color = Color;
    m_Width = Width;
}

void CLine::Serialize (CArchive &ar)
{
    CFigure::Serialize (ar);
    if (ar.IsStoring ())
        ar << m_Width;
    else
        ar >> m_Width;
}

void CLine::Draw (CDC *PDC)
{
    CPen Pen, *POldPen;

    // выбор пера/кисти
    Pen.CreatePen (PS_SOLID, m_Width, m_Color);
    POldPen = PDC->SelectObject (&Pen);

    // рисование фигуры
    PDC->MoveTo (m_X1, m_Y1);
    PDC->LineTo (m_X2, m_Y2);

    // восстановление пера/кисти
    PDC->SelectObject (POldPen);
}

IMPLEMENT_SERIAL (CRectangle, CFigure, 2);

CRectangle::CRectangle (int X1, int Y1, int X2, int Y2,
                        COLORREF Color, int Width)
{
    m_X1 = X1;
    m_Y1 = Y1;
    m_X2 = X2;
    m_Y2 = Y2;
    m_Color = Color;
    m_Width = Width;
}

void CRectangle::Serialize (CArchive &ar)
{
    CFigure::Serialize (ar);
    if (ar.IsStoring ())
        ar << m_Width;
    else
        ar >> m_Width;
}

```



```

void CRectangle::Draw (CDC *PDC)
{
    CPen Pen, *PoldPen;

    // выбор пера/кисти
    Pen.CreatePen (PS_INSIDEFRAME, m_Width, m_Color);
    PoldPen = PDC->SelectObject (&Pen);
    PDC->SelectStockObject (NULL_BRUSH);

    // рисование фигуры
    PDC->Rectangle (m_X1, m_Y1, m_X2, m_Y2);

    // восстановление пера/кисти
    PDC->SelectObject (PoldPen);
}

IMPLEMENT_SERIAL (CRectangle, CFigure, 2);

CRectangle::CRectangle (int X1, int Y1, int X2, int Y2,
                        COLORREF Color, int Width)
{
    m_X1 = X1;
    m_Y1 = Y1;
    m_X2 = X2;
    m_Y2 = Y2;
    m_Color = Color;
    m_Width = Width;
}

void CRectangle::Serialize (CArchive &ar)
{
    CFigure::Serialize (ar);
    if (ar.IsStoring ())
        ar << m_Width;
    else
        ar >> m_Width;
}

void CRectangle::Draw (CDC *PDC)
{
    CPen Pen, *PoldPen;

    // выбор пера/кисти
    Pen.CreatePen (PS_INSIDEFRAME, m_Width, m_Color);
    PoldPen = PDC->SelectObject (&Pen);
    PDC->SelectStockObject (NULL_BRUSH);

    // рисование фигуры
    PDC->Rectangle (m_X1, m_Y1, m_X2, m_Y2);

    // восстановление пера/кисти
    PDC->SelectObject (PoldPen);
}

```

```

IMPLEMENT_SERIAL (CFRRectangle, CFigure, 2);

CFRRectangle::CFRRectangle (int X1, int Y1, int X2, int Y2,
                             COLORREF Color, int Width)
{
    m_X1 = X1;
    m_Y1 = Y1;
    m_X2 = X2;
    m_Y2 = Y2;
    m_Color = Color;
    m_Width = Width;
}

void CFRRectangle::Draw (CDC *PDC)
{
    CBrush Brush, *PoldBrush;
    CPen Pen, *PoldPen;

    // выбор пера/кисти
    Pen.CreatePen (PS_INSIDEFRAME, 1, m_Color);
    PoldPen = PDC->SelectObject (&Pen);
    Brush.CreateSolidBrush (m_Color);
    PoldBrush=PDC->SelectObject (&Brush);

    // рисование фигуры
    int SizeRound = (m_X2 - m_X1, m_Y2 - m_Y1) / 8;
    PDC->RoundRect (m_X1, m_Y1, m_X2, m_Y2, SizeRound, SizeRound);

    // восстановление пера/кисти
    PDC->SelectObject (PoldPen);
    PDC->SelectObject (PoldBrush);
}

IMPLEMENT_SERIAL (CRRectangle, CFigure, 2);

CRRectangle::CRRectangle (int X1, int Y1, int X2, int Y2,
                           COLORREF Color, int Width)
{
    m_X1 = X1;
    m_Y1 = Y1;
    m_X2 = X2;
    m_Y2 = Y2;
    m_Color = Color;
    m_Width = Width;
}

void CRRectangle::Serialize (CArchive &ar)
{
    CFigure::Serialize (ar);
    if (ar.IsStoring ())
        ar << m_Width;
    else

```

```

        ar << m_Width;
    }

void CRectangle::Draw (CDC *PDC)
{
    CPen Pen, *POldPen;

    // выбор пера/кисти
    Pen.CreatePen (PS_INSIDEFRAME, m_Width, m_Color);
    POldPen = PDC->SelectObject (&Pen);
    PDC->SelectStockObject (NULL_BRUSH);

    // рисование фигуры
    int SizeRound = (m_X2 - m_X1 + m_Y2 - m_Y1) / 8;
    PDC->RoundRect (m_X1, m_Y1, m_X2, m_Y2, SizeRound, SizeRound);

    // восстановление пера/кисти
    PDC->SelectObject (POldPen);
}

IMPLEMENT_SERIAL (CFCircle, CFigure, 2);

CFCircle::CFCircle (int X1, int Y1, int X2, int Y2,
                    COLORREF Color)
{
    m_X1 = X1;
    m_Y1 = Y1;
    m_X2 = X2;
    m_Y2 = Y2;
    m_Color = Color;
}

void CFCircle::Draw (CDC *PDC)
{
    CBrush Brush, *POldBrush;
    CPen Pen, *POldPen;

    // выбор пера/кисти
    Pen.CreatePen (PS_INSIDEFRAME, m_Width, m_Color);
    POldPen = PDC->SelectObject (&Pen);
    Brush.CreateSolidBrush (m_Color);
    POldBrush = PDC->SelectObject (&Brush);

    // рисование фигуры
    PDC->Ellipse (m_X1, m_Y1, m_X2, m_Y2);

    // восстановление пера/кисти
    PDC->SelectObject (POldPen);
    PDC->SelectObject (POldBrush);
}

IMPLEMENT_SERIAL (CCircle, CFigure, 2);

```

```

CCircle::CCircle (int X1, int Y1, int X2, int Y2,
                  COLORREF Color, int Width)
{
    m_X1 = X1;
    m_Y1 = Y1;
    m_X2 = X2;
    m_Y2 = Y2;
    m_Color = Color;
    m_Width = Width;
}

void CCircle::Serialize (CArchive &ar)
{
    CFigure::Serialize (ar);
    if (ar.IsStoring ())
        ar << m_Width;
    else
        ar >> m_Width;
}

void CCircle::Draw (CDC *PDC)
{
    CPen Pen, *PoldPen;

    // выбор пера/кисти
    Pen.CreatePen (PS_INSIDEFRAME, m_Width, m_Color);
    PoldPen = PDC->SelectObject (&Pen);
    PDC->SelectStockObject (NULL_BRUSH);

    // рисование фигуры
    PDC->Ellipse (m_X1, m_Y1, m_X2, m_Y2);

    // восстановление пера/кисти
    PDC->SelectObject (PoldPen);
}

```

Для каждого класса функция Draw использует средства рисования фигур, описанные в этой главе. Функция Draw при рисовании незакрашенной фигуры выбирает объект NULL\_BRUSH так, что внутренняя область фигуры не заполняется. Заметьте: функция Draw для замкнутых фигур инициализирует перо, используя стиль PS\_INSIDEFRAME (фигура будет находиться полностью внутри ограничивающего прямоугольника, созданного в окне представления). Классы рисования закругленных прямоугольников CRRectangle и CFRectangle вычисляют размер эллипса для рисования закругленных углов. Диаметр окружности, используемой для закругления углов, равен одной четверти среднего арифметического длин сторон прямоугольника.

```

int SizeRound = (m_X2 - m_X1 + m_Y2 - m_Y1) / 8;
PDC->RoundRect (m_X1, m_Y1, m_X2, m_Y2, SizeRound, SizeRound);

```

Теперь внесем в программу другие изменения, необходимые для рисования различных типов фигур. Удалите в файле ScratchBookDoc.h объявления функций AddLine и GetLine класса CscratchBookDoc. Замените имя функции CScratchBookDoc::GetNumLines на CscratchBookDoc::GetNumFigs, а имя переменной m\_LineArray на m\_FigArray (они используются для всех типов фигур, а не только линий). Измените также тип указателя, сохраняемого в CTypePtrArray с CLine на CFigure. Наконец, добавьте для функций AddFigure и GetFigure следующие объявления.

```

class CScratchBookDoc : public CDocument
{
protected:
    CTypedPtrArray<CObArray, CFigure*> m_FigArray;

public:
    int GetNumFigs ();
    void AddFigure (CFigure *PFigure);
    CFigure *GetFigure (int Index);

```

Далее в файле ScratchBookDoc.cpp удалите реализацию функций CScratchBookDoc::AddLine и CScratchBookDoc::GetLine, замените имя GetNumLines на GetNumFigs и добавьте для новых функций AddFigure и GetFigure реализации. Новые функции добавляют или получают указатели на объект класса CFigure вместо указателей на объект CLine. В отличие от функции AddLine функция AddFigure не вызывает операцию new для создания объекта. Эту задачу выполняет класс представления.

```

void CScratchBookDoc::AddFigure (CFigure *PFigure)
{
    m_FigArray.Add (PFigure);
    SetModifiedFlag ();
}

CFigure *CScratchBookDoc::GetFigure (int Index)
{
    if (Index < 0 || Index > m_FigArray.GetUpperBound ())
        return 0;
    return (CFigure *)m_FigArray.GetAt (Index);
}

int CScratchBookDoc::GetNumFigs ()
{
    return m_FigArray.GetSize ();
}

```

В файле ScratchBookDoc.cpp замените также все имена объекта m\_LineArray именем m\_FigArray. В файл ScratchBookView.h добавьте определение для переменной m\_PenDotted в начале определения класса CScratchBookView.

```

class CScratchBookView : public CScrollView
{
protected:
    CString m_ClassName;
    int m_Dragging;
    HCURSOR m_HCross;
    HCURSOR m_HArrow;
    CPoint m_PointOld;
    CPoint m_PointOrigin;
    CPen m_PenDotted;

```

В файле ScratchBookView.cpp добавьте в конструктор класса CScratchBookView инициализацию m\_PenDotted. Переменная m\_PenDotted используется для отображения временных пунктирных линий (вместо сплошных) при перетаскивании указателя мыши при рисовании фигуры.

```

CScratchBookView::CScratchBookView()
{
    // TODO: добавьте сюда собственный код конструктора

    m_Dragging = 0;
    m_HCross = AfxGetApp()->LoadStandardCursor (IDC_CROSS);
    m_HArrow = AfxGetApp()->LoadStandardCursor (IDC_ARROW);
    m_PenDotted.CreatePen (PS_DOT, 1, RGB (0, 0, 0));
}

```

Остальные изменения внесите в файл ScratchBookView.cpp. Измените функцию OnMouseMove. Добавленные строки удаляют прежнюю временную фигуру, а затем перерисовывают новую с текущей позиции мыши. Временная фигура отмечает место, где будет нарисована постоянная фигура, если пользователь отпустит кнопку мыши. Код начинается с выбора соответствующих инструментов рисования и установки необходимых атрибутов. Затем он переходит на соответствующую подпрограмму рисования для выбранного текущего инструмента рисования.

```

void CScratchBookView::OnMouseMove(UINT nFlags, CPoint point)
{
    // TODO: Вставьте сюда собственный код обработчика
    // или вызов стандартного

    int SizeRound;
    CClientDC ClientDC (this);
    OnPrepareDC (&ClientDC);
    ClientDC.DPtoLP (&point);

    if (!m_Dragging)
    {
        CSize ScrollSize = GetTotalSize ();
        CRect ScrollRect (0, 0, ScrollSize.cx, ScrollSize.cy);
        if (ScrollRect.PtInRect (point))
            ::SetCursor (m_HCross);
        else
            ::SetCursor (m_HArrow);
        return;
    }

    ClientDC.SetROP2 (R2_NOT);
    ClientDC.SelectObject (&m_PenDotted);
    ClientDC.SetBkMode (TRANSPARENT);
    ClientDC.SelectStockObject (NULL_BRUSH);

    switch (((CScratchBookApp *)AfxGetApp ()) ->m_CurrentTool)
    {
        case ID_TOOLS_LINE:
            ClientDC.MoveTo (m_PointOrigin);
            ClientDC.LineTo (m_PointOld);
            ClientDC.MoveTo (m_PointOrigin);
            ClientDC.LineTo (point);
            break;

        case ID_TOOLS_RECTANGLE:
        case ID_TOOLS_FRECTANGLE:

```

```

        ClientDC.Rectangle (m_PointOrigin.x, m_PointOrigin.y,
                           m_PointOld.x, m_PointOld.y);
        ClientDC.Rectangle (m_PointOrigin.x, m_PointOrigin.y,
                           point.x, point.y);
        break;

    case ID_TOOLS_RRECTANGLE:
    case ID_TOOLS_FRECTANGLE:
    {
        SizeRound = (abs(m_PointOld.x - m_PointOrigin.x)
+ abs (m_PointOld.y - m_PointOrigin.y)) / 8;
        ClientDC.RoundRect (m_PointOrigin.x,
m_PointOrigin.y, m_PointOld.x, m_PointOld.y, SizeRound,
SizeRound);
        SizeRound = (abs(point.x - m_PointOrigin.x)
+ abs (point.y - m_PointOrigin.y)) / 8;
        ClientDC.RoundRect (m_PointOrigin.x,
m_PointOrigin.y, point.x, point.y, SizeRound, SizeRound);

    case ID_TOOLS_CIRCLE:
    case ID_TOOLS_FCIRCLE:
        ClientDC.Ellipse (m_PointOrigin.x, m_PointOrigin.y,
                           m_PointOld.x, m_PointOld.y);
        ClientDC.Ellipse (m_PointOrigin.x, m_PointOrigin.y,
                           point.x, point.y);
        break;
    }
    m_PointOld = point;

    CScrollView::OnMouseMove(nFlags, point);
}
}

```

Внесите изменения в функцию OnLButtonUp.

```

void CScratchBookView::OnLButtonUp(UINT nFlags, CPoint point)
{
    // TODO: Вставьте сюда собственный код обработчика
    // или вызов стандартного
    int SizeRound;

    if (!m_Dragging) return;

    m_Dragging = 0;
    ::ReleaseCapture ();
    ::ClipCursor (NULL);

    CClientDC ClientDC(this);
    OnPrepareDC (&ClientDC);
    ClientDC.DPtoLP (&point);
    ClientDC.SetROP2 (R2_NOT);
    ClientDC.SelectObject (&m_PenDotted);
    ClientDC.SetBkMode (TRANSPARENT);
    ClientDC.SelectStockObject (NULL_BRUSH);
}

```

```

CScratchBookApp *PApp = (CScratchBookApp *)AfxGetApp ();
CFigure *PFigure;

switch (PApp->m_CurrentTool)
{
case ID_TOOLS_LINE:
    ClientDC.MoveTo (m_PointOrigin);
    ClientDC.LineTo (m_PointOld);
    PFigure = new CLine
        (m_PointOrigin.x, m_PointOrigin.y, point.x,
         point.y,
         PApp->m_CurrentColor, PApp->m_CurrentWidth);
    break;

case ID_TOOLS_RECTANGLE:
    ClientDC.Rectangle (m_PointOrigin.x, m_PointOrigin.y,
                        m_PointOld.x, m_PointOld.y);
    PFigure = new CRectangle
        (m_PointOrigin.x, m_PointOrigin.y, point.x,
         point.y,
         PApp->m_CurrentColor, PApp->m_CurrentWidth);
    break;

case ID_TOOLS_FRECTANGLE:
    ClientDC.Rectangle (m_PointOrigin.x, m_PointOrigin.y,
                        m_PointOld.x, m_PointOld.y);
    PFigure = new CFRectangle
        (m_PointOrigin.x, m_PointOrigin.y, point.x,
         point.y,
         PApp->m_CurrentColor, PApp->m_CurrentWidth);
    break;

case ID_TOOLS_RRECTANGLE:
    SizeRound = (abs(m_PointOld.x - m_PointOrigin.x) +
                 abs (m_PointOld.y - m_PointOrigin.y)) / 8;
    ClientDC.RoundRect (m_PointOrigin.x, m_PointOrigin.y,
                        m_PointOld.x, m_PointOld.y, SizeRound, SizeRound);
    PFigure = new CRRectangle
        (m_PointOrigin.x, m_PointOrigin.y, point.x,
         point.y,
         PApp->m_CurrentColor, PApp->m_CurrentWidth);
    break;

case ID_TOOLS_FRRECTANGLE:
    SizeRound = (abs(m_PointOld.x - m_PointOrigin.x) +
                 abs (m_PointOld.y - m_PointOrigin.y)) / 8;
    ClientDC.RoundRect (m_PointOrigin.x, m_PointOrigin.y,
                        m_PointOld.x, m_PointOld.y, SizeRound, SizeRound);
    PFigure = new CFRRectangle
        (m_PointOrigin.x, m_PointOrigin.y, point.x,
         point.y,
         PApp->m_CurrentColor, PApp->m_CurrentWidth);
    break;
}

```



```

case ID_TOOLS_CIRCLE:
    ClientDC.Ellipse (m_PointOrigin.x, m_PointOrigin.y,
        m_PointOld.x, m_PointOld.y);
    PFigure = new CCircle
        (m_PointOrigin.x, m_PointOrigin.y, point.x,
            point.y,
        PApp->m_CurrentColor, PApp->m_CurrentWidth);
    break;

case ID_TOOLS_FCIRCLE:
    ClientDC.Ellipse (m_PointOrigin.x, m_PointOrigin.y,
        m_PointOld.x, m_PointOld.y);
    PFigure = new CFCircle
        (m_PointOrigin.x, m_PointOrigin.y, point.x,
            point.y,
        PApp->m_CurrentColor);
    break;

    ClientDC.SetROP2 (R2_COPYPEN);
    PFigure->Draw (&ClientDC);
    CScratchBookDoc *PDoc = GetDocument ();
    PDoc->AddFigure (PFigure);

    PDoc->UpdateAllViews (this, 0, PFigure);

CScrollView::OnLButtonUp(nFlags, point);
}
}

```

Добавленный код также начинается с выбора необходимых инструментов и установки атрибутов рисования. Затем выполняется переход на фрагмент, соответствующий выбранному текущему инструменту рисования. Каждая подпрограмма удаляет временную фигуру, а затем создает объект корректного класса для сохранения и рисования новой постоянной фигуры, присваивая адрес объекта PFigure. После выполнения оператора switch функция OnLButtonUp вызывает функцию SetROP2, чтобы восстановить стандартный режим рисования, и использует указатель PFigure, чтобы вызвать функцию Draw для рисования постоянной фигуры. Поскольку функция Draw является виртуальной, ее вызов автоматически вызывает соответствующую версию Draw для текущего типа фигуры. Наконец, новый код вызывает функцию ScratchBookDoc::AddFigure, чтобы сохранить фигуру внутри класса документа, и функцию UpdateAllViews для перерисовки другого окна представления (в случае, если открыты два окна).

Внутри функции Draw необходимо изменить несколько строк. Измененный код использует новый класс CFigure вместо CLine. Он рисует фигуры в том порядке, в котором они добавлялись (предыдущий вариант рисовал их в обратном порядке), так что все фигуры переопределяются в том порядке, в котором они были нарисованы.

```

void CScratchBookView::OnDraw(CDC* pDC)
{
    CScratchBookDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    // TODO: вставьте сюда код прорисовки собственных данных
    CSize ScrollSize = GetTotalSize ();
    pDC->MoveTo (ScrollSize.cx, 0);
}

```

```

pDC->LineTo (ScrollSize.cx, ScrollSize.cy);
pDC->LineTo (0, ScrollSize.cy);

CRect ClipRect;
CRect DimRect;
CRect IntRect;
CFigure *PFigure;
pDC->GetClipBox (&ClipRect);

int NumFigs = pDoc->GetNumFigs ();
for (int Index = 0; Index < NumFigs; ++Index)
{
    PFigure = pDoc->GetFigure (Index);
    DimRect = PFigure->GetDimRect ();
    if (IntRect.IntersectRect (DimRect, ClipRect))
        PFigure->Draw(pDC);
}
}

```

## Текст программы ScratchBook

Теперь программу ScratchBook можно построить и выполнить. В листингах (19.9—19.16) приведен текст программы ScratchBook, созданной в этой главе.

---

### Листинг 19.9

```

// ScratchBook.h : главный заголовочный файл приложения ScratchBook
//

#pragma once

#ifdef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"          // основные символы

// CScratchBookApp:
// Смотрите реализацию этого класса в файле ScratchBook.cpp
//

class CScratchBookApp : public CWinApp
{
public:
    int m_CurrentWidth;
    UINT m_CurrentTool;
    COLORREF m_CurrentColor;
    UINT m_IdxCmd;

public:
    CScratchBookApp();

// Переопределения
public:
    virtual BOOL InitInstance();

```

```
// Реализация
afx_msg void OnAppAbout();
DECLARE_MESSAGE_MAP()
afx_msg void OnToolsLine();
afx_msg void OnToolsRectangle();
afx_msg void OnUpdateToolsRectangle(CCmdUI *pCmdUI);
afx_msg void OnToolsFrextangle();
afx_msg void OnUpdateToolsFrextangle(CCmdUI *pCmdUI);
afx_msg void OnToolsRrectangle();
afx_msg void OnUpdateToolsRrectangle(CCmdUI *pCmdUI);
afx_msg void OnToolsFrrectangle();
afx_msg void OnUpdateToolsFrrectangle(CCmdUI *pCmdUI);
afx_msg void OnToolsCircle();
afx_msg void OnUpdateToolsCircle(CCmdUI *pCmdUI);
afx_msg void OnToolsFcircle();
afx_msg void OnUpdateToolsFcircle(CCmdUI *pCmdUI);
afx_msg void OnLineSingle();
afx_msg void OnUpdateLineSingle(CCmdUI *pCmdUI);
afx_msg void OnLineDouble();
afx_msg void OnUpdateLineDouble(CCmdUI *pCmdUI);
afx_msg void OnLineTriple();
afx_msg void OnUpdateLineTriple(CCmdUI *pCmdUI);
afx_msg void OnUpdateToolsLine(CCmdUI *pCmdUI);
afx_msg void OnColorBlack();
afx_msg void OnUpdateColorBlack(CCmdUI *pCmdUI);
afx_msg void OnColorWhite();
afx_msg void OnUpdateColorWhite(CCmdUI *pCmdUI);
afx_msg void OnColorRed();
afx_msg void OnUpdateColorRed(CCmdUI *pCmdUI);
afx_msg void OnColorGreen();
afx_msg void OnUpdateColorGreen(CCmdUI *pCmdUI);
afx_msg void OnColorBlue();
afx_msg void OnUpdateColorBlue(CCmdUI *pCmdUI);
afx_msg void OnColorYellow();
afx_msg void OnUpdateColorYellow(CCmdUI *pCmdUI);
afx_msg void OnColorCyan();
afx_msg void OnUpdateColorCyan(CCmdUI *pCmdUI);
afx_msg void OnUpdateColorMagenta(CCmdUI *pCmdUI);
afx_msg void OnColorCustom();
afx_msg void OnUpdateColorCustom(CCmdUI *pCmdUI);
void OnColorMagenta(void);
};
```

---

## Листинг 19.10

```
// ScratchBook.cpp : Определяет поведение классов приложения
//

#include "stdafx.h"
#include "ScratchBook.h"
#include "MainFrm.h"

#include "ScratchBookDoc.h"
#include "ScratchBookView.h"

#ifdef _DEBUG
```

```

#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// CScratchBookApp

BEGIN_MESSAGE_MAP(CScratchBookApp, CWinApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
    // Стандартные команды работы с файлами документов
    ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
    ON_COMMAND(ID_TOOLS_LINE, OnToolsLine)
    ON_COMMAND(ID_TOOLS_RECTANGLE, OnToolsRectangle)
    ON_UPDATE_COMMAND_UI(ID_TOOLS_RECTANGLE, OnUpdateToolsRectangle)
    ON_COMMAND(ID_TOOLS_FRECTANGLE, OnToolsFrectangle)
    ON_UPDATE_COMMAND_UI(ID_TOOLS_FRECTANGLE,
        OnUpdateToolsFrectangle)
    ON_COMMAND(ID_TOOLS_RRECTANGLE, OnToolsRrectangle)
    ON_UPDATE_COMMAND_UI(ID_TOOLS_RRECTANGLE,
        OnUpdateToolsRrectangle)
    ON_COMMAND(ID_TOOLS_FRRECTANGLE, OnToolsFrrectangle)
    ON_UPDATE_COMMAND_UI(ID_TOOLS_FRRECTANGLE,
        OnUpdateToolsFrrectangle)
    ON_COMMAND(ID_TOOLS_CIRCLE, OnToolsCircle)
    ON_UPDATE_COMMAND_UI(ID_TOOLS_CIRCLE, OnUpdateToolsCircle)
    ON_COMMAND(ID_TOOLS_FCIRCLE, OnToolsFcircle)
    ON_UPDATE_COMMAND_UI(ID_TOOLS_FCIRCLE, OnUpdateToolsFcircle)
    ON_COMMAND(ID_LINE_SINGLE, OnLineSingle)
    ON_UPDATE_COMMAND_UI(ID_LINE_SINGLE, OnUpdateLineSingle)
    ON_COMMAND(ID_LINE_DOUBLE, OnLineDouble)
    ON_UPDATE_COMMAND_UI(ID_LINE_DOUBLE, OnUpdateLineDouble)
    ON_COMMAND(ID_LINE_TRIPLE, OnLineTriple)
    ON_UPDATE_COMMAND_UI(ID_LINE_TRIPLE, OnUpdateLineTriple)
    ON_UPDATE_COMMAND_UI(ID_TOOLS_LINE, OnUpdateToolsLine)
    ON_COMMAND(ID_COLOR_BLACK, OnColorBlack)
    ON_UPDATE_COMMAND_UI(ID_COLOR_BLACK, OnUpdateColorBlack)
    ON_COMMAND(ID_COLOR_WHITE, OnColorWhite)
    ON_UPDATE_COMMAND_UI(ID_COLOR_WHITE, OnUpdateColorWhite)
    ON_COMMAND(ID_COLOR_RED, OnColorRed)
    ON_UPDATE_COMMAND_UI(ID_COLOR_RED, OnUpdateColorRed)
    ON_COMMAND(ID_COLOR_GREEN, OnColorGreen)
    ON_UPDATE_COMMAND_UI(ID_COLOR_GREEN, OnUpdateColorGreen)
    ON_COMMAND(ID_COLOR_BLUE, OnColorBlue)
    ON_UPDATE_COMMAND_UI(ID_COLOR_BLUE, OnUpdateColorBlue)
    ON_COMMAND(ID_COLOR_YELLOW, OnColorYellow)
    ON_UPDATE_COMMAND_UI(ID_COLOR_YELLOW, OnUpdateColorYellow)
    ON_COMMAND(ID_COLOR_CYAN, OnColorCyan)
    ON_UPDATE_COMMAND_UI(ID_COLOR_CYAN, OnUpdateColorCyan)
    ON_COMMAND(ID_COLOR_MAGENTA, OnColorMagenta)
    ON_UPDATE_COMMAND_UI(ID_COLOR_MAGENTA, OnUpdateColorMagenta)
    ON_COMMAND(ID_COLOR_CUSTOM, OnColorCustom)
    ON_UPDATE_COMMAND_UI(ID_COLOR_CUSTOM, OnUpdateColorCustom)
END_MESSAGE_MAP()

```

```

// Конструктор CScratchBookApp

CScratchBookApp::CScratchBookApp()
{
    // TODO: добавьте сюда код конструктора
    // Поместите весь существенный код инициализации
    // в функцию InitInstance

    m_CurrentColor = RGB (0, 0, 0);
    m_CurrentWidth = 1;
    m_CurrentTool=ID_TOOLS_LINE;
    m_IdxCOLORCmd = ID_COLOR_BLACK;
}

// Единственный объект класса CScratchBookApp

CScratchBookApp theApp;

// Инициализация CScratchBookApp

BOOL CScratchBookApp::InitInstance()
{
    CWinApp::InitInstance();

    // Стандартная инициализация.
    // Если вы не используете какие-то из предоставленных
    // возможностей и хотите уменьшить размер конечного
    // исполняемого модуля, удалите из последующего кода
    // отдельные команды инициализации элементов, которые
    // вам не нужны.
    // Измените ключ, под которым ваши установки хранятся в реестре
    // TODO: Измените эту строку на что-нибудь подходящее,
    // например, на имя вашей компании или организации
    SetRegistryKey(_T("Local AppWizard-Generated Applications"));
    LoadStdProfileSettings(4); // Загрузка стандартных установок
                               // из INI-файла (включая MRU)
    // Регистрация шаблона документа приложения. Шаблоны
    // документов служат связью между документами, окнами
    // документов и окнами приложений
    CSingleDocTemplate* pDocTemplate;
    pDocTemplate = new CSingleDocTemplate(
        IDR_MAINFRAME,
        RUNTIME_CLASS(CScratchBookDoc),
        RUNTIME_CLASS(CMainFrame), // главное окно
                                   // SDI-приложения
        RUNTIME_CLASS(CScratchBookView));
    AddDocTemplate(pDocTemplate);

    EnableShellOpen ();
    RegisterShellFileTypes ();

    // Просмотр командной строки для обнаружения стандартных
    // команд оболочки, DDE, открытия файлов
    CCommandLineInfo cmdInfo;
    ParseCommandLine(cmdInfo);
    // Выполнение команд, указанных в командной строке.

```

```

        // Вернет FALSE, если приложение было запущено с
        // /RegServer, /Register, /Unregserver или /Unregister.
        if (!ProcessShellCommand(cmdInfo))
            return FALSE;
        // Показ и обновление единственного проинициализированного окна
        m_pMainWnd->ShowWindow(SW_SHOW);
        m_pMainWnd->UpdateWindow();

        m_pMainWnd->DragAcceptFiles ();

        return TRUE;
    }

    // CAboutDlg диалог, используемый в App About

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

    // Данные для диалога
    enum { IDD = IDD_ABOUTBOX };

protected:
    virtual void DoDataExchange(CDataExchange* pDX);
    // Поддержка DDX/DDV

    // Реализация
protected:
    DECLARE_MESSAGE_MAP()
public:
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)

END_MESSAGE_MAP()

// Команда приложения для выполнения диалога
void CScratchBookApp::OnAppAbout()
{
    CAboutDlg aboutDlg;
    aboutDlg.DoModal();
}

// Обработчики сообщений класса CScratchBookApp

void CScratchBookApp::OnToolsLine()

```

```

{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentTool = ID_TOOLS_LINE;
}

void CScratchBookApp::OnToolsRectangle()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentTool = ID_TOOLS_RECTANGLE;
}

void CScratchBookApp::OnUpdateToolsRectangle(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck
        (m_CurrentTool == ID_TOOLS_RECTANGLE ? 1 : 0);
}

void CScratchBookApp::OnToolsFrextangle()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentTool = ID_TOOLS_FRECTANGLE;
}

void CScratchBookApp::OnUpdateToolsFrextangle(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck
        (m_CurrentTool == ID_TOOLS_FRECTANGLE ? 1 : 0);
}

void CScratchBookApp::OnToolsRrectangle()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentTool = ID_TOOLS_RRECTANGLE;
}

void CScratchBookApp::OnUpdateToolsRrectangle(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck
        (m_CurrentTool == ID_TOOLS_RRECTANGLE ? 1 : 0);
}

void CScratchBookApp::OnToolsFrrectangle()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentTool = ID_TOOLS_FRRECTANGLE;
}

void CScratchBookApp::OnUpdateToolsFrrectangle(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck
        (m_CurrentTool == ID_TOOLS_FRRECTANGLE ? 1 : 0);
}

```

```

void CScratchBookApp::OnToolsCircle()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentTool = ID_TOOLS_CIRCLE;
}

void CScratchBookApp::OnUpdateToolsCircle(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck
        (m_CurrentTool == ID_TOOLS_CIRCLE ? 1 : 0);
}

void CScratchBookApp::OnToolsFcicle()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentTool = ID_TOOLS_FCIRCLE;
}

void CScratchBookApp::OnUpdateToolsFcicle(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck
        (m_CurrentTool==ID_TOOLS_FCIRCLE ? 1 : 0);
}

void CScratchBookApp::OnLineSingle()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentWidth = 1;
}

void CScratchBookApp::OnUpdateLineSingle(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck(m_CurrentWidth == 1 ? 1 : 0);
}

void CScratchBookApp::OnLineDouble()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentWidth = 2;
}

void CScratchBookApp::OnUpdateLineDouble(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck(m_CurrentWidth == 2 ? 1 : 0);
}

void CScratchBookApp::OnLineTriple()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentWidth = 3;
}

```



```

void CScratchBookApp::OnUpdateLineTriple(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck(m_CurrentWidth == 3 ? 1 : 0);
}

void CScratchBookApp::OnUpdateToolsLine(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck(m_CurrentTool == ID_TOOLS_LINE ? 1 : 0);
}

void CScratchBookApp::OnColorBlack()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentColor = RGB (0, 0, 0);
    m_IdxColorCmd = ID_COLOR_BLACK;
}

void CScratchBookApp::OnUpdateColorBlack(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck (m_IdxColorCmd == ID_COLOR_BLACK ? 1 : 0);
}

void CScratchBookApp::OnColorWhite()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentColor = RGB (255, 255, 255);
    m_IdxColorCmd = ID_COLOR_WHITE;
}

void CScratchBookApp::OnUpdateColorWhite(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck(m_IdxColorCmd == ID_COLOR_WHITE ? 1 : 0);
}

void CScratchBookApp::OnColorRed()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentColor = RGB (255, 0, 0);
    m_IdxColorCmd = ID_COLOR_RED;
}

void CScratchBookApp::OnUpdateColorRed(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI ->SetCheck(m_IdxColorCmd == ID_COLOR_RED ? 1 : 0);
}

void CScratchBookApp::OnColorGreen()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentColor = RGB (0, 255, 0);
    m_IdxColorCmd = ID_COLOR_GREEN;
}

```

```

void CScratchBookApp::OnUpdateColorGreen(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck(m_IdxColorCmd == ID_COLOR_GREEN ? 1 : 0);
}

void CScratchBookApp::OnColorBlue()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentColor = RGB (0, 0, 255);
    m_IdxColorCmd = ID_COLOR_BLUE;
}

void CScratchBookApp::OnUpdateColorBlue(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck(m_IdxColorCmd == ID_COLOR_BLUE ? 1 : 0);
}

void CScratchBookApp::OnColorYellow()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentColor = RGB (255, 255, 0);
    m_IdxColorCmd = ID_COLOR_YELLOW;
}

void CScratchBookApp::OnUpdateColorYellow(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck (m_IdxColorCmd == ID_COLOR_YELLOW ? 1 : 0);
}

void CScratchBookApp::OnColorCyan()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentColor = RGB (0, 255, 255);
    m_IdxColorCmd = ID_COLOR_CYAN;
}

void CScratchBookApp::OnUpdateColorCyan(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck(m_IdxColorCmd == ID_COLOR_CYAN ? 1 : 0);
}

void CScratchBookApp::OnColorMagenta()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentColor = RGB (255, 0, 255);
    m_IdxColorCmd = ID_COLOR_MAGENTA;
}

void CScratchBookApp::OnUpdateColorMagenta(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck(m_IdxColorCmd == ID_COLOR_MAGENTA ? 1 : 0);
}

```

```

void CScratchBookApp::OnColorCustom()
{
    // TODO: Добавьте сюда собственный код обработчика
    CColorDialog ColorDialog;

    if (ColorDialog.DoModal () == IDOK)
    {
        m_CurrentColor = ColorDialog.GetColor ();
        m_IdxColorCmd = ID_COLOR_CUSTOM;
    }
}

void CScratchBookApp::OnUpdateColorCustom(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI ->SetCheck(m_IdxColorCmd == ID_COLOR_CUSTOM ? 1 : 0);
}

```

---

### Листинг 19.11

```

// ScratchBookDoc.h : интерфейс класса CScratchBookDoc
//

```

```

#pragma once

```

```

class CFigure: public CObject
{
protected:
    COLORREF m_Color;
    DWORD m_X1, m_Y1, m_X2, m_Y2;
    CFigure () {}
    DECLARE_SERIAL (CFigure)

public:
    virtual void Draw (CDC *PDC) {}
    CRect GetDimRect ();
    virtual void Serialize (CArchive& ar);
};

```

```

class CLine: public CFigure
{
protected:
    DWORD m_Width;
    CLine () {}
    DECLARE_SERIAL (CLine)

public:
    CLine (int X1, int Y1, int X2, int Y2,
           COLORREF Color, int Width);
    virtual void Draw (CDC *PDC);
    virtual void Serialize (CArchive& ar);
};

```

```

class CRectangle: public CFigure
{
protected:

```

```

        DWORD m_Width;
        CRectangle () {}
        DECLARE_SERIAL (CRectangle)

public:
    CRectangle (int X1, int Y1, int X2, int Y2,
                COLORREF Color, int Width);
    virtual void Draw (CDC *PDC);
    virtual void Serialize (CArchive& ar);
};

class CRectangle: public CFigure
{
protected:
    DWORD m_Width;
    CRectangle () {}
    DECLARE_SERIAL (CRectangle)

public:
    CRectangle (int X1, int Y1, int X2, int Y2,
                COLORREF Color, int Width);
    virtual void Draw (CDC *PDC);
    virtual void Serialize (CArchive& ar);
};

class CRRectangle: public CFigure
{
protected:
    DWORD m_Width;
    CRRectangle () {}
    DECLARE_SERIAL (CRRectangle)

public:
    CRRectangle (int X1, int Y1, int X2, int Y2,
                 COLORREF Color, int Width);
    virtual void Draw (CDC *PDC);
    virtual void Serialize (CArchive& ar);
};

class CFRRectangle: public CFigure
{
protected:
    DWORD m_Width;
    CFRRectangle () {}
    DECLARE_SERIAL (CFRRectangle)

public:
    CFRRectangle (int X1, int Y1, int X2, int Y2,
                  COLORREF Color, int Width);
    virtual void Draw (CDC *PDC);
    virtual void Serialize (CArchive& ar);
};

class CCircle: public CFigure
{
protected:

```

```

        DWORD m_Width;
        CCircle () {}
        DECLARE_SERIAL (CCircle)

public:
    CCircle (int X1, int Y1, int X2, int Y2,
             COLORREF Color, int Width);
    virtual void Draw (CDC *PDC);
    virtual void Serialize (CArchive& ar);
};

class CFCircle: public CFigure
{
protected:
    DWORD m_Width;
    CFCircle () {}
    DECLARE_SERIAL (CFCircle)

public:
    CFCircle (int X1, int Y1, int X2, int Y2, COLORREF Color);
    virtual void Draw (CDC *PDC);
    virtual void Serialize (CArchive& ar);
};

class CScratchBookDoc : public CDocument
{
protected:
    CTypedPtrArray<CObArray, CFigure*> m_FigArray;

public:
    int GetNumFigs ();
    void AddFigure (CFigure *PFigure);
    CFigure *GetFigure (int Index);

protected: // используется только для сериализации
    CScratchBookDoc();
    DECLARE_DYNCREATE(CScratchBookDoc)

// Атрибуты
public:

// Операции
public:

// Переопределения
    public:
        virtual BOOL OnNewDocument();
        virtual void Serialize(CArchive& ar);

// Реализация
public:
    virtual ~CScratchBookDoc();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif
}

```

```
protected:

// Сгенерированные функции карты сообщений
protected:
    DECLARE_MESSAGE_MAP()
public:
    virtual void DeleteContents();
    afx_msg void On57633();
    afx_msg void OnUpdate57633(CCmdUI *pCmdUI);
    afx_msg void On57643();
    afx_msg void OnUpdate57643(CCmdUI *pCmdUI);
};
```

---

## Листинг 19.12

```
// ScratchBookDoc.cpp : реализация класса CScratchBookDoc
//

#include "stdafx.h"
#include "ScratchBook.h"

#include "ScratchBookDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// CScratchBookDoc

IMPLEMENT_DYNCREATE(CScratchBookDoc, CDocument)

BEGIN_MESSAGE_MAP(CScratchBookDoc, CDocument)
    ON_COMMAND(57633, On57633)
    ON_UPDATE_COMMAND_UI(57633, OnUpdate57633)
    ON_COMMAND(57643, On57643)
    ON_UPDATE_COMMAND_UI(57643, OnUpdate57643)
END_MESSAGE_MAP()

// Конструктор/деструктор класса CScratchBookDoc

CScratchBookDoc::CScratchBookDoc()
{
    // TODO: добавьте сюда код одноразового конструктора
}

CScratchBookDoc::~CScratchBookDoc()
{
}

BOOL CScratchBookDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;
```

```

        // TODO: добавьте сюда код повторной инициализации
        // {SDI-документы будут использовать этот документ
        // многократно}

        return TRUE;
    }

// Сериализация CScratchBookDoc

void CScratchBookDoc::Serialize(CArchive& ar)
{
    if (ar.IsStoring())
    {
        // TODO: добавьте сюда код сохранения
        m_FigArray.Serialize(ar);
    }
    else
    {
        // TODO: добавьте сюда код загрузки
        m_FigArray.Serialize(ar);
    }
}

// Диагностика класса CScratchBookDoc

#ifdef _DEBUG
void CScratchBookDoc::AssertValid() const
{
    CDocument::AssertValid();
}

void CScratchBookDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}
#endif // _DEBUG

// Команды класса CScratchBookDoc

CFigure *CScratchBookDoc::AddFigure (int X1, int Y1, int X2, int Y2)
{
    CFigure *PFigure = new CFigure (X1, Y1, X2, Y2);
    m_FigArray.Add (PFigure);
    SetModifiedFlag( );
    return PFigure;
}

void CScratchBookDoc::DeleteContents()
{
    // TODO: Добавьте сюда собственный код или
    // вызов базового класса
    int Index = m_FigArray.GetSize ();
    while (Index--)
        delete m_FigArray.GetAt (Index);
    m_FigArray.RemoveAll ();
}

```

```

        CDocument::DeleteContents();
    }

void CScratchBookDoc::On57633()
{
    // TODO: Добавьте сюда собственный код обработчика
    DeleteContents ();
    UpdateAllViews (0);
    SetModifiedFlag ( );
}

void CScratchBookDoc::OnUpdate57633(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика

    pCmdUI->Enable (m_FigArray.GetSize ());
}

void CScratchBookDoc::On57643()
{
    // TODO: Добавьте сюда собственный код обработчика
    int Index = m_FigArray.GetUpperBound ();
    if (Index > -1)
    {
        delete m_FigArray.GetAt (Index);
        m_FigArray.RemoveAt (Index);
    }
    UpdateAllViews (0);
    SetModifiedFlag ();
}

void CScratchBookDoc::OnUpdate57643(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->Enable (m_FigArray.GetSize ());
}

// реализация классов фигур

IMPLEMENT_SERIAL (CFigure, CObject, 2)

CRect CFigure::GetDimRect ()
{
    return CRect
        (min (m_X1, m_X2), min (m_Y1, m_Y2),
         (max (m_X1, m_X2))+1, (max (m_Y1, m_Y2))+1);
}

void CFigure::Serialize (CArchive& ar)
{
    if (ar.IsStoring ())
        ar << m_X1 << m_Y1 << m_X2 << m_Y2 << m_Color;
    else

```



```

        ar >> m_X1 >> m_Y1 >> m_X2 >> m_Y2 >> m_Color;
    }

IMPLEMENT_SERIAL (CLine, CFigure, 2);

CLine::CLine (int X1, int Y1, int X2, int Y2,
              COLORREF Color, int Width)
{
    m_X1 = X1;
    m_Y1 = Y1;
    m_X2 = X2;
    m_Y2 = Y2;
    m_Color = Color;
    m_Width = Width;
}

void CLine::Serialize (CArchive &ar)
{
    CFigure::Serialize (ar);
    if (ar.IsStoring ())
        ar << m_Width;
    else
        ar >> m_Width;
}

void CLine::Draw (CDC *PDC)
{
    CPen Pen, *POldPen;

    // выбор пера/кисти
    Pen.CreatePen (PS_SOLID, m_Width, m_Color);
    POldPen = PDC->SelectObject (&Pen);

    // рисование фигуры
    PDC->MoveTo (m_X1, m_Y1);
    PDC->LineTo (m_X2, m_Y2);

    // восстановление пера/кисти
    PDC->SelectObject (POldPen);
}

IMPLEMENT_SERIAL (CRectangle, CFigure, 2);

CRectangle::CRectangle (int X1, int Y1, int X2, int Y2,
                        COLORREF Color, int Width)
{
    m_X1 = X1;
    m_Y1 = Y1;
    m_X2 = X2;
    m_Y2 = Y2;
    m_Color = Color;
    m_Width = Width;
}

void CRectangle::Serialize (CArchive &ar)
{

```

```

        CFigure::Serialize (ar);
        if (ar.IsStoring ())
            ar << m_Width;
        else
            ar >> m_Width;
    }

void CRectangle::Draw (CDC *PDC)
{
    CPen Pen, *POldPen;

    // выбор пера/кисти
    Pen.CreatePen (PS_INSIDEFRAME, m_Width, m_Color);
    POldPen = PDC->SelectObject (&Pen);
    PDC->SelectStockObject (NULL_BRUSH);

    // рисование фигуры
    PDC->Rectangle (m_X1, m_Y1, m_X2, m_Y2);

    // восстановление пера/кисти
    PDC->SelectObject (POldPen);
}

IMPLEMENT_SERIAL (CRectangle, CFigure, 2);

CRectangle::CRectangle (int X1, int Y1, int X2, int Y2,
                        COLORREF Color, int Width)
{
    m_X1 = X1;
    m_Y1 = Y1;
    m_X2 = X2;
    m_Y2 = Y2;
    m_Color = Color;
    m_Width = Width;
}

void CRectangle::Serialize (CArchive &ar)
{
    CFigure::Serialize (ar);
    if (ar.IsStoring ())
        ar << m_Width;
    else
        ar >> m_Width;
}

void CRectangle::Draw (CDC *PDC)
{
    CPen Pen, *POldPen;

    // выбор пера/кисти
    Pen.CreatePen (PS_INSIDEFRAME, m_Width, m_Color);
    POldPen = PDC->SelectObject (&Pen);
    PDC->SelectStockObject (NULL_BRUSH);

    // рисование фигуры

```

```

        PDC->Rectangle (m_X1, m_Y1, m_X2, m_Y2);

        // восстановление пера/кисти
        PDC->SelectObject (PoldPen);
    }

IMPLEMENT_SERIAL (CFRRectangle, CFigure, 2);

CFRRectangle::CFRRectangle (int X1, int Y1, int X2, int Y2,
                             COLORREF Color, int Width)
{
    m_X1 = X1;
    m_Y1 = Y1;
    m_X2 = X2;
    m_Y2 = Y2;
    m_Color = Color;
    m_Width = Width;
}

void CFRRectangle::Draw (CDC *PDC)
{
    CBrush Brush, *PoldBrush;
    CPen Pen, *PoldPen;

    // выбор пера/кисти
    Pen.CreatePen (PS_INSIDEFRAME, 1, m_Color);
    PoldPen = PDC->SelectObject (&Pen);
    Brush.CreateSolidBrush (m_Color);
    PoldBrush=PDC->SelectObject (&Brush);

    // рисование фигуры
    int SizeRound = (m_X2 - m_X1, m_Y2 - m_Y1) / 8;
    PDC->RoundRect (m_X1, m_Y1, m_X2, m_Y2, SizeRound, SizeRound);

    // восстановление пера/кисти
    PDC->SelectObject (PoldPen);
    PDC->SelectObject (PoldBrush);
}

IMPLEMENT_SERIAL (CRRectangle, CFigure, 2);

CRRectangle::CRRectangle (int X1, int Y1, int X2, int Y2,
                           COLORREF Color, int Width)
{
    m_X1 = X1;
    m_Y1 = Y1;
    m_X2 = X2;
    m_Y2 = Y2;
    m_Color = Color;
    m_Width = Width;
}

void CRRectangle::Serialize (CArchive &ar)
{
    CFigure::Serialize (ar);
    if (ar.IsStoring ())

```

```

        ar << m_Width;
    else
        ar << m_Width;
}

void CRRectangle::Draw (CDC *PDC)
{
    CPen Pen, *PoldPen;

    // выбор пера/кисти
    Pen.CreatePen (PS_INSIDEFRAME, m_Width, m_Color);
    PoldPen = PDC->SelectObject (&Pen);
    PDC->SelectStockObject (NULL_BRUSH);

    // рисование фигуры
    int SizeRound = (m_X2 - m_X1 + m_Y2 - m_Y1) / 8;
    PDC->RoundRect (m_X1, m_Y1, m_X2, m_Y2, SizeRound, SizeRound);

    // восстановление пера/кисти
    PDC->SelectObject (PoldPen);
}

IMPLEMENT_SERIAL (CFCircle, CFigure, 2);

CFCircle::CFCircle (int X1, int Y1, int X2, int Y2, COLORREF Color)
{
    m_X1 = X1;
    m_Y1 = Y1;
    m_X2 = X2;
    m_Y2 = Y2;
    m_Color = Color;
}

void CFCircle::Draw (CDC *PDC)
{
    CBrush Brush, *PoldBrush;
    CPen Pen, *PoldPen;

    // выбор пера/кисти
    Pen.CreatePen (PS_INSIDEFRAME, m_Width, m_Color);
    PoldPen = PDC->SelectObject (&Pen);
    Brush.CreateSolidBrush (m_Color);
    PoldBrush = PDC->SelectObject (&Brush);

    // рисование фигуры
    PDC->Ellipse (m_X1, m_Y1, m_X2, m_Y2);

    // восстановление пера/кисти
    PDC->SelectObject (PoldPen);
    PDC->SelectObject (PoldBrush);
}

IMPLEMENT_SERIAL (CCircle, CFigure, 2);

CCircle::CCircle (int X1, int Y1, int X2, int Y2,
                  COLORREF Color, int Width)
{
    m_X1 = X1;

```

```

        m_Y1 = Y1;
        m_X2 = X2;
        m_Y2 = Y2;
        m_Color = Color;
        m_Width = Width;
    }

void CCircle::Serialize (CArchive &ar)
{
    CFigure::Serialize (ar);
    if (ar.IsStoring ())
        ar << m_Width;
    else
        ar >> m_Width;
}

void CCircle::Draw (CDC *PDC)
{
    CPen Pen, *PoldPen;

    // выбор пера/кисти
    Pen.CreatePen (PS_INSIDEFRAME, m_Width, m_Color);
    PoldPen = PDC->SelectObject (&Pen);
    PDC->SelectStockObject (NULL_BRUSH);

    // рисование фигуры
    PDC->Ellipse (m_X1, m_Y1, m_X2, m_Y2);

    // восстановление пера/кисти
    PDC->SelectObject (PoldPen);
}

void CScratchBookDoc::AddFigure (CFigure *PFigure)
{
    m_FigArray.Add (PFigure);
    SetModifiedFlag ();
}

CFigure *CScratchBookDoc::GetFigure (int Index)
{
    if (Index < 0 || Index > m_FigArray.GetUpperBound ())
        return 0;
    return (CFigure *)m_FigArray.GetAt (Index);
}

int CScratchBookDoc::GetNumFigs ()
{
    return m_FigArray.GetSize ();
}

```

---

### Листинг 19.13

```

// ScratchBookView.h : интерфейс класса CScratchBookView
//

#pragma once

class CScratchBookView : public CScrollView

```

```

{
protected:
    CString m_ClassName;
    int m_Dragging;
    HCURSOR m_HCross;
    HCURSOR m_HArrow;
    CPoint m_PointOld;
    CPoint m_PointOrigin;
    CPen m_PenDotted;

protected: // используется только для сериализации
    CScratchBookView();
    DECLARE_DYNCREATE(CScratchBookView)

// Атрибуты
public:
    CScratchBookDoc* GetDocument() const;

// Операции
public:

// Переопределения
public:
    virtual void OnDraw(CDC* pDC);
    // переопределена для прорисовки этого вида
virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
protected:

// Реализация
public:
    virtual ~CScratchBookView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif
protected:

// Сгенерированные функции карты сообщений
protected:
    DECLARE_MESSAGE_MAP()
public:
    afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
    afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
    afx_msg void OnMouseMove(UINT nFlags, CPoint point);
    virtual void OnInitialUpdate();
protected:
    virtual void OnUpdate(CView* /*pSender*/, LPARAM /*lHint*/,
                        CObject* /*pHint*/);
};

#ifdef _DEBUG // отладочная версия в файле ScratchBookView.cpp
inline CScratchBookDoc* CScratchBookView::GetDocument() const
    { return (CScratchBookDoc*)m_pDocument; }
#endif

```

---

## Листинг 19.14

```
// ScratchBookView.cpp : реализация класса CScratchBookView
//

#include "stdafx.h"
#include "ScratchBook.h"

#include "ScratchBookDoc.h"
#include "ScratchBookView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// CScratchBookView

IMPLEMENT_DYNCREATE(CScratchBookView, CScrollView)

BEGIN_MESSAGE_MAP(CScratchBookView, CScrollView)
    ON_WM_LBUTTONDOWN()
    ON_WM_LBUTTONUP()
    ON_WM_MOUSEMOVE()
END_MESSAGE_MAP()

// Конструктор/деструктор класса CScratchBookView

CScratchBookView::CScratchBookView()
{
    // TODO: добавьте сюда собственный код конструктора

    m_Dragging = 0;
    m_HCross = AfxGetApp()->LoadStandardCursor (IDC_CROSS);
    m_HArrow = AfxGetApp()->LoadStandardCursor (IDC_ARROW);
    m_PenDotted.CreatePen (PS_DOT, 1, RGB (0, 0, 0));
}

CScratchBookView::~CScratchBookView()
{
}

BOOL CScratchBookView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Модифицируйте класс или стили окна,
    // добавляя и изменяя поля структуры cs

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

```

        return CScrollView::PreCreateWindow(cs);
    }

// Прорисовка CScratchBookView

void CScratchBookView::OnDraw(CDC* pDC)
{
    CScratchBookDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    // TODO: вставьте сюда код прорисовки собственных данных
    CSize ScrollSize = GetTotalSize ();
    pDC->MoveTo (ScrollSize.cx, 0);
    pDC->LineTo (ScrollSize.cx, ScrollSize.cy);
    pDC->LineTo (0, ScrollSize.cy);

    CRect ClipRect;
    CRect DimRect;
    CRect IntRect;
    CFigure *PFigure;
    pDC->GetClipBox (&ClipRect);

    int NumFigs = pDoc->GetNumFigs ();
    for (int Index = 0; Index < NumFigs; ++Index)
    {
        PFigure = pDoc->GetFigure (Index);
        DimRect = PFigure->GetDimRect ();
        if (IntRect.IntersectRect (DimRect, ClipRect))
            PFigure->Draw(pDC);
    }
}

// Диагностика класса CScratchBookView

#ifdef _DEBUG
void CScratchBookView::AssertValid() const
{
    CScrollView::AssertValid();
}

void CScratchBookView::Dump(CDumpContext& dc) const
{
    CScrollView::Dump(dc);
}

CScratchBookDoc* CScratchBookView::GetDocument() const
// не отладочная версия встроена
{
    ASSERT(m_pDocument->IsKindOf (RUNTIME_CLASS (CScratchBookDoc)));
    return (CScratchBookDoc*)m_pDocument;
}
#endif //_DEBUG

// Обработки сообщений класса CScratchBookView

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

```



```

{
    // TODO: Вставьте сюда собственный код обработчика
    // или вызов стандартного
    CClientDC ClientDC (this);
    OnPrepareDC (&ClientDC);
    ClientDC.DPtoLP (&point);

    // проверка нахождения курсора внутри области
    // окна представления
    CSize ScrollSize = GetTotalSize ();
    CRect ScrollRect (0, 0, ScrollSize.cx, ScrollSize.cy);
    if (!ScrollRect.PtInRect (point))
        return;

    // сохранение позиции курсора, захват мыши и ,
    // установка флага перемещения
    m_PointOrigin = point;
    m_PointOld = point;
    SetCapture ();
    m_Dragging = 1;

    // ограничение перемещений курсора мыши
    ClientDC.LPtoDP (&ScrollRect);
    CRect ViewRect;
    GetClientRect (&ViewRect);
    CRect IntRect;
    IntRect.IntersectRect (&ScrollRect, &ViewRect);
    ClientToScreen (&IntRect);
    ::ClipCursor (&IntRect);

    CScrollView::OnLButtonDown(nFlags, point);
}

void CScratchBookView::OnLButtonUp(UINT nFlags, CPoint point)
{
    // TODO: Вставьте сюда собственный код обработчика
    // или вызов стандартного
    int SizeRound;

    if (!m_Dragging) return;

    m_Dragging = 0;
    ::ReleaseCapture ();
    ::ClipCursor (NULL);

    CClientDC ClientDC(this);
    OnPrepareDC (&ClientDC);
    ClientDC.DPtoLP (&point);
    ClientDC.SetROP2 (R2_NOT);
    ClientDC.SelectObject (&m_PenDotted);
    ClientDC.SetBkMode (TRANSPARENT);
    ClientDC.SelectStockObject (NULL_BRUSH);

    CScratchBookApp *PApp = (CScratchBookApp *)AfxGetApp ();
    CFigure *PFigure;

```

```

switch (PApp->m_CurrentTool)
{
case ID_TOOLS_LINE:
    ClientDC.MoveTo (m_PointOrigin);
    ClientDC.LineTo (m_PointOld);
    PFigure = new CLine
        (m_PointOrigin.x, m_PointOrigin.y, point.x,
         point.y, PApp->m_CurrentColor,
         PApp->m_CurrentWidth);
    break;

case ID_TOOLS_RECTANGLE:
    ClientDC.Rectangle (m_PointOrigin.x, m_PointOrigin.y,
                        m_PointOld.x, m_PointOld.y);
    PFigure = new CRectangle
        (m_PointOrigin.x, m_PointOrigin.y, point.x,
         point.y, PApp->m_CurrentColor,
         PApp->m_CurrentWidth);
    break;

case ID_TOOLS_FRECTANGLE:
    ClientDC.Rectangle (m_PointOrigin.x, m_PointOrigin.y,
                        m_PointOld.x, m_PointOld.y);
    PFigure = new CFRectangle
        (m_PointOrigin.x, m_PointOrigin.y, point.x,
         point.y, PApp->m_CurrentColor,
         PApp->m_CurrentWidth);
    break;

case ID_TOOLS_RRECTANGLE:
    SizeRound = (abs(m_PointOld.x - m_PointOrigin.x) +
                 abs (m_PointOld.y - m_PointOrigin.y)) / 8;
    ClientDC.RoundRect (m_PointOrigin.x, m_PointOrigin.y,
                        m_PointOld.x, m_PointOld.y, SizeRound, SizeRound);
    PFigure = new CRRectangle
        (m_PointOrigin.x, m_PointOrigin.y, point.x,
         point.y, PApp->m_CurrentColor,
         PApp->m_CurrentWidth);
    break;

case ID_TOOLS_FRRECTANGLE:
    SizeRound = (abs(m_PointOld.x - m_PointOrigin.x) +
                 abs (m_PointOld.y - m_PointOrigin.y)) / 8;
    ClientDC.RoundRect (m_PointOrigin.x, m_PointOrigin.y,
                        m_PointOld.x, m_PointOld.y, SizeRound, SizeRound);
    PFigure = new CFRRectangle
        (m_PointOrigin.x, m_PointOrigin.y, point.x,
         point.y, PApp->m_CurrentColor,
         PApp->m_CurrentWidth);
    break;

case ID_TOOLS_CIRCLE:
    ClientDC.Ellipse (m_PointOrigin.x, m_PointOrigin.y,
                     m_PointOld.x, m_PointOld.y);
    PFigure = new CCircle
        (m_PointOrigin.x, m_PointOrigin.y, point.x,

```

```

        point.y, PApp->m_CurrentColor,
        PApp->m_CurrentWidth);
    break;

case ID_TOOLS_FCIRCLE:
    ClientDC.Ellipse (m_PointOrigin.x, m_PointOrigin.y,
        m_PointOld.x, m_PointOld.y);
    PFigure = new CFCircle
        (m_PointOrigin.x, m_PointOrigin.y, point.x,
        point.y, PApp->m_CurrentColor);
    break;

    ClientDC.SetROP2 (R2_COPYPEN);
    PFigure->Draw (&ClientDC);
    CScratchBookDoc *PDoc = GetDocument ();
    PDoc->AddFigure (PFigure);

    PDoc->UpdateAllViews (this, 0, PFigure);

CScrollView::OnLButtonUp(nFlags, point);
}
}

void CScratchBookView::OnMouseMove(UINT nFlags, CPoint point)
{
    // TODO: Вставьте сюда собственный код обработчика
    // или вызов стандартного

    int SizeRound;
    CClientDC ClientDC (this);
    OnPrepareDC (&ClientDC);
    ClientDC.DPtoLP (&point);

    if (!m_Dragging)
    {
        CSize ScrollSize = GetTotalSize ();
        CRect ScrollRect (0, 0, ScrollSize.cx, ScrollSize.cy);
        if (ScrollRect.PtInRect (point))
            ::SetCursor (m_HCross);
        else
            ::SetCursor (m_HArrow);
        return;
    }

    ClientDC.SetROP2 (R2_NOT);
    ClientDC.SelectObject (&m_PenDotted);
    ClientDC.SetBkMode (TRANSPARENT);
    ClientDC.SelectStockObject (NULL_BRUSH);

    switch (((CScratchBookApp *)AfxGetApp ()) ->m_CurrentTool)
    {
    case ID_TOOLS_LINE:
        ClientDC.MoveTo (m_PointOrigin);
        ClientDC.LineTo (m_PointOld);
        ClientDC.MoveTo (m_PointOrigin);
        ClientDC.LineTo (point);
        break;
    }
}

```

```

case ID_TOOLS_RECTANGLE:
case ID_TOOLS_FRECTANGLE:
    ClientDC.Rectangle (m_PointOrigin.x, m_PointOrigin.y,
        m_PointOld.x, m_PointOld.y);
    ClientDC.Rectangle (m_PointOrigin.x, m_PointOrigin.y,
        point.x, point.y);
    break;

case ID_TOOLS_RRECTANGLE:
case ID_TOOLS_FRRECTANGLE:
    {
        SizeRound = (abs(m_PointOld.x - m_PointOrigin.x) +
            abs (m_PointOld.y - m_PointOrigin.y)) / 8;
        ClientDC.RoundRect (m_PointOrigin.x,
            m_PointOrigin.y, m_PointOld.x,
            m_PointOld.y, SizeRound,
            SizeRound);
        SizeRound = (abs(point.x - m_PointOrigin.x) +
            abs (point.y - m_PointOrigin.y)) / 8;
        ClientDC.RoundRect (m_PointOrigin.x,
            m_PointOrigin.y, point.x,
            point.y, SizeRound,
            SizeRound);

        case ID_TOOLS_CIRCLE:
        case ID_TOOLS_FCIRCLE:
            ClientDC.Ellipse (m_PointOrigin.x, m_PointOrigin.y,
                m_PointOld.x, m_PointOld.y);
            ClientDC.Ellipse (m_PointOrigin.x, m_PointOrigin.y,
                point.x, point.y);
            break;
    }
    m_PointOld = point;

    CScrollView::OnMouseMove(nFlags, point);
}

void CScratchBookView::OnInitialUpdate()
{
    CScrollView::OnInitialUpdate();

    // TODO: Добавьте сюда собственный код или
    // вызов базового класса

    SIZE Size = {800, 600};
    SetScrollSizes (MM_TEXT, Size);
}

void CScratchBookView::OnUpdate(CView* pSender, LPARAM lHint,
                                CObject* pHint)
{
    // TODO: Добавьте сюда собственный код или
    // вызов базового класса
    if (pHint != 0)
    {

```

```

        CRect InvalidRect = ((CLine *)pHint)->GetDimRect ();
        CClientDC ClientDC (this);
        OnPrepareDC (&ClientDC);
        ClientDC.LPtoDP (&InvalidRect);
        InvalidateRect (&InvalidRect);
    }
    else
        CScrollView::OnUpdate (pSender, lHint, pHint);
}

```

---

### Листинг 19.15

```

// MainFrm.h : интерфейс класса CMainFrame
//

#pragma once
class CMainFrame : public CFrameWnd
{
protected:
    CSplitterWnd m_SplitterWnd;

protected: // используется только для сериализации
    CMainFrame();
    DECLARE_DYNCREATE(CMainFrame)

// Атрибуты
public:

// Операции
public:

// Переопределения
public:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);

// Реализация
public:
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif
protected: // встроенные элементы управления
    CStatusBar m_wndStatusBar;
    CToolBar m_wndToolBar;

// Сгенерированные функции карты сообщений
protected:
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    DECLARE_MESSAGE_MAP()
    virtual BOOL OnCreateClient(LPCREATESTRUCT lpcs, CCreateContext*
                                pContext);
};

```

## Листинг 19.16

```
// MainFrm.cpp : реализация класса CMainFrame
//

#include "stdafx.h"
#include "ScratchBook.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
    ON_WM_CREATE()
END_MESSAGE_MAP()

static UINT indicators[] =
{
    ID_SEPARATOR,           // индикатор строки состояния
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
};

// Конструктор/деструктор класса CMainFrame

CMainFrame::CMainFrame()
{
    // TODO: добавьте сюда код инициализации элементов
}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD |
        WS_VISIBLE | CBRS_TOP | CBRS_GRIPPER | CBRS_TOOLTIPS |
        CBRS_FLYBY | CBRS_SIZE_DYNAMIC) ||
        !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
    {
        TRACE0("Failed to create toolbar\n");
        return -1;          // не удалось создать панель инструментов
    }
}
```

```

        if (!m_wndStatusBar.Create(this) ||
            !m_wndStatusBar.SetIndicators(indicators,
            sizeof(indicators)/sizeof(UINT)))
        {
            TRACE0("Failed to create status bar\n");
            return -1;        // не удалось создать строку состояния
        }
        // TODO: Удалите три следующие строки, если не хотите,
        // чтобы панель инструментов была паркующей
        m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
        EnableDocking(CBRS_ALIGN_ANY);
        DockControlBar(&m_wndToolBar);

        return 0;
    }

    BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
    {
        if( !CFrameWnd::PreCreateWindow(cs) )
            return FALSE;
        // TODO: Модифицируйте стили или классы окна здесь,
        // добавляя или изменяя поля структуры cs

        return TRUE;
    }

    // Диагностика класса CMainFrame

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
    CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
    CFrameWnd::Dump(dc);
}

#endif // _DEBUG

// Обработчики сообщений класса CMainFrame

BOOL CMainFrame::OnCreateClient(LPCREATESTRUCT lpcs,
                               CCreateContext* pContext)
{
    // TODO: Добавьте сюда собственный код
    // или вызов базового класса
    return m_SplitterWnd.Create
        (this,                // родительское окно разделенного окна;
         1,                    // максимальное число строк;
         2,                    // максимальное число столбцов;
         CSize(20, 20),        // минимальный размер окна представления;
         pContext);            // информация о контексте устройства
}

```

## Резюме

---

Рассмотрено рисование фигур с помощью функций класса CDC.

- *Объект контекста устройства.* Рисование начинается с создания объекта контекста устройства. При рисовании с помощью функции OnDraw класса представления можно использовать объект контекста устройства, передаваемый в функцию. При рисовании с помощью функции другого типа можно создать экземпляр класса CClientDC, а при рисовании в окне представления, поддерживающем прокрутку – передать объект в функцию CScrollView::OnPrepareDC до его использования.
- *Перо и кисть.* Далее необходимо выбрать перо для рисования линий и границ фигур с замкнутыми контурами и кисть для закрашивания внутренних областей фигур. Стандартное перо или кисть выбирается при вызове функции CDC::SelectStockObject. Можно выбрать перо или кисть, создавая объект класса CPen или CBrush, и вызывая соответствующую функцию для инициализации пера или кисти, а затем вызвав функцию CDC::SelectObject для выбора инструмента в объекте контекста устройства. Предыдущий выбранный объект, возвращенный функцией SelectObject, необходимо сохранить.
- *Атрибуты рисования.* Для установки любых желаемых атрибутов рисования используйте функции класса CDC (см. табл. 19.4). К атрибутам относится режим отображения, определяющий единицы измерения и положительное направление используемых для рисования координат.
- *Рисование изображения.* Теперь можно приступить к рисованию изображения, вызывая функции класса CDC, предоставляющего функции для закрашивания отдельных пикселей, рисования прямых или кривых линий и таких фигур, как прямоугольники и эллипсы. Завершив рисование, если вы создали и выбрали пользовательское перо или кисть, то чтобы вернуться к предыдущему перу или кисти, удалите их из объекта контекста устройства (вызвав функцию SelectObject).



## Глава 20

# Растровые изображения и битовые операции

---

- Растровые изображения
- Битовые операции
- Значки
- Программа ChessBoard

Для сохранения точного представления рисунка в памяти или файле Windows используется специальная структура данных – *растровое изображение (или точечный рисунок)*. Растровое изображение хранит цвет каждого пикселя, необходимого для генерации рисунка на таких устройствах, как экран дисплея или принтер. В этой главе вы узнаете, как создаются и отображаются растровые изображения и как использовать преимущества универсальных и эффективных функций битовых операций, предоставляемых для перемещения и модификации блоков графических данных. В конце главы будет создана демонстрирующая способы отображения растрового изображения в окне представления программа ChessBoard.

## Растровые изображения

---

Для управления растровыми изображениями в библиотеке MFC предусмотрен класс CBitmap. Поэтому при их создании первым действием будет объявление экземпляра этого класса. Объект растрового изображения обычно объявляется как переменная в одном из классов главной программы, например, классе представления. После объявления объекта класса CBitmap

```
CBitmap m_Bitmap;
```

необходимо вызвать соответствующую функцию этого класса для инициализации объекта. В этой главе рассмотрена:

- инициализация объекта класса CBitmap вызовом функции LoadBitmap для загрузки растрового изображения из ресурсов программы;
- инициализация объекта класса CBitmap вызовом функции CreateCompatibleBitmap для создания пустого растрового изображения, в котором можно нарисовать требуемое изображение при выполнении программы.

Кроме двух упомянутых методов, описанных в этой главе, Windows предоставляет несколько других способов инициализации объекта растрового изображения. Например, можно вызвать функцию:

- CBitmap::LoadOEMBitmap – для загрузки предопределенного растрового изображения, предоставляемого Windows;
- CBitmap::CreateBitmap – для создания растрового изображения, совместимого с соответствующей структурой;
- CGdiObject::Attach – для инициализации объекта растрового изображения дескриптором растрового изображения Windows (CGdiObject – базовый класс для класса CBitmap). Информация по этому вопросу помещена в документации на класс CBitmap в справочной системе.

## Растровое изображение в ресурсах

Чтобы в ресурсы программы включить растровое изображение, можно воспользоваться средствами конструирования растрового изображения, входящими в состав

- интерактивного графического редактора из среды Visual C++ или
- отдельной программы рисования, например, Paint в Windows.

Данный метод особенно полезен для создания относительно сложных рисунков, которые с помощью функций рисования сгенерировать трудно. Для нашего примера создайте рисунок шахматной доски размером приблизительно 300×300 пикселей в Paint или другом графическом редакторе, способном создавать bmp-файлы.

Начинать, как обычно, нужно с открытия (или создания) проекта программы, использующей растровое изображение. (Если проект не открыт, ресурс растрового изображения будет помещен в отдельный файл ресурсов с расширением .rc.) В Visual Studio откройте (создайте) проект ChessBoard со стандартными установками, отключив генерацию панели инструментов и строки состояния. Откройте вкладку Resource View и, щелкнув на списке ресурсов правой кнопкой мыши, из контекстного меню выберите пункт New. В открывшемся диалоговом меню выберите пункт Bitmap. Новому растровому изображению будет присвоен стандартный идентификатор: IDB\_Bitmap1 – первому создаваемому изображению, IDB\_Bitmap2 – второму и т.д. Если хотите, задайте другой стандартный идентификатор: выполните двойной щелчок в окне растрового изображения, откройте вкладку General диалогового окна Properties и введите новый идентификатор в текстовое поле ID. Приняв стандартный идентификатор или присвоив собственный, запомните его, потому что при составлении программы загрузки растрового изображения он понадобится.

Для создания растрового изображения можно воспользоваться собственным графическим редактором Visual C++ или отдельной программы формирования изображения с последующим сохранением его в файле. Можно использовать любую программу, сохраняющую файлы в формате растрового изображения (обычно файл имеет расширение .BMP или .DIF). Пример такой программы – утилита Paint в Windows 95. Чтобы использовать растровое изображение, поместите его файл в буфер в программе работы с изображениями, щелкните правой кнопкой на вкладке Resource View, создайте новое или выберите загрузку уже созданного растрового изображения, а в запущенном редакторе растровых изображений выберите команду Paste в меню Edit. В графическом редакторе Visual C++ будет отображено импортированное растровое изображение. Его можно редактировать, изменять размеры или присваивать другой идентификатор. Построено ли растровое изображение в графическом редакторе Visual C++ или импортировано из файла – в любом случае оно будет включено в ресурсы программы. При ее выполнении можно загрузить растровое изображение и использовать его для инициализации объекта. Для этого вызовите функцию LoadBitmap класса CBitmap, как показано ниже.

```
class CProgView : public CView    // класс представления программы
{
// ...
    CBitmap m_Bitmap;
    void LoadBitmapImage ();
// ...
};

// ...
void CProgView::LoadBitmapImage ()
{
    // ...

    m_Bitmap.LoadBitmap (IDB_BITMAP1);
}
```

```
// ...
}
```

В качестве параметра в функцию `LoadBitmap` передается идентификатор, присвоенный растровому изображению при его создании или импортировании. Конструирование растрового изображения в графическом редакторе Visual C++ и использование его для инициализации объекта описаны в конце главы, в упражнении по созданию программы `ChessBoard`.

## Рисование растрового изображения

При выполнении программы можно инициализировать пустое растровое изображение и использовать функции рисования MFC для генерации картинки. Для этого необходимо выполнить следующую процедуру:

1. Следует выполнить инициализацию пустого растрового изображения. Для инициализации пустого растрового изображения вызывается функция `CreateCompatibleBitmap`. Например:

```
class CProgView : public CView // класс представления программы
{
// ...
    CBitmap m_Bitmap;
    void DrawBitmapImage ();
// ...
};

// ...

void CProgView::DrawBitmapImage ()
{
    CClientDC ClientDC (this); // создание объекта контекста
                               // устройства окна представления

    m_Bitmap.CreateCompatibleBitmap (&ClientDC, 32, 32);

// ...
}
```

Передаваемый в функцию `CreateCompatibleBitmap` первый параметр, является адресом объекта контекста устройства. Растровое изображение совместимо с устройством, соответствующим объекту. Термин *совместимо* означает, что графические данные растрового изображения будут структурированы способом, подобным используемому при структурировании графических данных самим устройством (при совместимости графические данные можно быстро передавать между растровым изображением и устройством). Чтобы создать растровое изображение на экране, необходимо передать для него адрес объекта контекста устройства. Обычно это явно определенный объект контекста устройства `CClientDC` или объект контекста устройства, передаваемый в функцию `OnDraw` класса представления. Растровое изображение при вызове функции `CBitmap::LoadBitmap` для загрузки (см. предыдущий параграф) автоматически становится совместимым с экраном. Второй параметр, передаваемый в функцию `CreateCompatibleBitmap`, – это ширина растрового изображения, третий – высота. Оба размера задаются в пикселях.

Обрабатывая вызов функции `CreateCompatibleBitmap`, Windows резервирует блок памяти для растрового изображения. Цветовые коды пикселей, хранящиеся внутри него, изначально не определены. Для создания нужного изображения используются функции рисования.

2. Но прежде чем начать рисовать, необходимо создать объект контекста устройства, связанный с этим изображением (в точности так же, как необходимо иметь объект контекста устройств для

передачи выводимых данных на устройство типа экрана). Система Windows предоставляет специальный объект контекста устройства для доступа к растровому изображению, называемый *объектом памяти контекста устройства*. Чтобы его создать, объявляется экземпляр класса CDC, а затем вызывается функция CreateCompatibleDC этого класса.

```
void CProgView::DrawBitmapImage ()
{
    CClientDC ClientDC (this); // объект контекста устройства
                                // окна представления
    CDC MemDC;                // объект памяти контекста устройства

    m_Bitmap.CreateCompatibleBitmap (&ClientDC, 32, 32);

    MemDC.CreateCompatibleDC (&ClientDC);

    // ...
}
```

Параметр, передаваемый в функцию CreateCompatibleDC, – это адрес объекта контекста устройства. Результирующий объект памяти контекста устройства будет совместим с устройством, связанным с этим объектом. Передаваемый объект контекста устройства должен быть связан с *таким же* устройством, как и объект, передаваемый в функцию CreateCompatibleBitmap. Другими словами, растровое изображение и объект памяти контекста устройства, используемый для доступа к растровому изображению, должны быть совместимы с одним устройством. В этих примерах как растровое изображение, так и объект памяти контекста устройства, совместимы с экраном. При необходимости инициализации объекта памяти контекста устройства следует в функцию CreateCompatibleDC передать значения NULL.

3. Необходимо вызвать функцию SelectObject класса CDC, чтобы выбрать объект растрового изображения в объекте памяти контекста устройства. Например:

```
MemDC.SelectObject (&m_Bitmap);
```

Параметр, передаваемый в функцию SelectObject – это адрес объекта растрового изображения.

4. Теперь можно нарисовать необходимый фрагмент внутри растрового изображения, вызывая функции рисования класса CDC для объекта контекста устройства. Внутри растрового изображения можно отобразить текст или графику, используя функции, применяемые для рисования внутри окна (см. гл. 18 и 19). Кроме того, можно использовать любой из описанных ниже битовых операторов. Например, функция PatBlt полезна для создания цветного фона внутри растрового изображения. Следующий фрагмент программы рисует белый фон, а затем круг внутри растрового изображения, переданного в объект памяти контекста устройства MemDC. В этом примере предполагается, что растровому изображению был задан размер 32×32 пикселя при вызове функции CreateCompatibleBitmap.

```
// рисование белого фона:
MemDC.PatBlt (0, 0, 32, 32, WHITENESS);

// рисование круга:
MemDC.Ellipse (2, 2, 30, 30);
```

Ниже приведен фрагмент программы, который иллюстрирует все шаги, описанные в этом параграфе. Он инициализирует растровое изображение, а затем строит рисунок внутри него.

```
class CProgView : public CView // класс представления программы
{
    // ...
}
```

```

        CBitmap m_Bitmap;
        void DrawBitmapImage ();
    // ...
};

// ...

void CProgView::DrawBitmapImage ()
{
    CClientDC ClientDC (this);    // объект контекста устройства
                                   // окна представления;
    CDC MemDC;                    // объект памяти контекста устройства

    // инициализация пустого растрового изображения:
    m_Bitmap.CreateCompatibleBitmap (&ClientDC, 32, 32);

    // инициализация объекта памяти контекста устройства (КУ)
    MemDC.CreateCompatibleDC (&ClientDC);

    // передача объекта растрового изображения в объект КУ памяти:
    MemDC.SelectObject (&m_Bitmap);

    // использование функции класса CDC для рисования внутри
    // растрового изображения:

    // рисование белого фона:
    MemDC.PatBlt (0, 0, 32, 32, WHITENESS);

    // рисование круга:
    MemDC.Ellipse (2, 2, 30, 30);

    // вызов других функций рисования ...
}

```

Как правило, объект растрового изображения объявляется как переменная одного из классов основной программы (например, класса представления) и, следовательно, обрабатывается на протяжении всей программы. Однако если объект растрового изображения удаляется *перед* удалением объекта памяти контекста устройства, его необходимо сначала удалить из объекта памяти контекста устройства. Как это сделать для инструментов рисования, описано в гл. 19. А именно: в результате работы функции `SelectObject` указатель будет указывать на стандартное растровое изображение, которое для созданного объекта памяти контекста устройства является монохромным изображением, содержащим один пиксель. По окончании работы с растровым изображением снова вызывается функция `SelectObject`, устанавливающая указатель на объект контекста устройства.

## Отображение растрового изображения

Создав и инициализировав объект растрового изображения с использованием одного из двух рассмотренных методов, можно отобразить этот объект прямо внутри окна или другого устройства. Средства Win32 API и библиотека MFC не содержат функции, которую можно вызвать для простого отображения растрового изображения на устройстве. Поэтому напомним свою собственную функцию. Она может выглядеть так:

```

void DisplayBitmap (CDC *PDC, CBitmap *PBitmap, int X, int Y)
{
    BITMAP BM;

```

```

CDC MemDC;

MemDC.CreateCompatibleDC (NULL);
MemDC.SelectObject (PBitmap);
PBitmap->GetObject (sizeof (BM), &BM);
PDC->BitBlt
    (X,          // логическая горизонтальная координата
     Y,          // приемника;
     BM.bmWidth, // логическая вертикальная координата
     BM.bmHeight, // приемника;
     BM.bmWidth, // ширина перемещаемого блока
     BM.bmHeight, // (в логических единицах);
     BM.bmHeight, // высота перемещаемого блока
     BM.bmHeight, // (в логических единицах);
     &MemDC,      // КУ источника для графических данных;
     0,          // логическая горизонтальная координата
     0,          // блока внутри источника;
     0,          // логическая вертикальная координата
     0,          // блока внутри источника;
     SRCCOPY);   // код типа перемещения
    )

```

Предлагаемая функция `DisplayBitmap` отображает растровое изображение на устройстве, соответствующем объекту контекста устройства, передаваемому в первом параметре. Второй параметр содержит адрес объекта растрового изображения, который необходимо инициализировать, используя один из описанных ниже способов, *совместимых с экраном*. Последние два параметра описывают горизонтальную и вертикальную координаты позиции внутри целевого устройства, в которой должен размещаться левый верхний угол растрового изображения.

Функция `DisplayBitmap` сначала создает объект контекста устройства, совместимый с экраном, и передает растровое изображение внутрь этого объекта, поэтому он может иметь доступ к содержимому растрового изображения. Затем она вызывает функцию `GetObject` класса `CGdiObject`, заполняющую элементы структуры `BITMAP` информацией растрового изображения. Функция `DisplayBitmap` получает размер растрового изображения из переменных `bmWidth` и `bmHeight` этой структуры.

Затем она вызывает функцию `BitBlt` класса `CDC`, перемещающую графические данные, содержащиеся в растровом изображении, прямо на целевое устройство. Первые два параметра, передаваемые в функцию `BitBlt`, задают левый верхний угол размещения приемника, третий и четвертый – ширину и высоту перемещаемого блока данных. Функция `DisplayBitmap` перемещает все растровое изображение, передавая его ширину и высоту, полученные из переменных `bmWidth` и `bmHeight` структуры `BITMAP`. Пятый параметр `&MemDC` – это адрес объекта контекста устройства, являющегося источником графических данных; функция `DisplayBitmap` описывает объект памяти контекста устройства, связанного с растровым изображением. Шестой и седьмой параметры соответствуют левому верхнему углу блока графических данных, перемещаемых из объекта контекста устройства источника. Поскольку функция `DisplayBitmap` перемещает растровое изображение целиком, она задает координаты (0, 0). Последний параметр – это код, который указывает, как должны быть перемещены графические данные. Значение `SRCCOPY` показывает, что они должны быть скопированы без изменений. Функция `BitBlt` – одна из функций битовых операций, описанных в следующем параграфе. Функция `DisplayBitmap` задает объекту контекста устройства для целевого устройства стандартный режим отображения `MM_TEXT`. Чтобы при копировании растровое изображение было расширено или сжато, объект должен использовать другой режим отображения.

Ниже приведен пример, который иллюстрирует, как для отображения растрового изображения внутри окна представления с логическими координатами (0, 0) может использоваться функция `DisplayBitmap`.

```
void CProgView::OnDraw(CDC* pDC)
{
    DisplayBitmap (pDC, &m_Bitmap, 0, 0);
}
```

Здесь задан объект `m_Bitmap` класса `CBitmap` в форме элемента класса представления `CProgView`, инициализированный как совместимый с экраном с помощью одного из описанных ранее методов. Программа `ChessBoard`, приведенная в конце главы, демонстрирует альтернативные методы отображения растрового изображения в окне представления. Кроме того, функцию `DisplayBitmap` можно использовать для печати растрового изображения, передав его в объект контекста устройства, соответствующего принтеру. Способы печати описаны в гл. 21. При использовании монохромного принтера цвета растрового изображения могут быть заменены черно-белыми или оттенками серого цвета.

В программах библиотеки MFC инициализированный объект растрового изображения можно использовать и для других целей (кроме отображения на устройстве соответствующего точечного рисунка). Например, можно с помощью MFC-класса `CBitmapButton` создать специальный элемент управления типа кнопки помеченной растровым изображением, а не текстом. Вызывая функцию `CMenu::SetMenuItemBitmap` можно пометить команду меню специальным значком (необходимый для этого временный объект класса `CMenu`, подключаемый к меню главной программы, можно получить, вызывая функцию `CWnd::GetMenu` для объекта главного окна). Вызывая функцию `CMenu::AppendMenu` или другую функцию класса `CMenu`, программист может сконструировать специальную метку меню и задать растровое изображение вместо текстовой метки. А затем можно заполнить области с помощью специального шаблона, вызвав функцию `CBrush::CreatePatternBrush` для создания кисти, как описано в гл. 19. При выполнении любого из этих действий необходимо создать объект `CBitmap`, инициализируя его с использованием одного из методов, описанных в этой главе. Информация по любому из упомянутых способов использования растровых изображений приведена при описании классов или функций-членов в справочной системе.

## Битовые операции

В состав класса `CDC` входят три универсальные и эффективные функции для перемещения блоков графических данных: `PatBlt`, `BitBlt` и `StretchBlt`. Эти функции можно использовать при создании рисунков, а также для копирования блоков графических данных и их модификации простыми или сложными способами (например, инвертированием цветов или зеркальным отражением изображения). Ранее в этой главе было показано, как использовать функцию `PatBlt` для рисования цветного фона в растровом изображении и функцию `BitBlt` для пересылки графических данных из растровых изображений в окно или другое устройство. Функции битовых операций можно применять к объектам контекста устройства, связанным с экраном или к объектам памяти контекста устройства, совместимым с экраном. Однако они могут не поддерживаться некоторыми типами устройств, например, отдельными принтерами или плоттерами. Чтобы определить, поддерживает ли конкретное устройство функции битовых операций, можно передать значение `RASTERCAPS` в функцию `CDC::GetDeviceCaps` для связанного с ним объекта контекста устройства. Информация, возвращаемая функцией `GetDeviceCaps`, описана в документации по данной функции. Вообще функции `PatBlt`, `BitBlt` и `StretchBlt` можно применять:

- для копирования графических данных (с одного места в другое) на отображающей поверхности *в пределах одного отдельного устройства*;
- для копирования графических данных *с одного устройства на другое*;
- для обмена графическими данными *между устройством и растровым изображением*.

## Функция PatBlt

Принадлежащая классу CDC функция PatBlt предназначена для закрашивания прямоугольной области с использованием текущей кисти. Описание текущей кисти приведено в гл. 19. В контексте битовых операций на текущую кисть обычно ссылаются как на текущий *шаблон (pattern)*. (Область можно заполнить текущим шаблоном, вызывая функцию CDC::FillRec). Функция PatBlt более универсальна, ее синтаксис выглядит так:

```
BOOL PatBlt
(int x, int y,           // логическая координата верхнего
                        // левого угла области заполнения;
int nWidth, int nHeight, // размеры области заполнения в
                        // логических единицах;
DWORD dwRop);           // код растровой операции
```

Здесь два первых параметра задают логические координаты левого верхнего угла закрашиваемой прямоугольной области, два следующих – выраженную в логических единицах ширину и высоту этой области.

Последний параметр dwRop – это *код растровой операции*, который придает функции PatBlt (как и другим функциям битовых операций) свойство универсальности. Он задает способ объединения каждого пикселя внутри шаблона с текущим пикселем приемника, определяющими окончательный цвет пикселя. Задание кода растровой операции при вызове функции битовой операции похоже на установку режима рисования (см. гл. 19). При этом режим рисования влияет на линии и внутреннюю область замкнутых фигур, созданных с использованием команд рисования, но не влияет на результаты рассмотренных далее битовых (растровых) операций. Коды растровых операций, передаваемые в функцию PatBlt, перечислены в табл. 20.1. В ней для каждого кода приведено булево выражение, описывающее результирующий цвет каждого пикселя (D – пиксель приемника, P – пиксель источника) внутри заполняемой области.

Табл. 20.1. Коды растровых операций (последний параметр функции PatBlt)

Код растровой операции	Булево выражение	Описание результата в области приемника
BLACKNESS	D = 0	Каждый пиксель устанавливается черным
DSTINVERT	D = ~D	Цвет каждого пикселя инвертируется
PATCOPY	D = P	Каждому пикселю задается цвет пикселя шаблона
PATINVERT	D = D^P	Цвет каждого пикселя – это результат выполнения логической операции XOR над пикселем-приемником и пикселем шаблона
WHITENESS	D = 1	Каждый пиксель устанавливается белым

Система Windows выполняет указанную растровую операцию с каждым *битом*, используемым для кодирования цвета пикселя. Результирующий цвет зависит от способа представления цвета на устройстве, ассоциированном с объектом контекста устройства. В устройстве монохромного отображения для каждого пикселя используется только один бит. В устройстве цветного изображения – несколько битов. Например, при инвертировании цвета пикселя ( $D = \sim D$ ) Windows инвертирует каждый бит, используемый в коде цвета.

В приведенном ниже примере функция OnDraw нарисует все окно представления, используя текущий шаблон, полностью заменяющий содержимое текущего окна:

```
void CProgView::OnDraw(CDC* pDC)
{
```



```

RECT Rect;
GetClientRect (&Rect);

pDC->PatBlt
    (Rect.left,
     Rect.top,
     Rect.right - Rect.left,
     Rect.bottom - Rect.top,
     PATCOPY);
}

```

## Функция *BitBlt*

Принадлежащая классу CDC функция *BitBlt* передает блок графических данных из одного места хранения в другое. Источник и приемник могут находиться внутри одного устройства (или растрового изображения) или внутри различных устройств (или растровых изображений). Синтаксис функции *BitBlt* выглядит так:

```

BOOL BitBlt
    int x, int y,                // логические координаты левого
                                // верхнего угла блока приемника;
    int nwidth, int nheight,     // размеры блока в логических
                                // единицах;
    CDC* pSrcDC,                // объект контекста устройства
                                // источника;
    int xSrc, int ySrc,         // логические координаты левого
                                // верхнего угла блока внутри
                                // источника;
    DWORD dwRop);              // код растровой операции

```

Здесь два первых параметра задают логические координаты левого верхнего угла размещения приемника передаваемых данных, два следующих – размеры передаваемого блока графических данных в логических единицах. Пятый параметр *pSrcDC* – это указатель на объект контекста устройства источника. Шестой и седьмой (*xSrc* и *ySrc*) – задают логические координаты левого верхнего угла блока внутри устройства источника. Код растровой операции задается последним параметром *dwRop*. Функция *BitBlt* копирует блок графических данных из устройства, ассоциированного с объектом контекста устройства, представленного пятым параметром, на устройство, ассоциированное с объектом контекста устройства, для которого вызвана функция. Например, следующий вызов передает блок графических данных из растрового изображения, выбранного в объекте памяти контекста устройства *MemDC*, в связанное с объектом контекста устройства *\*pDC* окно представления.

```
pDC->BitBlt (x, y, width, height, &MemDC, 0, 0, SRCCOPY);
```

Окончательный цвет каждого пикселя области приемника, нарисованного функцией *BitBlt*, зависит:

- от текущего цвета этого пикселя *области приемника*;
- от цвета соответствующего пикселя внутри *устройства источника*;
- от цвета соответствующего пикселя внутри *текущего шаблона* (т.е. выбранной *текущей кисти* в объекте контекста устройства *приемника*).

Способ объединения этих значений функцией *BitBlt* зависит от растровой операции, задаваемой параметром *dwDrop*. Поскольку код растровой операции, передаваемый функции *BitBlt*, действует на способ объединения цвета из *трех* различных пикселей, получается намного больше возможных

кодовых комбинаций, чем можно задать для функции PatBlt. Код растровой операции, передаваемый в эту функцию, действует на способ объединения пикселей. Фактически, в функцию BitBlt можно передать 256 различных кодов растровых операций. Как и функция PatBlt, функция BitBlt выполняет указанную операцию над каждым битом, используемым для кодирования цвета пикселя. Наиболее распространенные операции перечислены в табл. 20.2. Полный их список приведен в справочной системе. В логических выражениях, приведенных в табл. 20.2, буква D указывает на пиксель приемника, S – на пиксель источника, а P – на пиксель шаблона.

Табл. 20.2. Некоторые коды растровых операций, передаваемые в функции BitBlt или StretchBlt

Код растровой операции	Булево выражение	Описание результата в области приемника
MERGECOPY	$D = P \& S$	Цвет каждого пикселя – результат объединения пикселей шаблона и источника с использованием логической операции AND
MERGEPAINT	$D = \sim P \mid S$	Цвет каждого пикселя – результат объединения инвертированного пикселя источника и пикселя приемника с использованием логической операции OR
NOTSRCOPY	$D = \sim S$	Цвет каждого пикселя устанавливается инверсно по отношению к пикселю источника
NOTSRCERASE	$D = \sim(D \mid S)$	Цвет каждого пикселя – результат объединения пикселей приемника и источника с использованием логического оператора OR и последующей инверсией результата
PATPAINT	$D = \sim S \mid P \mid D$	Цвет каждого пикселя – результат объединения инвертированного пикселя источника, пикселя шаблона и пикселя приемника с использованием логической операции OR
SRCAND	$D = D \& S$	Цвет каждого пикселя – результат объединения пикселей приемника и источника с использованием логической операции AND
SRCCOPY	$D = S$	Каждый пиксель устанавливается в цвет пикселя источника
SRCERASE	$D = \sim D \mid S$	Цвет каждого пикселя – результат объединения инвертированного пикселя приемника и пикселя источника с использованием логической операции OR
SRCINVERT	$D = D \wedge S$	Цвет каждого пикселя – результат объединения пикселей приемника и источника с использованием логической операции XOR
SRCPAINT	$D = D \mid S$	Цвет каждого пикселя – результат объединения пикселей приемника и источника с использованием логической операции OR

## Анимация с помощью функции BitBlt

Функцию BitBlt можно использовать для выполнения анимации, применив соответствующие коды растровых операций. При написании игр или приложений различных типов иногда нужно выполнить перемещение маленького рисунка поверх сложного фона. Его можно передвигать в соответствии с перемещением мыши или автоматически, используя таймер Windows. Если рисунок прямоугольный, то вызывается функция BitBlt с кодом растровой операции SRCCOPY, чтобы отобразить

растровое изображение на новом месте рисунка в окне. Этот метод используется для отображения рисунка, независимо от его формы, если окно имеет в качестве фона однородную заливку. В растровом изображении источника рисуется область вокруг рисунка с использованием цвета фона окна так, чтобы эта часть растрового изображения при копировании в окно становилась прозрачной (невидимой).

Однако, иногда требуется выполнить анимацию рисунка не прямоугольной формы внутри окна, содержащего различные цвета (пестрый фон). Например, это может быть перемещение рисунка шахматной фигуры внутри окна, содержащего в качестве фона рисунок шахматной доски. Проблема в том, что функция `BitBlt` всегда перемещает *прямоугольный* графический блок, и пиксели, окружающие рисунок в растровом изображении источника, перезапишутся на существующие пиксели экрана. Поэтому шахматная фигура будет иметь нежелательную окружающую ее прямоугольную “ауру”. Решение этой проблемы – создание двух исходных растровых изображений:

- *маски* (в растровом изображении маски рисунок закрашен черным цветом, фон – белым);
- *шаблона* (в растровом изображении шаблона рисунок приведен в его обычных цветах, фон закрашен черным). На следующем рисунке показаны растровые изображения маски и куба (шаблона).



Для отображения рисунка в определенном месте внутри окна, можно использовать два вызова функции `BitBlt`:

- в первом вызове перемещается *растровое изображение маски* с использованием кода растровой операции `SRCAND`;
- во втором вызове перемещается *растровое изображение шаблона* с использованием кода растровой операции `SRCINVERT`. Например, так.

```
void CProgView::DisplayDrawing (int X, int Y)
{
    CClientDC ClientDC (this);
    CDC MemDC;
    MemDC.CreateCompatibleDC (&ClientDC);

    // перемещение растрового изображения маски:
    MemDC.SelectObject (&m_MaskBitmap);

    ClientDC.BitBlt
        (X, Y,
         BMWIDTH, BMHEIGHT,
         &MemDC,
         0, 0,
         SRCAND);

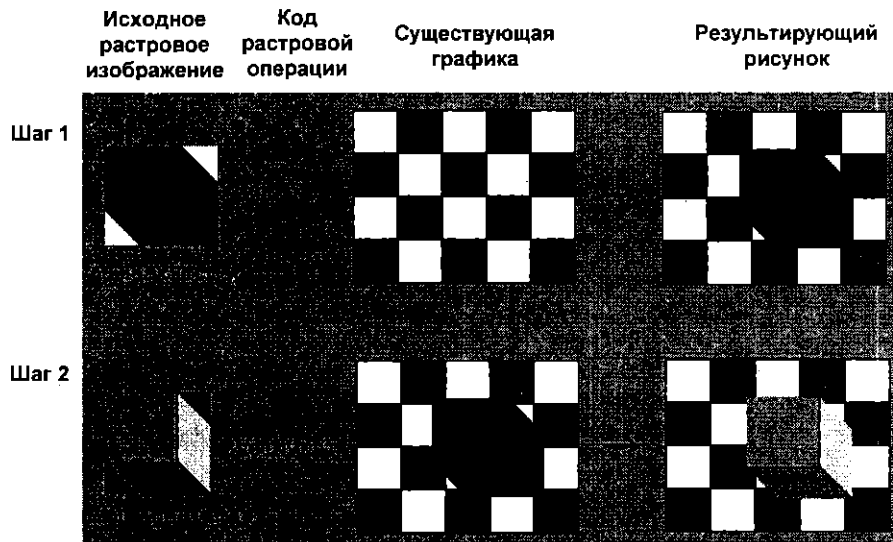
    // перемещение растрового изображения шаблона:
    MemDC.SelectObject (&m_ImageBitmap);
    ClientDC.BitBlt
        (X, Y,
         BMWIDTH, BMHEIGHT,
         &MemDC,
```

```

    0, 0,
    SRCINVERT);
},

```

Здесь предполагается, что программа уже создала и инициализировала два объекта растровых изображений, совместимых с экраном: `m_MaskBitmap` – для маски и `m_ImageBitmap` – для шаблона, а также определила константы `BMWIDTH` и `BMHEIGHT`, равные ширине и высоте этих растровых изображений. Первый вызов функции `BitBlt` отображает рисунок в черном цвете, при этом незатронутая часть фонового рисунка в окне остается неизменной. Второй – перемещает раскрашенную версию рисунка в окно, опять-таки не нарушая существующее графическое изображение вокруг рисунка. Результат двухшагового процесса для отображения непрямоугольного рисунка поверх существующего фонового графического изображения приведен на следующем рисунке.



Как правило, требуется сохранять и восстанавливать графические изображения в окне при каждом временном отображении движущегося рисунка. Для этого используется другое растровое изображение, имеющее такие же размеры, как и растровое изображение маски и шаблона. Функцию `BitBlt` можно использовать с кодом растровой операции `SRCCOPY` сначала для копирования фрагмента фоновой графики с экрана в это растровое изображение (перед отображением перемещаемого рисунка). Для восстановления фона (и стирания наложенного изображения в этой позиции) для копирования сохраненного фрагмента растрового изображения снова на экран вновь используется функция `BitBlt`.

## Функция `StretchBlt`

Функция `StretchBlt` – самая универсальная из рассмотренных в этой главе трех функций битовых операций. Она допускает выполнение всех операций, возможных при использовании функции `BitBlt`. Кроме того, позволяет *изменить размер* блока графических данных или повернуть блок (по горизонтали, по вертикали или в обоих направлениях) при его перемещении. Ее синтаксис выглядит так:

```

BOOL StretchBlt
(int x, int y,           // логические координаты левого
                        // верхнего угла блока-приемника;
int nwidth, int nHeight, // размеры блока назначения
                        // в логических единицах;

```

```

CDC* pSrcDC,           // объект контекста устройства
                        // источника;
int xSrc, int ySrc,     // логические координаты левого
                        // верхнего угла блока внутри
                        // источника;
int nSrcWidth, int nSrcHeight, // размеры блока источника
                        // в логических единицах;
DWORD dwRop);          // код растровой операции

```

Для функции `StretchBlt` задается размер блока источника и блока приемника. При последующем вызове функции `BitBlt` указывается только размер одного блока. Если размер приемника (`nWidth`, `nHeight`) меньше размера источника (`nSrcWidth`, `nSrcHeight`), то изображение сжимается, если наоборот – растягивается. При сжатии блока графических данных функция `StretchBlt` удаляет пиксели. Функция `CDC::SetStretchBltMode` позволяет подобрать способ, которым выполняется удаление. Если значения `nWidth` и `nSrcWidth` заданы с различными знаками (одно положительное, другое отрицательное), то изображение приемника будет зеркальным отображением источника (по горизонтали). Аналогично, если значения `nHeight` и `nSrcHeight` заданы с различными знаками, то изображение приемника будет зеркальным отображением по вертикали. Техника использования функции `StretchBlt` для отображения растрового изображения с заполнением всего окна представления продемонстрирована в программе `ChessBoard`, рассмотренной далее в этой главе.

## Значки

Значок представляет собой специальную форму растрового изображения, которая отличается от стандартного двумя основными признаками:

- Ресурс одного значка может содержать *одно или несколько изображений* (ресурс растрового изображения содержит единственное изображение). Например, ресурс значка может содержать 16-цветные изображения размером 16×16 пикселей и 32×32 пикселя, 2-цветное (монохромное изображение) размером 32×32 пикселя. При отображении значка система использует изображение, наиболее соответствующее текущему видеорежиму и размеру изображения. (Например, можно выбрать изображение крупного или мелкого значка в `Windows Explorer`.)
- При разработке значка с использованием графического редактора `Visual Studio` или другого конструктора значков, можно задавать альтернативные цвета каждому пикселю, присваивая ему *цвет экрана* или *инверсный цвет экрана*. При отображении значка цвет каждого существующего пикселя, который в файле значка обозначен как прозрачный, на экране остается неизменным, т.е. часть значка, окрашенная в цвет экрана, будет прозрачной. Аналогично цвет существующего пикселя в позиции каждого пикселя с инверсным цветом экрана будет инвертирован. Таким образом, часть значка с инверсным цветом экрана будет видима при любом цвете фона.

Два способа использования значков уже знакомы вам. Для главного окна программы (см. гл. 10) или для дочернего окна в MDI-приложении (см. гл. 17) можно создать отдельный значок, отображаемый в строке заголовка программы или в каком-либо другом месте. Кроме того, значок можно отобразить внутри диалогового окна (см. гл. 15).

Для отображения значков в любом месте внутри окна программы используется следующая процедура.

1. Начать необходимо с конструирования значка с использованием графического редактора `Visual Studio` (см. гл. 10). Если значку присвоить идентификатор `IDR_MAINFRAME` (или идентификатор типа документа программы в MDI-приложении), то он автоматически будет назначен главному или дочернему окну программы и отобразится в строке заголовка окна. Если не хотите, чтобы конструируемый значок был назначен окну, убедитесь, что он имеет другой идентификатор. `Visual`

Studio присваивает создаваемым или импортируемым значкам стандартные идентификаторы `IDI_ICON1`, `IDI_ICON2` и т.д.

Вместо конструирования значка в графическом редакторе, можно *импортировать* значок из файла `.ico`, создаваемого с использованием отдельной программы редактирования значков или полученного из другого источника. При желании его можно отредактировать или изменить его идентификатор с помощью графического редактора Visual Studio. Чтобы импортировать файл значка, выберите пункт `New` в контекстном меню окна `Resource View`, меню `Insert`, щелкните на кнопке `Import...` в диалоговом окне `Insert Resource` и выберите файл `.ico` в диалоговом окне `Import Resource`.

2. Далее необходимо созданный или импортированный значок сохранить в файле `.ico` и включить его в программные ресурсы при построении программы.
3. Перед отображением значка при выполнении программы необходимо вызвать функцию `LoadIcon` класса `CWinApp`, чтобы загрузить значок из ресурсов и получить его дескриптор, например, так:

```
HICON hIcon;
```

```
hIcon = AfxGetApp()->LoadIcon (IDI_ICON1);
```

Здесь `IDI_ICON1` – идентификатор, присвоенный значку при его создании или импортировании. Заметим: функция `AfxGetApp` вызывается, чтобы получить указатель на объект программы приложения, используемый для вызова функции `LoadIcon`. Если ресурс значка содержит несколько изображений, например, 16-цветное изображение `32x32` бита и 2-цветное монохромное `32x32` бита, то автоматически будет загружено изображение, наиболее подходящее для текущего видеорежима. Перед вызовом функции `LoadIcon` для загрузки отдельного значка из ресурсов программы можно вызвать функцию `LoadStandardIcon` или `LoadOEMIcon` класса `CWinApp`, чтобы получить дескриптор предопределенного значка, предоставляемый Windows. Подробную документацию по этим двум функциям смотрите в справочной системе.

4. Чтобы отобразить значок внутри окна вызовите функцию `DrawIcon` класса `CDC`. Задаваемые для этой функции параметры `x` и `y` определяют координаты левого верхнего угла места расположения значка. Параметр `hIcon` – дескриптор значка, получаемый из функции `LoadIcon`, `LoadOEMIcon` или `LoadStandardIcon`.

```
BOOL DrawIcon (int x, int y, HICON hIcon);
```

В приведенном ниже примере функция загружает и отображает созданный или импортированный в проект значок в центре окна представления. Эта функция вызывает функцию `Win32 API ::GetSystemMetrics`, чтобы получить размер значка для текущего видеорежима. Полученная информация используется для вычисления координат левого верхнего угла значка, размеры которого загружены функцией `LoadIcon` и всегда равны текущим системным размерам `SM_CXICON` и `SM_CYICON`. Если ресурс значка не имеет соответствующего изображения, то функция `LoadIcon` выполнит необходимое масштабирование.

```
void CProgView::DisplayIcon ()
{
    CClientDC ClientDC (this);
    HICON hIcon;
    int IconHeight;
    int IconWidth;
    RECT Rect;

    hIcon = AfxGetApp()->LoadIcon (IDI_ICON1);
    GetClientRect (&Rect);
```

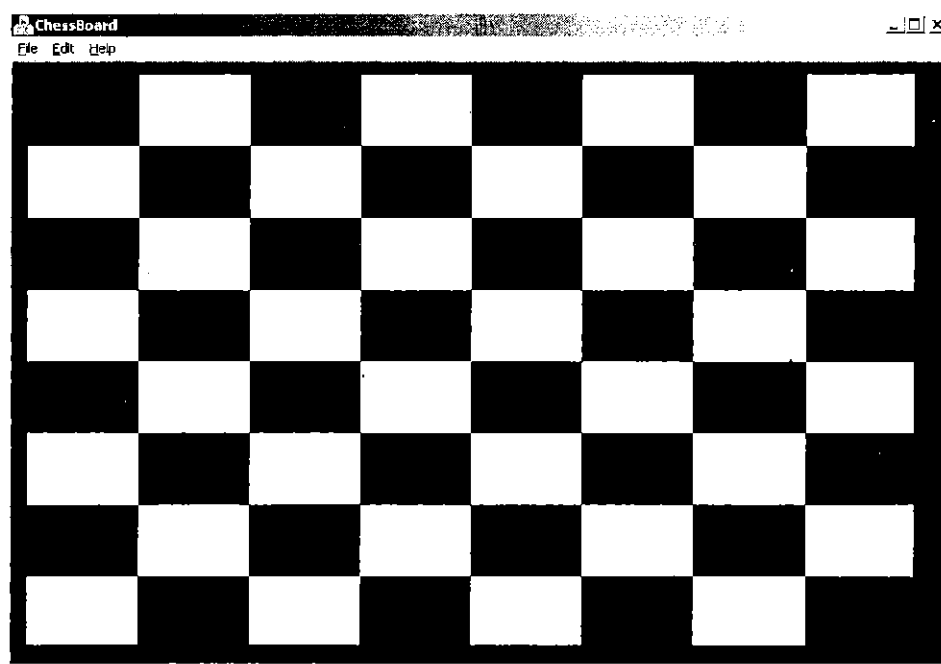
```

IconWidth = ::GetSystemMetrics (SM_CXICON);
IconHeight = ::GetSystemMetrics (SM_CYICON);
ClientDC.DrawIcon
    (Rect.right / 2 - IconWidth / 2,
     Rect.bottom / 2 - IconHeight / 2,
     HIcon);
}

```

## Программа ChessBoard

Разрабатываемая в этом разделе для иллюстрации средств разработки растрового изображения программа ChessBoard отображает растровые изображения в окне представления программы. Программа ChessBoard отображает шахматную доску, заполняющую окно представления. Чтобы окно всегда было заполнено при изменении размера окна, изображение растягивается или сжимается соответствующим образом.



1. Исходные файлы программы сгенерируйте с помощью мастера Application Wizard, присвоив проекту имя ChessBoard. На вкладках окна мастера выберите создание однодокументного приложения, в мастере Application Wizard выберите *те же* установки, что и для программы WinHello в гл. 9, отключив создание панели инструментов и строки состояния.
2. Завершив генерацию исходных файлов, спроектируйте растровое изображение, порождаемое программой. Для этого выберите команду Resource... в меню Insert и тип ресурса Bitmap в диалоговом окне Insert Resource. При щелчке на кнопке New графический редактор создаст пустое растровое изображение и присвоит ему стандартный идентификатор IDB\_BITMAP1, который вы не должны изменять. Далее используйте инструменты и команды графического редактора для создания растрового изображения шахматной доски размером приблизительно 300x300 пикселей.

Можно спроектировать растровое изображение альтернативным методом с помощью программы Paint Windows 95 или другой программы рисования, сохраняющей рисунки в формате растрового изображения. Готовый рисунок сохраните в файле .bmp или .dib и возвратитесь в Visual C++. Вставьте растровое изображение в программу ChessBoard. После этого проверьте, имеет ли растровое изображение идентификатор IDB\_BITMAP1.

3. Для отображения растрового изображения необходимо изменить класс представления программы. Необходимые переменные определите в файле ChessBoardView.h в начале определения класса CChessBoardView. Переменная m\_Bitmap является объектом растрового изображения, а m\_BitmapHeight и m\_BitmapWidth хранят его размеры.

```
class CChessBoardView : public CView
{
protected:
    CBitmap m_Bitmap;
    int m_BitmapHeight;
    int m_BitmapWidth;

protected: // используется только для сериализации
    CChessBoardView();
    DECLARE_DYNCREATE(CChessBoardView)
```

Далее в файле ChessBoardView.cpp эти переменные инициализируются внутри конструктора класса. Причем вызов CBitmap::LoadBitmap инициализирует объект растрового изображения и загружает его. Размеры изображения получают, вызывая функцию CBitmap::GetObject (она возвращает необходимые сведения).

```
CChessBoardView::CChessBoardView()
{
    // TODO: добавьте сюда код конструктора
    BITMAP BM;

    m_Bitmap.LoadBitmap (IDB_BITMAP1);
    m_Bitmap.GetObject (sizeof (BM), &BM);
    m_BitmapWidth = BM.bmWidth;
    m_BitmapHeight = BM.bmHeight;
}
```

4. Далее, для отображения растрового изображения, добавьте приведенный ниже фрагмент функции OnDraw в файл ChessBoardView.cpp. Функция OnDraw отображает растровое изображение так же, как и функция DisplayBitmap, рассмотренная в параграфе “Отображение растрового изображения”. Однако чтобы скопировать растровое изображение в окно представления при необходимости и сжать или растянуть его (оно в точности должно разместиться внутри окна) функция OnDraw использует функцию StretchBlt (вместо использования функции BitBlt для перемещения растрового изображения без изменения его размеров).

```
void CChessBoardView::OnDraw(CDC* pDC)
{
    CChessBoardDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    // TODO: добавьте сюда код отображения собственных данных
    CDC MemDC;
    RECT ClientRect;
```



```

// Создаем объект памяти контекста устройства и выбираем
// в нем объект растрового изображения
MemDC.CreateCompatibleDC (NULL);
MemDC.SelectObject (&m_Bitmap);

// Устанавливаем текущие размеры окна представления
GetClientRect (&ClientRect);

// Отображаем растровое изображение, согласованное с окном
// представления
pDC->StretchBlt (0, 0,
                ClientRect.right,
                ClientRect.bottom,
                &MemDC,
                0, 0,

                m_BitmapWidth,
                m_BitmapHeight,
                SRCCOPY);
}

```

5. В файле ChessBoard.cpp добавьте в функцию InitInstance обычный вызов функции SetWindowText.

```

m_pMainWnd->ShowWindow(SW_SHOW);
m_pMainWnd->UpdateWindow();

m_pMainWnd->SetWindowText ("ChessBoard");
return TRUE;
}

```

Программа ChessBoard готова к построению и выполнению.

## Текст программы ChessBoard

Исходный текст программы ChessBoard размещен в листингах 20.1—20.8.

---

### Листинг 20.1.

```

// ChessBoard.h : главный заголовочный файл приложения ChessBoard
//
#pragma once

#ifdef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"          // основные символы

// CChessBoardApp:
// Смотрите реализацию этого класса в файле ChessBoard.cpp
//

class CChessBoardApp : public CWinApp

```

```

{
public:
    CChessBoardApp();

    // Переопределения
public:
    virtual BOOL InitInstance();

    // Реализация
    afx_msg void OnAppAbout();
    DECLARE_MESSAGE_MAP()
};

extern CChessBoardApp theApp;

```

---

## Листинг 20.2.

```

// ChessBoard.cpp : Определяет поведение классов приложения
//

#include "stdafx.h"
#include "ChessBoard.h"
#include "MainFrm.h"

#include "ChessBoardDoc.h"
#include "ChessBoardView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CChessBoardApp

BEGIN_MESSAGE_MAP(CChessBoardApp, CWinApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
    // Стандартные команды работы с файлами документов
    ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
    // Стандартная команда подготовки к печати
    ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()

// Конструктор CChessBoardApp

CChessBoardApp::CChessBoardApp()
{
    // TODO: поместите сюда собственный код конструктора
    // Поместите весь существенный код инициализации
    // в функцию InitInstance
}

// Единственный объект класса CChessBoardApp

CChessBoardApp theApp;

// Инициализация CChessBoardApp

```

```

BOOL CChessBoardApp::InitInstance()
{
    CWinApp::InitInstance();

    // Инициализация библиотек OLE
    if (!AfxOleInit())
    {
        AfxMessageBox(IDP_OLE_INIT_FAILED);
        return FALSE;
    }
    AfxEnableControlContainer();
    // Стандартная инициализация.
    // Если вы не используете какие-то из возможностей и хотите
    // уменьшить размер оконечного исполняемого модуля,
    // удалите отдельные процедуры инициализации
    // из последующего кода.
    // Измените строку-аргумент функции (ключ в реестре,
    // под которым хранятся в реестре ваши установки).
    // TODO: измените эту строку на что-нибудь подходящее,
    // например, название вашей компании или организации
    SetRegistryKey(_T("Local AppWizard-Generated Applications"));
    LoadStdProfileSettings(4); // Загрузка установок из INI-файла
                               // (включая MRU)
    // Регистрация шаблонов документов приложения. Шаблоны
    // документов служат связью между документами, окнами
    // документов и окнами представлений
    CSingleDocTemplate* pDocTemplate;
    pDocTemplate = new CSingleDocTemplate(
        IDR_MAINFRAME,
        RUNTIME_CLASS(CChessBoardDoc),
        RUNTIME_CLASS(CMainFrame), // главное окно
                                   // SDI-приложения
        RUNTIME_CLASS(CChessBoardView));
    AddDocTemplate(pDocTemplate);
    // Поиск в командной строке команд управления, DDE,
    // открытия файлов
    CCommandLineInfo cmdInfo;
    ParseCommandLine(cmdInfo);
    // Выполнение команд, указанных в командной строке. Вернет
    // FALSE, если приложение запускалось с /RegServer, /Register,
    // /Unregserver или /Unregister.
    if (!ProcessShellCommand(cmdInfo))
        return FALSE;
    // Прорисовка и обновление единственного
    // проинициализированного окна
    m_pMainWnd->ShowWindow(SW_SHOW);
    m_pMainWnd->UpdateWindow();

    m_pMainWnd->SetWindowText ("ChessBoard");
    return TRUE;
}

// CAboutDlg диалог, используемый в App About

class CAboutDlg : public CDialog
{

```

```

public:
    CAboutDlg();

// Данные для диалога
    enum { IDD = IDD_ABOUTBOX };

protected:
    virtual void DoDataExchange(CDataExchange* pDX);
    // DDX/DDV поддержка

// Реализация
protected:
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
END_MESSAGE_MAP()

// Команда приложения на запуск диалога
void CChessBoardApp::OnAppAbout()
{
    CAboutDlg aboutDlg;
    aboutDlg.DoModal();
}

// Обработчики сообщений класса CChessBoardApp

```

---

### Листинг 20.3.

```

// ChessBoardDoc.h : интерфейс класса CChessBoardDoc
//

#pragma once

class CChessBoardDoc : public CDocument
{
protected: // используется только для сериализации
    CChessBoardDoc();
    DECLARE_DYNCREATE(CChessBoardDoc)

// Атрибуты
public:

// Операции
public:

// Переопределения

```

```

        public:
        virtual BOOL OnNewDocument();
        virtual void Serialize(CArchive& ar);

// Реализация
public:
    virtual ~CChessBoardDoc();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Сгенерированные функции карты сообщений
protected:
    DECLARE_MESSAGE_MAP()
};

```

---

#### Листинг 20.4.

```

// ChessBoardDoc.cpp : реализация класса CChessBoardDoc
//

#include "stdafx.h"
#include "ChessBoard.h"

#include "ChessBoardDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CChessBoardDoc

IMPLEMENT_DYNCREATE(CChessBoardDoc, CDocument)

BEGIN_MESSAGE_MAP(CChessBoardDoc, CDocument)
END_MESSAGE_MAP()

// Конструктор/деструктор CChessBoardDoc

CChessBoardDoc::CChessBoardDoc()
{
    // TODO: добавьте сюда одноразовый код конструктора
}

CChessBoardDoc::~CChessBoardDoc()
{
}

BOOL CChessBoardDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;
}

```

```

        // TODO: добавьте сюда код реинициализации
        // (SDI-документы будут многократно использовать
        // этот документ)

        return TRUE;
    }

    // Сериализация класса CChessBoardDoc

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

// Диагностика CChessBoardDoc

#ifdef _DEBUG
void CChessBoardDoc::AssertValid() const
{
    CDocument::AssertValid();
}

void CChessBoardDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}
#endif // _DEBUG

// Команды класса CChessBoardDoc

```

---

## Листинг 20.5.

```

// ChessBoardView.h : интерфейс класса CChessBoardView
//

#pragma once

class CChessBoardView : public CView
{
protected:
    CBitmap m_Bitmap;
    int m_BitmapHeight;
    int m_BitmapWidth;

protected: // используется только для сериализации
    CChessBoardView();
    DECLARE_DYNCREATE(CChessBoardView)

```

```

// Атрибуты
public:
    CChessBoardDoc* GetDocument() const;

// Операции
public:

// Переопределения
public:
    virtual void OnDraw(CDC* pDC);
    // переопределена для прорисовки этого вида
virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
protected:
    virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
    virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
    virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);

// Реализация
public:
    virtual ~CChessBoardView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Сгенерированные функции карты сообщений
protected:
    DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG // отладочная версия в файле ChessBoardView.cpp
inline CChessBoardDoc* CChessBoardView::GetDocument() const
{ return reinterpret_cast<CChessBoardDoc*>(m_pDocument); }
#endif

```

---

## Листинг 20.6.

```

// ChessBoardView.cpp : реализация класса CChessBoardView
//

#include "stdafx.h"
#include "ChessBoard.h"

#include "ChessBoardDoc.h"
#include "ChessBoardView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CChessBoardView

IMPLEMENT_DYNCREATE(CChessBoardView, CView)

```

```

BEGIN_MESSAGE_MAP(CChessBoardView, CView)
    // Стандартные команды печати
    ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
END_MESSAGE_MAP()

// Конструктор/деструктор класса CChessBoardView

CChessBoardView::CChessBoardView()
{
    // TODO: добавьте сюда код конструктора
    BITMAP BM;

    m_Bitmap.LoadBitmap (IDB_BITMAP1);
    m_Bitmap.GetObject (sizeof (BM), &BM);
    m_BitmapWidth = BM.bmWidth;
    m_BitmapHeight = BM.bmHeight;
}

CChessBoardView::~CChessBoardView()
{
}

BOOL CChessBoardView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Измените класс или структуры окна здесь,
    // изменяя и добавляя поля структуры cs

    return CView::PreCreateWindow(cs);
}

// Прорисовка класса CChessBoardView

void CChessBoardView::OnDraw(CDC* pDC)
{
    CChessBoardDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    // TODO: добавьте сюда код отображения собственных данных
    CDC MemDC;
    RECT ClientRect;

    // Создаем объект памяти контекста устройства и выбираем
    // в нем объект растрового изображения
    MemDC.CreateCompatibleDC (NULL);
    MemDC.SelectObject (&m_Bitmap);

    // Устанавливаем текущие размеры окна представления
    GetClientRect (&ClientRect);

    // Отображаем растровое изображение,
    // согласованное с окном представления
    pDC->StretchBlt (0, 0,
        ClientRect.right,
        ClientRect.bottom,

```



```

        &MemDC,
        0, 0,

        m_BitmapWidth,
        m_BitmapHeight,
        SRCCOPY);
    }

    // Печать CChessBoardView

    BOOL CChessBoardView::OnPreparePrinting(CPrintInfo* pInfo)
    {
        // подготовка по умолчанию
        return DoPreparePrinting(pInfo);
    }

    void CChessBoardView::OnBeginPrinting(    CDC*        /*pDC*/,
                                             CPrintInfo*    /*pInfo*/)
    {
        // TODO: добавьте сюда дополнительную инициализацию
    }

    void CChessBoardView::OnEndPrinting(    CDC*        /*pDC*/,
                                           CPrintInfo*    /*pInfo*/)
    {
        // TODO: добавьте очистку после печати
    }

    // Диагностика CChessBoardView

#ifdef _DEBUG
    void CChessBoardView::AssertValid() const
    {
        CView::AssertValid();
    }

    void CChessBoardView::Dump(CDumpContext& dc) const
    {
        CView::Dump(dc);
    }

    CChessBoardDoc* CChessBoardView::GetDocument() const
    // не отладочная версия встроена
    {
        ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CChessBoardDoc)));
        return (CChessBoardDoc*)m_pDocument;
    }
#endif // _DEBUG

    // Обработчики сообщений класса CChessBoardView

```

---

## Листинг 20.7.

```

// MainFrm.h : интерфейс класса CMainFrame
//

```

```

#pragma once
class CMainFrame : public CFrameWnd
{

protected: // используется только для сериализации
    CMainFrame();
    DECLARE_DYNCREATE(CMainFrame)

// Атрибуты
public:

// Операции
public:

// Переопределения
public:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);

// Реализация
public:
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

// Сгенерированные функции карты сообщений
protected:
    DECLARE_MESSAGE_MAP()
};

```

---

## Листинг 20.8.

```

// MainFrm.cpp : реализация класса CMainFrame
//

#include "stdafx.h"
#include "ChessBoard.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
END_MESSAGE_MAP()

// Конструктор/деструктор CMainFrame

CMainFrame::CMainFrame()
{

```

```

        // TODO: добавьте сюда код конструктора
    }

CMainFrame::~CMainFrame()
{
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    if( !CFrameWnd::PreCreateWindow(cs) )
        return FALSE;
    // TODO: Измените класс или стили окна здесь,
    // изменяя и добавляя поля структуры cs

    return TRUE;
}

// Диагностика CMainFrame

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
    CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
    CFrameWnd::Dump(dc);
}

#endif // _DEBUG

// Обработчики сообщений класса CMainFrame

```

## Резюме

---

Мы рассмотрели, как создаются и отображаются растровые изображения, и как используются функции битовых операций для перемещения и манипулирования блоками графических данных.

- *Растровое изображение* сохраняет в памяти или файле точный “слепок” изображения, записывая состояние каждого пикселя, используемого для создания изображения на конкретном устройстве. Чтобы создать растровое изображение, сначала объявите экземпляр MFC-класса `CBitmap`, а затем вызовите для инициализации объекта соответствующую функцию этого класса.
- *Инициализация растрового изображения.* Объект растрового изображения можно инициализировать, вызывая функцию `CBitmap::LoadBitmap` для загрузки растрового изображения из ресурса программы. Чтобы воспользоваться этим методом, необходимо иметь растровое изображение, спроектированное в графическом редакторе Visual C++, или импортированное из какого-либо файла. В качестве альтернативного способа можно инициализировать пустое растровое изображение, вызвав функцию `CBitmap::CreateCompatibleBitmap` и задав необходимый размер изображения. Затем можно нарисовать нужный рисунок внутри этого изображения, выбирая объект растрового изображения внутри объекта контекста устройства памяти и применяя любые функции рисования класса `CDC` к этому объекту.

- *Отображение растрового изображения.* Растровое изображение можно отобразить внутри окна или на другом устройстве, передавая изображение в объект памяти контекста устройства. Затем, используя функцию `CDC::BitBlt`, можно переместить область растрового изображения в окно или другое устройство. Можно вызвать функцию `CDC::PatBlt`, чтобы нарисовать прямоугольную область, используя текущий шаблон, т.е. текущую выбранную кисть в объекте контекста устройства. В функцию `PatBlt` передается код растровой операции, задающий способ объединения пикселей шаблона с пикселями в области приемника. Для перемещения блока графических данных вызывается функция `CDC::BitBlt`. Код растровой операции, передаваемый в функцию `BitBlt`, задает способ объединения пикселей устройства-источника, текущего шаблона и устройства-приемника. Функция `CDC::StretchBlt` предоставляет те же возможности, что и функция `BitBlt`, а также позволяет изменять размер графического блока или сжимать блок в горизонтальном или вертикальном направлении при переносе.

# Глава 21

## Печать и предварительный просмотр

---

- Простая печать и предварительный просмотр
- Многостраничная печать
- Текст программы ScratchBook

Эта глава посвящена печати текстов и графических изображений, а также предварительному просмотру внешнего вида документа перед печатью. Вы узнаете, как выполняются стандартные команды Print..., Print Preview и Print Setup из меню File. Так как в Windows применяется модель вывода данных, не зависящая от устройств, для отображения текста и графики на печатной странице можно использовать уже известные вам способы. Рассмотрим специфические аспекты печати – выбор и установку принтера, разбиение документа на страницы и другие действия, необходимые для работы принтеров. В главе описано, как обеспечивается поддержка печати, позволяющая программе печатать или предварительно просматривать одну страницу. Показаны более совершенные способы печати или просмотра страниц документа, который не помещается на одну страницу. В программе ScratchBook приведены примеры добавления средств поддержки печати и предварительного просмотра.

### ***Простая печать и предварительный просмотр***

---

Мастер Application Wizard при генерации новой программы позволяет включить в нее основные средства поддержки печати и предварительного просмотра, установив опцию Printing And Print Preview на вкладке Advanced Features в диалоговом окне Application Wizard, вследствие чего в меню File программы добавляются команды Print..., Print Preview и Print Setup... При реализации мастером Application Wizard команда Print... печатает ту часть документа, которая помещается на одной странице. Оставшаяся часть документа игнорируется. Подобным образом команда Print Preview отображает распечатку, появляющуюся на одной странице документа. Как вы увидите, команды Print... и Print Preview вызывают функцию OnDraw, чтобы сгенерировать реально выводимую текстовую или графическую информацию. Команда Print Setup... отображает обычное диалоговое окно Print Setup, позволяющее выбрать тип принтера и задать его установки. В этом разделе в программу ScratchBook добавлены все средства печати, позволяющее печатать так же, как и при выборе опции Printing And Print Preview в первом варианте программы, сгенерированном мастером Application Wizard. В новую версию программы ScratchBook включены все средства, которые вошли в программу (см. гл. 19).

Начать следует с *модификации ресурсов программы*. Чтобы отобразить ресурсы программы, перед началом модификации ресурсов откройте проект ScratchBook и вкладку Resource View. Затем откройте меню IDR\_MAINFRAME в редакторе меню. Непосредственно под существующей командой Save As... в меню File добавьте разделитель и команды Print..., Print Preview и Print Setup.... Свойства каждого из этих элементов меню показаны в табл. 21.1.

В редакторе горячих клавиш откройте таблицу IDR\_MAINFRAME, чтобы задать сочетание клавиш для команды Print... (Ctrl+P). Используйте способ, описанный ранее в гл. 10, и добавьте акселератор с идентификатором ID\_FILE\_PRINT и сочетанием клавиш Ctrl+P.

Теперь можно переходить к *модификации текста программы*.

1. Первый шаг модификации текста программы состоит в создании обработчика для новой команды Print Setup..., добавленной в меню File. Его не нужно писать самостоятельно, так как класс

CWinApp предоставляет обработчик, называемый OnFilePrintSetup. Однако MFC не добавляет его в схему сообщений, т. е. текущая функция не получает управление при выборе команды Print Setup... Следовательно, его необходимо вручную добавить в схему сообщений класса приложения ScratchBook. Для этого откройте файл ScratchBook.cpp и добавьте в определение схемы обработки сообщений операторы, выделенные полужирным шрифтом.

Табл. 21.1. Свойства новых элементов меню File

Идентификатор	Элемент меню	Подсказка	Другие свойства
ID_FILE_PRINT	&Print...\tCtrl+P	Print the Document	Разделитель
ID_FILE_PRINT_PREVIEW	Print Pre&view	Display full pages	
ID_FILE_PRINT_SETUP	P&rint Setup...	Change the printer and printing options	

```
BEGIN_MESSAGE_MAP(CScratchBookApp, CWinApp)
ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
// Стандартные команды работы с файлами документов
ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
```

Добавление в схему сообщений данной записи приводит к тому, что при выборе команды Print Setup... будет вызываться функция CWinApp::OnFilePrintSetup. Функция OnFilePrintSetup отображает диалоговое окно Print Setup, в котором нужно выбрать тип принтера и установить его параметры. Это все, что необходимо для поддержки команды Print Setup... Подобным образом класс CView предоставляет обработчики сообщений для стандартных команд меню Print и Print Preview.

2. Обработчики стандартных команд меню Print и Print Preview необходимо активировать, добавив их в схему сообщений для класса представления программы ScratchBook. Откройте файл ScratchBookView.cpp и добавьте в конце схемы обработки сообщений следующие две записи.

```
BEGIN_MESSAGE_MAP(CScratchBookView, CScrollView)
ON_WM_LBUTTONDOWN()
ON_WM_LBUTTONUP()
ON_WM_MOUSEMOVE()
ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview);
END_MESSAGE_MAP()
```

Использованные в этих записях функции CView::OnFilePrint и CView::OnFilePrintPreview управляют операциями печати. Однако OnFilePrint передает результат на принтер, а функция OnFilePrintPreview – в окно предварительного просмотра печати, отображаемое поверх обычного окна программы. При этом она показывает результат в виде одной или двух печатаемых страниц. В процессе управления печатью эти функции вызывают виртуальные функции, определенные внутри класса CView. Реализация виртуальных функций в классе CView по умолчанию накладывает ограничение на процесс печати. Для улучшения возможностей печати эти функции можно переопределить (см. ниже).

3. Поддержку печати или предварительного просмотра печати следует реализовать путем переопределения только виртуальной функции OnPreparePrinting. MFC вызывает ее перед

печатью или предварительным просмотром. Чтобы создать функцию `OnPreparePrinting` для программы `ScratchBook`, в окне `Properties` создайте в классе `CScratchBookView` обработчик сообщения `OnPreparePrinting`. В созданной функции `OnPreparePrinting` *удалите* вызов основной версии функции `OnPreparePrinting` и добавьте вызов функции `CView::DoPreparePrinting`.

```
BOOL CScratchBookView::OnPreparePrinting(CPrintInfo* pInfo)
{
    // TODO: вызовите DoPreparePrinting для вызова окна Print

    return DoPreparePrinting (pInfo);
}
```

Вызываемая функция `DoPreparePrinting` создает объект контекста устройства, связанный с принтером.

- Если документ *печатается*, то `DoPreparePrinting` отображает обычное диалоговое окно `Print`, позволяющее выбрать определенный принтер и установить ряд опций печати. Затем функция создает объект контекста устройства для выбранного принтера и задает для него выбранные установки.
- При *предварительном просмотре* документа функция `DoPreparePrinting` создает объект контекста устройства для текущего стандартного принтера `Windows`, а затем присваивает стандартные установки принтера, не отображая диалоговое окно `Print`.

Если объект контекста устройства связан с принтером, он *дополнительно* сохраняет установки принтера при сохранении атрибутов и инструментов рисования, описанных в предыдущих главах. Обратите внимание: функцию `OnPreparePrinting` *необходимо* создать, так как ее стандартная версия *ничего* не выполняет. Это может привести к тому, что MFC попытается напечатать или просмотреть документ без наличия корректного объекта контекста устройства. Указатель на объект класса `CPrintInfo` передается во все виртуальные функции выполнения печати. Этот объект содержит информацию о печати и предоставляет функции и переменные, которые используются виртуальными функциями для получения или изменения установок принтера. Например, если известен номер печатаемой страницы документа, то из функции `OnPreparePrinting` можно вызвать функцию `CPrintInfo::SetMaxPage` (*перед* вызовом `DoPreparePrinting`), чтобы задать номер печатаемой страницы. Тогда при печати документа этот номер отобразится в диалоговом окне `Print` (в текстовом поле `To`). При просмотре документа предоставление этого номера приведет к отображению MFC полосы прокрутки в окне предварительного просмотра, что позволит прокручивать страницы документа. Для корректной установки позиции бегунка на полосе прокрутки MFC должно передаваться общее число страниц.

Функция `OnPreparePrinting`, вызванная чтобы подготовить объект контекста устройства для печати или просмотра, передает этот объект в функцию `OnDraw` класса представления. Поскольку объект контекста устройства связывается с *принтером*, а не с окном представления, выводимая графическая информация появляется на печатаемой странице (или в окне предварительного просмотра печати), а не внутри окна представления. Один и тот же программный код рисования внутри функции `OnDraw` имеет возможность отобразить выводимую информацию как в окне представления, так и при печати, поскольку вызываемые для этого функции класса `CDC` аппаратно независимы в достаточной степени. Когда документ печатается или просматривается, MFC готовит объект контекста устройства, связанный с *принтером*, и передает его в функцию `OnDraw`. Однако при просмотре документа объект контекста устройства фактически передается в окно предварительного просмотра. При этом используется отдельный контекст устройства, который связан с окном представления и задает установки для *имитации* страницы, предназначенной для печати.

Внеся все описанные модификации в ресурсы и программу ScratchBook, можно построить и выполнить ее новую версию. Особенности этой программы состоят в следующем.

- Если выбрать команду Print Setup..., программа открывает одноименное диалоговое окно, позволяющее выбрать принтер, принимающий выводимую на печать информацию (если в Windows установлено несколько принтеров), и задать некоторые его параметры. Щелчок на кнопке Properties... в диалоговом окне Print Setup позволяет получить доступ ко всем имеющимся установкам принтера (у разных принтеров он может быть разным).
- Если выбрать команду Print..., программа откроет диалоговое окно Print. В этом диалоговом окне можно выбрать принтер, принимающий выводимую информацию, и указать некоторые параметры печати (качество печати и число копий). Щелчок на кнопке Properties... в диалоговом окне Print позволяет изменить любой из доступных параметров принтера непосредственно перед печатью. Эти же параметры отображаются при щелчке на кнопке Properties... в диалоговом окне Print Setup. Если рисунок печатается не очень быстро, то во время печати будет видно диалоговое окно Printing. В этом окне можно щелкнуть на кнопке Cancel, чтобы остановить работу принтера. Если сделать это до передачи Print Manager выводимой информации, то печать не будет выполняться.
- Если выбрать команду Print Preview, программа отображает окно предварительного просмотра печати, содержащее образ целой печатаемой страницы в масштабе, соответствующем окну программы. Окно предварительного просмотра печати позволяет оценить внешний вид скомпонованной страницы (но не выполнять редактирование рисунка). Для редактирования необходимо вернуться в обычное окно представления, нажав кнопку Close.

## Средства печати в окне представления класса *CEditView*

При наследовании класса представления в программе от класса *CEditView*, рассмотренного в гл. 10, MFC и Windows предоставляют большую часть кода, требуемого для печати. Даже если в мастере Application Wizard не была выбрана опция Printing and Print Preview, можно реализовать команды Print..., Print Preview и Print Setup самому, потратив небольшие усилия на программирование:

- Команда Print... реализуется простым добавлением ее в меню File. Для этого достаточно задать идентификатор `ID_FILE_PRINT_PREVIEW`, без дальнейшего изменения кода. Результирующая команда Print будет печатать весь текст документа (как одно-, так и многостраничного). Создавать функцию *OnPreparePrinting* не нужно.
- Команда Print Preview реализуется простым добавлением ее в меню File. Для этого достаточно присвоить идентификатор `ID_FILE_PRINT_PREVIEW`, затем добавить в схему сообщений оператор `include` для вставки файла `Afxprint.rc` и макрос `ON_COMMAND` для класса представления (см. выше). Создавать функцию *OnPreparePrinting* не нужно.
- Команда Print Setup реализуется в точности так, как это описано выше в этом параграфе для программы ScratchBook.

## Усовершенствованная печать

Полученная в предыдущем разделе версия программы ScratchBook печатает или просматривает только часть рисунка, поместившуюся на одной странице. Оставшаяся часть рисунка игнорируется. В этом разделе возможности программы будут расширены таким образом, чтобы она печатала весь рисунок. Любая часть рисунка, которая не поместилась на одной странице, будет печататься на дополнительных. Как вы увидите, это достигается переопределением некоторых вызываемых при печати виртуальных функций. Заметьте: текущая версия функции *OnDraw* всегда рисует границу справа и внизу рисунка. Однако граница служит только для ограничения рисунка внутри окна



представления. На печатной копии рисунка она не появляется. В этом разделе функция OnDraw модифицируется. В результате: только в случае, если выводимая информация направляется в окно представления, будет отображаться граница.

## Размер рисунка

Размер рисунка в программе ScratchBook устанавливается равным 800 на 600 пикселей (см. гл. 13). Для большинства принтеров рисунок такого размера легко помещается на одной странице. Чтобы продемонстрировать способы печати нескольких страниц, необходимо модифицировать программу ScratchBook для работы с рисунками, размеры которых превышают стандартную печатную страницу. Чтобы выполнить это, сначала определите целочисленные константы для ширины и высоты рисунка в начале файла ScratchBookView.h.

```
// ScratchBookView.h : интерфейс класса CScratchBookView
//
const int DRAWWIDTH = 4000; // ширина рисунка
const int DRAWHEIGHT = 6000; // высота рисунка
```

В файле ScratchBookView.cpp в функции OnInitialUpdate используйте вместо числовых значений (800 и 600) эти константы. Использование констант DRAWWIDTH и DRAWHEIGHT вместо числовых значений облегчает изменение размера рисунка. Функция OnInitialUpdate описана в параграфе “Границы рисунка в окне представления” гл. 13.

```
void CScratchBookView::OnInitialUpdate()
{
    CScrollView::OnInitialUpdate();

    // TODO: Добавьте сюда собственный код или вызов базового
    // класса

    SIZE Size = {DRAWWIDTH, DRAWHEIGHT};
    SetScrollSizes (MM_TEXT, Size);
}
```

Изменяя размер рисунка, необходимо изменить номер версии, используемый для сериализации документа, чтобы не прочитать по ошибке файл, созданный предыдущей версией (или, используя предыдущую версию программы, не прочитать файл, созданный текущей версией). Номера версий рассмотрены в параграфе “Чтение/запись данных” гл. 12. Для модификации программы откройте файл ScratchBookView.cpp и измените номер версии с 2 на 3 в каждом вхождении макроса IMPLEMENT\_SERIAL (вы должны найти восемь вхождений). Например, макрос

```
IMPLEMENT_SERIAL (CFigure, CObject, 2)
```

вы должны изменить на

```
IMPLEMENT_SERIAL (CFigure, CObject, 3)
```

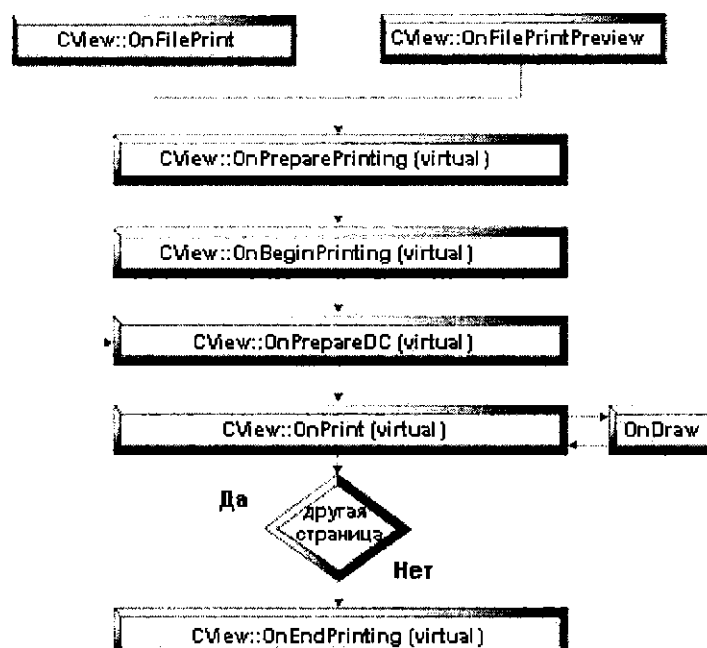
Обычно размер каждого рисунка в программах рисования устанавливается с помощью команды меню Options. Если рисунок сохраняется в файле на диске, то его размер тоже будет сохранен.

Использование *стандартного режима отображения* MM\_TEXT упрощает создание программы. В этом случае размеры текстовых и графических изображений *в значительной степени* зависят от разрешающей способности устройства. Вспомните: в режиме отображения MM\_TEXT все координаты задаются в пикселях. Число пикселей на физический дюйм *различно* и зависит от разрешающей способности устройства. В режиме MM\_TEXT размеры изображений, выведенных на печать, зависят не только от их размеров на экране, но и от типа принтера. Более того, на лазерном принтере и других

принтерах с высоким разрешением размеры печатаемого изображения намного меньше их размеров на экране. Чтобы создать изображения, имеющие одинаковый размер, независимо от выходного устройства, можно использовать один из *альтернативных режимов отображения*: MM\_HIENGLISH, MM\_METRIC, MM\_LOENGLISH, MM\_LOMETRIC и MM\_TWIPS. В каждом из этих режимов координаты указываются в стандартных единицах измерения (дюймах, миллиметрах и так далее), а не в аппаратно зависимых пикселях. Можно использовать один из определяемых пользователем режимов отображения (MM\_ANISOMETRIC или MM\_ISOMETRIC) с заданием собственных единиц измерения. Использование альтернативных режимов изменяет логическую структуру отображающего кода. Подробное рассмотрение этих режимов выходит за рамки книги. Информацию по ним смотрите в справочной системе.

## Переопределение виртуальных функций печати

Когда MFC печатает или просматривает документ, вызываются виртуальные функции, определенные внутри класса CView и предназначенные для выполнения различных задач печати. Чтобы сделать процесс печати более эффективным, можно переопределить одну или несколько из этих функций внутри класса представления программы, и добавить их код. На следующем рисунке показан общий процесс печати и предварительного просмотра, а также место вызова каждой виртуальной функции класса CView внутри процедуры. Это циклический процесс: при печати каждой страницы выполняется один цикл.



Виртуальные функции с описанием выполняемых задач приведены в табл. 21.2. Подробности можно найти в документации на эти функции. Заметим: поскольку MFC вызывает виртуальные функции и при печати, и при просмотре документов, переопределенные функции влияют на оба процесса. Всем, кто создавал программы печати в среде Windows без использования библиотеки MFC, будет приятно узнать, что MFC предоставляет все коды, необходимые для отображения диалогового окна Printing во время печати документа и позволяющие прервать выполнение печати. Окно также содержит обработчик сообщений программы во время печати, позволяющий обработать щелчок на

кнопке Cancel в диалоговом окне Printing. Этот обработчик называется *процедурой преждевременного прекращения (abort procedure)*.

Табл. 21.2. Виртуальные функции класса CView

Имя функции	Задачи, выполняемые переопределенной функцией
OnBeginPrinting	Размещение любых шрифтов, перьев, кистей и других объектов, используемых только при печати. Вычисление и установка длины документа, основанной на объекте контекста устройства. Сохранение любой требуемой информации об объекте контекста устройства (это первая виртуальная функция, которая имеет к нему доступ)
OnEndPrinting	Удаление объектов, размещенных функцией OnBeginPrinting, вызовом функции CGdiObject::DeleteObject
OnPrepareDC	Установка атрибутов текста или рисунка и модификация начала представления для печати текущей страницы. Если длина документа не установлена – завершение цикла печати в конце документа с одновременным присваиванием переменной CPrintInfo::m_bContinuePrinting значения FALSE
OnPreparePrinting	Вызов функции класса CPrintInfo (например, функции CPrintInfo::SetMaxPage для установки длины документа) или установка переменных класса CPrintInfo для выполнения операции просмотра диалогового окна Print. Вызов функции DoPreparePrinting для создания объекта контекста устройства, используемого при печати или предварительном просмотре (обратите внимание: необходимо переопределить функцию OnPreparePrinting и вызвать функцию DoPreparePrinting)
OnPrint	Выбор любых шрифтов или других объектов, размещенных функцией OnBeginPrinting. Последующий вызов функции OnDraw; отмена выбора объектов. Печать колонтитулов, появляющихся только в печатном варианте документа. (Если печатный документ отличается от выведенного на экран, печатайте его с помощью этой функции, а не OnDraw.)

Виртуальную функцию OnPreparePrinting мы уже переопределили. Для реализации многостраничной печати переопределим виртуальные функции OnBeginPrinting и OnPrepareDC. Однако перед этим необходимо добавить в класс представления несколько переменных. Откройте файл ScratchBookView.h и в определение класса CScratchBookView добавьте выделенные в следующем листинге полужирным шрифтом определения переменных. Переменные m\_PageHeight и m\_PageWidth используются для хранения размеров страницы. Переменные m\_NumCols и m\_NumRows используются для хранения количества страниц по горизонтали и вертикали, необходимых для печати всего рисунка.

```
class CScratchBookView : public CScrollView
{
protected:
    CString m_ClassName;
    int m_Dragging;
    HCURSOR m_HCross;
    HCURSOR m_HArrow;
    CPoint m_PointOld;
```

```

CPoint m_PointOrigin;
CPen m_PenDotted;
int m_NumCols, m_NumRows;
int m_PageHeight, m_PageWidth;

```

Чтобы выполнить переопределение виртуальной функции OnBeginPrinting, в окне мастера Properties откройте вкладку Message Maps для класса CScratchBookView, в списке Messages – имя функции OnBeginPrinting и щелкните на кнопке Add Function. Таким же образом переопределите виртуальную функцию OnPrepareDC. Откройте файл CScratchBookView.cpp и добавьте в сгенерированную функцию OnBeginPrinting следующий фрагмент программы.

```

void CScratchBookView::OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo)
{
    // TODO: Добавьте сюда собственный код обработчика или вызов
    // базового класса

    m_PageHeight = pDC->GetDeviceCaps (VERTRES);
    m_PageWidth = pDC->GetDeviceCaps (HORZRES);

    m_NumRows = DRAWHEIGHT / m_PageHeight + (DRAWHEIGHT %
        m_PageHeight > 0);
    m_NumCols = DRAWWIDTH / m_PageWidth + (DRAWWIDTH %
        m_PageWidth > 0);
    pInfo->SetMinPage (1);
    pInfo->SetMaxPage (m_NumRows * m_NumCols);

    CScrollView::OnBeginPrinting(pDC, pInfo);
}

```

Выше в схеме алгоритма показано, что виртуальная функция OnBeginPrinting вызывается только в начале печати *после* создания объекта контекста устройства, но *перед* выполнением цикла, печатающего каждую страницу. Виртуальная функция печати OnBeginPrinting первой получает доступ к объекту контекста устройства. Фрагмент программы, добавленный в функцию OnBeginPrinting, использует объект контекста устройства для вызова функции GetDeviceCaps класса CDC, чтобы получить размеры доступной для печати области страницы.

- Когда передается значение параметра VERTRES, функция GetDeviceCaps возвращает *высоту* доступной для печати области страницы в пикселях (значение высоты хранится в переменной m\_PageHeight).
- Когда передается значение параметра HORZRES, функция GetDeviceCaps возвращает *ширину* в пикселях области страницы, доступной для печати (хранится в переменной m\_PageWidth).

Размеры печатной страницы и рисунка затем используются функцией OnBeginPrinting, чтобы вычислить число страниц, требуемых для печати всего рисунка. Сначала вычисляется число страниц в вертикальном направлении, и значение сохраняется в переменной m\_NumRows, после чего вычисляется число страниц по горизонтали, а результат сохраняется в переменной m\_NumCols. В обоих случаях при вычислении числа страниц используется оператор % для округления результата деления до ближайшего меньшего целого. Произведение m\_NumRows \* m\_NumCols позволяет вычислить *общее* число страниц, необходимое для печати рисунка.

В заключение функция OnBeginPrinting передает общее число страниц MFC и вызывает функцию CPrintInfo::SetMinPage, чтобы задать номер первой страницы (1), и CPrintInfo::SetMaxPage, чтобы указать номер последней страницы (m\_NumRows \* m\_NumCols). MFC напечатает указанное

число страниц, т.е. вызовет для каждой указанной страницы виртуальные функции OnPrepareDC и OnPrint (см. схему алгоритма выше).

Выбрать ограниченный диапазон страниц, подлежащих печати, внутри диапазона, заданного с помощью функций SetMinPage и SetMaxPage, можно в диалоговом окне Print. В этом случае MFC печатает ограниченный диапазон страниц. Заметим: если для установки максимального числа страниц не вызывается функция SetMaxPage, то функцию OnPrepareDC, рассмотренную ниже, необходимо для прерывания цикла печати определить вручную, присваивая после печати последней страницы значение FALSE переменной CPrintInfo::m\_bContinuePrinting.

В сгенерированную функцию OnPrepareDC в файле CScratchBookView.cpp необходимо добавить следующий фрагмент.

```
void CScratchBookView::OnPrepareDC(CDC* pDC, CPrintInfo* pInfo)
{
    // TODO: Добавьте сюда собственный код обработчика или вызов
    // базового класса

    CScrollView::OnPrepareDC(pDC, pInfo);

    if (pInfo == NULL) return;

    int CurRow = pInfo->m_nCurPage / m_NumCols +
        (pInfo->m_nCurPage % m_NumCols > 0);
    int CurCol = (pInfo->m_nCurPage - 1) % m_NumCols + 1;

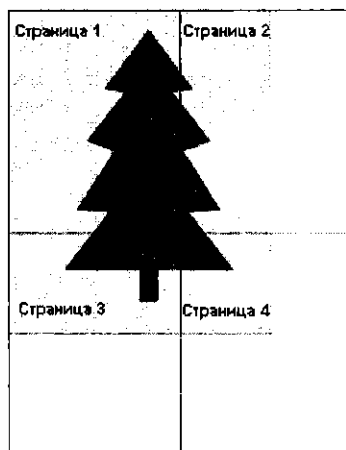
    pDC->SetViewportOrg (-m_PageWidth * (CurCol - 1),
        -m_PageHeight * (CurRow - 1));
}
```

Выше в схеме алгоритма показано, что MFC вызывает функцию OnPrepareDC перед печатью каждой страницы, поэтому очень важно подготовить объект контекста устройства для печати текущей страницы. MFC *также* вызывает функцию OnPrepareDC непосредственно перед вызовом OnDraw (см. гл. 13), чтобы перерисовать окно представления. В этом случае необходимо согласовать начальную позицию просмотра с текущей позицией прокрутки документа (класс представления поддерживает прокрутку, поскольку наследуется из класса CScrollView).

- Если вызвать функцию OnPrepareDC для *перерисовки окна представления*, то вызов CScrollView::OnPrepareDC в начале функции настраивает объект контекста устройства на текущую позицию прокрутки (в случае необходимости). После этого указателю pInfo присваивается значение NULL и происходит выход из функции OnPrepareDC.
- Если вызвать функцию OnPrepareDC для *печати страницы*, то такой вызов не вносит изменений в объект контекста устройства. Однако указатель pInfo в этом случае содержит адрес объекта CPrintInfo (предоставляющего информацию для печати), а добавленный фрагмент программы настраивает объект контекста устройства так, чтобы функция OnDraw печатала на текущей странице *следующую часть* рисунка.

Добавленный в функцию OnPrepareDC фрагмент размещает объект контекста устройства, используя *тот же* метод, который использовался в классе CScrollView при прокрутке документа в окне представления, т.е. настраивает начало представления (см. параграф “Логические и фактические координаты” гл. 13). Перед печатью каждой новой страницы функция OnPrepareDC настраивает начало представления, сдвигая позиции фигур относительно страницы, чтобы при вызове функции OnDraw печаталась *следующая часть* документа. При печати первой страницы начало представления устанавливается в (0, 0) (значение по умолчанию), так что печатается та часть документа, которая

расположена в его левом верхнем углу. При печати следующей страницы функция OnPrepareDC вычитает ширину страницы из горизонтальной установки начала представления, что приводит к печати следующей порции рисунка справа. Таким образом, строка за строкой, продолжается печать всего документа, как показано на следующем рисунке.



Передаваемая в функцию OnPrepareDC переменная m\_nCurPage класса CPrintInfo содержит номер текущей печатаемой страницы. Функция OnPrepareDC использует это значение вместе с переменной m\_NumCols, установленной функцией OnBeginPrinting, для вычисления строки (CurRow) и столбца (CurCol) позиции части рисунка, печатаемой на текущей странице. Далее значения CurRow и CurCol вместе с размерами страницы, сохраненными в переменных m\_PageWidth и m\_PageHeight, используются указанной функцией для вычисления передаваемых в функцию SetViewportOrg класса CDC новых координат начала представления.

## Модификация функции OnDraw

Выше в схеме алгоритма показано, что после вызова функции OnPrepareDC MFC вызывает виртуальную функцию OnPrint. Стандартная реализация этой функции вызывает функцию OnDraw класса представления программы, передавая ей объект контекста устройства, созданный функцией OnPreparePrinting и подготовленный функцией OnPrepareDC. Поскольку объект контекста устройства связан с принтером, результат, сгенерированный функцией OnDraw, автоматически передается принтеру (или в окно предварительного просмотра печати), а не в окно представления. Кроме того, поскольку начало представления согласовано функцией OnPrepareDC с объектом контекста устройства, функция OnDraw автоматически печатает корректную часть рисунка (т.е. часть, появляющуюся на текущей странице). Печатается часть рисунка, соответствующая физической странице принтера, и, следовательно, зависящая от текущих координат начала представления, установленных в функции OnPrepareDC. Другие части рисунка *отсекаются* (т.е. отбрасываются, так как выходят за границу физической страницы). Функция OnDraw не вызывает функции рисования для любых фигур, находящихся за пределами страницы. После вызова функции GetClipBox передаются логические координаты области рисунка, появляющейся на физической странице. Для любой из фигур, находящихся полностью за пределами страницы, функция OnDraw не вызывает функцию класса CFigureDraw.

Функция OnDraw программы ScratchBook отображает границу только при передаче выводимой информации в окно представления. Чтобы предотвратить рисование границы при печати, поместите в блок if операторы рисования линий, приведенные ниже. Если функции GetDeviceCap передать константу TECHNOLOGY, то она возвратит код, указывающий тип устройства, связанного с объектом контекста устройства. Код DT\_RASDISPLAY показывает, что это устройство – экран. Если функция

OnDraw получает этот код, то результат передается в окно представления (а не на принтер или в окно предварительного просмотра) и создается граница рисунка.

```
void CScratchBookView::OnDraw(CDC* pDC)
{
    CScratchBookDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    // TODO: вставьте сюда код прорисовки собственных данных

    if (pDC->GetDeviceCaps (TECHNOLOGY) == DT_RASDISPLAY)
    {
        CSize ScrollSize = GetTotalSize ();
        pDC->MoveTo (ScrollSize.cx, 0);
        pDC->LineTo (ScrollSize.cx, ScrollSize.cy);
        pDC->LineTo (0, ScrollSize.cy);
    }
}
```

В общем, этот способ можно использовать при настройке выводимой информации, сгенерированной функцией OnDraw, на целевое устройство.

Обратитесь к разделам справочной системы, описывающим функцию GetDeviceCaps, чтобы получить описание различных кодов, которые возвращает эта функция при передаче константы TECHNOLOGY. GetDeviceCaps – это функция класса CDC, которая вызывается для получения помощи при управлении печатью, начиная с момента получения объекта контекста устройства для принтера. Обычно она вызывается из определенных пользователем виртуальных функций OnBeginPrinting, OnPrepareDC или OnPrint, или из стандартной функции OnDraw класса представления. Вам уже знакомо использование функции GetDeviceCaps для определения типа устройства вывода (передачей аргумента TECHNOLOGY) и получения размера доступной для печати области страницы в пикселях (передачей HORZRES и VERTRES).

Можно получить ширину области, доступной для печати (в *миллиметрах*), передавая константу HORZSIZE, а высоту – передавая константу VERTSIZE. Обратите внимание: функция GetDeviceCap возвращает размеры той части страницы, на которой реально выполняется печать. Размеры, передаваемые функцией GetDeviceCaps, обычно меньше физического размера страницы, поскольку многие принтеры (например, лазерные) не могут печатать до самого края страницы.

Дисплей поддерживает все основные операции рисования и отображения (см. гл. 19 и 20). Однако принтер или плоттер могут не соответствовать этим требованиям. Чтобы узнать, выполняема ли требуемая операция на данном принтере, вызовите функцию GetDeviceCaps. Если операция – растровая, то в функцию GetDeviceCaps (см. гл. 20) передается константа RASTERCAPS. Если возвращаемый код содержит значение RC\_BITBLT, то устройство поддерживает функции PatBlt и BitBlt. Если же он содержит значение RC\_STRETCHBLT, то поддерживается функция StretchBlt. Тестирование на выполнение растровых операций можно выполнить, например, так:

```
int Caps;

Caps = pDC->GetDeviceCaps (RASTERCAPS);

if (Caps & RC_BITBLT)
    // теперь можно вызвать 'PatBlt' и 'BitBlt'

if (Caps & RC_STRETCHBLT)
    // теперь можно вызвать 'StretchBlt'
```

Возможности принтера при рисовании графических изображений можно определить, передавая в функцию GetDeviceCaps константы CURVECAPS (рисование кривых) или POLYGONCAPS

(рисование прямоугольников и других многоугольников). Полное описание информации, возвращаемой при передаче этих и других индексов в функцию `GetDeviceCaps`, смотрите в справочной системе.

В разработанной версии программы `ScratchBook` рисунок несколько больше, чем в предыдущей версии и для доступа к различным его частям используются полосы прокрутки. При выборе команды `Print...` печатаются *все* страницы рисунка. При печати на принтере с разрешением 300 на 300 пикселей рисунок занимает четыре страницы. Просмотреть *все* страницы, требуемые для печати рисунка, можно, выбрав команду `Print Preview` в меню `File`.

## Текст программы *ScratchBook*

---

Исходные тексты последней (и окончательной) версии программы `ScratchBook` содержатся в листингах 21.1—21.8.

---

### Листинг 21.1.

```
// ScratchBook.h : главный заголовочный файл приложения ScratchBook
//

#pragma once

#ifndef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"          // основные символы

// CScratchBookApp:
// Смотрите реализацию этого класса в файле ScratchBook.cpp
//

class CScratchBookApp : public CWinApp
{
public:
    int m_CurrentWidth;
    UINT m_CurrentTool;
    COLORREF m_CurrentColor;
    UINT m_IdxCmd;

public:
    CScratchBookApp();

// Переопределения
public:
    virtual BOOL InitInstance();

// Реализация
    afx_msg void OnAppAbout();
    DECLARE_MESSAGE_MAP()
    afx_msg void OnToolsLine();
    afx_msg void OnToolsRectangle();
    afx_msg void OnUpdateToolsRectangle(CCmdUI *pCmdUI);
    afx_msg void OnToolsFrextangle();
```



```

afx_msg void OnUpdateToolsFrrectangle(CCmdUI *pCmdUI);
afx_msg void OnToolsRrectangle();
afx_msg void OnUpdateToolsRrectangle(CCmdUI *pCmdUI);
afx_msg void OnToolsFrrectangle();
afx_msg void OnUpdateToolsFrrectangle(CCmdUI *pCmdUI);
afx_msg void OnToolsCircle();
afx_msg void OnUpdateToolsCircle(CCmdUI *pCmdUI);
afx_msg void OnToolsFcircle();
afx_msg void OnUpdateToolsFcircle(CCmdUI *pCmdUI);
afx_msg void OnLineSingle();
afx_msg void OnUpdateLineSingle(CCmdUI *pCmdUI);
afx_msg void OnLineDouble();
afx_msg void OnUpdateLineDouble(CCmdUI *pCmdUI);
afx_msg void OnLineTriple();
afx_msg void OnUpdateLineTriple(CCmdUI *pCmdUI);
afx_msg void OnUpdateToolsLine(CCmdUI *pCmdUI);
afx_msg void OnColorBlack();
afx_msg void OnUpdateColorBlack(CCmdUI *pCmdUI);
afx_msg void OnColorWhite();
afx_msg void OnUpdateColorWhite(CCmdUI *pCmdUI);
afx_msg void OnColorRed();
afx_msg void OnUpdateColorRed(CCmdUI *pCmdUI);
afx_msg void OnColorGreen();
afx_msg void OnUpdateColorGreen(CCmdUI *pCmdUI);
afx_msg void OnColorBlue();
afx_msg void OnUpdateColorBlue(CCmdUI *pCmdUI);
afx_msg void OnColorYellow();
afx_msg void OnUpdateColorYellow(CCmdUI *pCmdUI);
afx_msg void OnColorCyan();
afx_msg void OnUpdateColorCyan(CCmdUI *pCmdUI);
afx_msg void OnUpdateColorMagenta(CCmdUI *pCmdUI);
afx_msg void OnColorCustom();
afx_msg void OnUpdateColorCustom(CCmdUI *pCmdUI);
void OnColorMagenta(void);
};

```

---

## Листинг 21.2.

```

// ScratchBook.cpp : Определяет поведение классов приложения
//

#include "stdafx.h"
#include "ScratchBook.h"
#include "MainFrm.h"

#include "ScratchBookDoc.h"
#include "ScratchBookView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// CScratchBookApp

```

```

BEGIN_MESSAGE_MAP(CScratchBookApp, CWinApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
    // Стандартные команды работы с файлами документов
    ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
    ON_COMMAND(ID_TOOLS_LINE, OnToolsLine)
    ON_COMMAND(ID_TOOLS_RECTANGLE, OnToolsRectangle)
    ON_UPDATE_COMMAND_UI(ID_TOOLS_RECTANGLE, OnUpdateToolsRectangle)
    ON_COMMAND(ID_TOOLS_FRECTANGLE, OnToolsFrectangle)
    ON_UPDATE_COMMAND_UI(ID_TOOLS_FRECTANGLE,
        OnUpdateToolsFrectangle)
    ON_COMMAND(ID_TOOLS_RRECTANGLE, OnToolsRrectangle)
    ON_UPDATE_COMMAND_UI(ID_TOOLS_RRECTANGLE,
        OnUpdateToolsRrectangle)
    ON_COMMAND(ID_TOOLS_FRRECTANGLE, OnToolsFrrectangle)
    ON_UPDATE_COMMAND_UI(ID_TOOLS_FRRECTANGLE,
        OnUpdateToolsFrrectangle)
    ON_COMMAND(ID_TOOLS_CIRCLE, OnToolsCircle)
    ON_UPDATE_COMMAND_UI(ID_TOOLS_CIRCLE, OnUpdateToolsCircle)
    ON_COMMAND(ID_TOOLS_FCIRCLE, OnToolsFcircle)
    ON_UPDATE_COMMAND_UI(ID_TOOLS_FCIRCLE, OnUpdateToolsFcircle)
    ON_COMMAND(ID_LINE_SINGLE, OnLineSingle)
    ON_UPDATE_COMMAND_UI(ID_LINE_SINGLE, OnUpdateLineSingle)
    ON_COMMAND(ID_LINE_DOUBLE, OnLineDouble)
    ON_UPDATE_COMMAND_UI(ID_LINE_DOUBLE, OnUpdateLineDouble)
    ON_COMMAND(ID_LINE_TRIPLE, OnLineTriple)
    ON_UPDATE_COMMAND_UI(ID_LINE_TRIPLE, OnUpdateLineTriple)
    ON_UPDATE_COMMAND_UI(ID_TOOLS_LINE, OnUpdateToolsLine)
    ON_COMMAND(ID_COLOR_BLACK, OnColorBlack)
    ON_UPDATE_COMMAND_UI(ID_COLOR_BLACK, OnUpdateColorBlack)
    ON_COMMAND(ID_COLOR_WHITE, OnColorWhite)
    ON_UPDATE_COMMAND_UI(ID_COLOR_WHITE, OnUpdateColorWhite)
    ON_COMMAND(ID_COLOR_RED, OnColorRed)
    ON_UPDATE_COMMAND_UI(ID_COLOR_RED, OnUpdateColorRed)
    ON_COMMAND(ID_COLOR_GREEN, OnColorGreen)
    ON_UPDATE_COMMAND_UI(ID_COLOR_GREEN, OnUpdateColorGreen)
    ON_COMMAND(ID_COLOR_BLUE, OnColorBlue)
    ON_UPDATE_COMMAND_UI(ID_COLOR_BLUE, OnUpdateColorBlue)
    ON_COMMAND(ID_COLOR_YELLOW, OnColorYellow)
    ON_UPDATE_COMMAND_UI(ID_COLOR_YELLOW, OnUpdateColorYellow)
    ON_COMMAND(ID_COLOR_CYAN, OnColorCyan)
    ON_UPDATE_COMMAND_UI(ID_COLOR_CYAN, OnUpdateColorCyan)
    ON_COMMAND(ID_COLOR_MAGENTA, OnColorMagenta)
    ON_UPDATE_COMMAND_UI(ID_COLOR_MAGENTA, OnUpdateColorMagenta)
    ON_COMMAND(ID_COLOR_CUSTOM, OnColorCustom)
    ON_UPDATE_COMMAND_UI(ID_COLOR_CUSTOM, OnUpdateColorCustom)
    ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()

// Конструктор CScratchBookApp

CScratchBookApp::CScratchBookApp()
{
    // TODO: добавьте сюда код конструктора.
    // Поместите весь существенный код инициализации
    // в функцию InitInstance

```

```

        m_CurrentColor = RGB (0, 0, 0);
        m_CurrentWidth = 1;
        m_CurrentTool=ID_TOOLS_LINE;
        m_IdxCmd = ID_COLOR_BLACK;
    }

    // Единственный объект класса CScratchBookApp

CScratchBookApp theApp;

// Инициализация CScratchBookApp

BOOL CScratchBookApp::InitInstance()
{
    CWinApp::InitInstance();

    // Стандартная инициализация.
    // Если вы не используете какие-то из предоставленных
    // возможностей и хотите уменьшить размер конечного
    // исполняемого модуля, удалите из последующего кода
    // отдельные команды инициализации элементов, которые
    // вам не нужны.
    // Измените ключ, под которым ваши установки хранятся в реестре.
    // TODO: Измените эту строку на что-нибудь подходящее,
    // например, на имя вашей компании или организации
    SetRegistryKey(_T("Local AppWizard-Generated Applications"));
    LoadStdProfileSettings(4); // Загрузка стандартных установок
                               // из INI-файла (включая MRU)
    // Регистрация шаблона документа приложения. Шаблоны
    // документов служат связью между документами, окнами
    // документов и окнами приложений
    CSingleDocTemplate* pDocTemplate;
    pDocTemplate = new CSingleDocTemplate(
        IDR_MAINFRAME,
        RUNTIME_CLASS(CScratchBookDoc),
        RUNTIME_CLASS(CMainFrame) // главное окно
                                // SDI-приложения
        RUNTIME_CLASS(CScratchBookView));
    AddDocTemplate(pDocTemplate);

    EnableShellOpen ();
    RegisterShellFileTypes ();

    // Просмотр командной строки для обнаружения стандартных
    // команд оболочки, DDE, открытия файлов
    CCommandLineInfo cmdInfo;
    ParseCommandLine(cmdInfo);
    // Выполнение команд, указанных в командной строке.
    // Вернет FALSE, если приложение было запущено с
    // /RegServer, /Register, /Unregserver или /Unregister.
    if (!ProcessShellCommand(cmdInfo))
        return FALSE;
    // Показ и обновление единственного проинициализированного окна
    m_pMainWnd->ShowWindow(SW_SHOW);
    m_pMainWnd->UpdateWindow();
}

```

```

        m_pMainWnd->DragAcceptFiles ();

        return TRUE;
    }

    // CAboutDlg диалог, используемый в App About

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

        // Данные для диалога
        enum { IDD = IDD_ABOUTBOX };

    protected:
        virtual void DoDataExchange(CDataExchange* pDX);
        // Поддержка DDX/DDV

        // Реализация
    protected:
        DECLARE_MESSAGE_MAP()
    public:
    };

    CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
    {
        //
    }

    void CAboutDlg::DoDataExchange(CDataExchange* pDX)
    {
        CDialog::DoDataExchange(pDX);
    }

    BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)

    END_MESSAGE_MAP()

    // Команда приложения для выполнения диалога
    void CScratchBookApp::OnAppAbout()
    {
        CAboutDlg aboutDlg;
        aboutDlg.DoModal();
    }

    // Обработчики сообщений класса CScratchBookApp

    void CScratchBookApp::OnToolsLine()
    {
        // TODO: Добавьте сюда собственный код обработчика
        m_CurrentTool = ID_TOOLS_LINE;
    }

    void CScratchBookApp::OnToolsRectangle()
    {

```

```

        // TODO: Добавьте сюда собственный код обработчика
        m_CurrentTool = ID_TOOLS_RECTANGLE;
    }

void CScratchBookApp::OnUpdateToolsRectangle(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck
        (m_CurrentTool == ID_TOOLS_RECTANGLE ? 1 : 0);
}

void CScratchBookApp::OnToolsFrextangle()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentTool = ID_TOOLS_FREXTANGLE;
}

void CScratchBookApp::OnUpdateToolsFrextangle(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck
        (m_CurrentTool == ID_TOOLS_FREXTANGLE ? 1 : 0);
}

void CScratchBookApp::OnToolsRrectangle()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentTool = ID_TOOLS_RRECTANGLE;
}

void CScratchBookApp::OnUpdateToolsRrectangle(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck
        (m_CurrentTool == ID_TOOLS_RRECTANGLE ? 1 : 0);
}

void CScratchBookApp::OnToolsFrrectangle()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentTool = ID_TOOLS_FRRECTANGLE;
}

void CScratchBookApp::OnUpdateToolsFrrectangle(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck
        (m_CurrentTool == ID_TOOLS_FRRECTANGLE ? 1 : 0);
}

void CScratchBookApp::OnToolsCircle()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentTool = ID_TOOLS_CIRCLE;
}

```

```

void CScratchBookApp::OnUpdateToolsCircle(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck
        (m_CurrentTool == ID_TOOLS_CIRCLE ? 1 : 0);
}

void CScratchBookApp::OnToolsFcircle()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentTool = ID_TOOLS_FCIRCLE;
}

void CScratchBookApp::OnUpdateToolsFcircle(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck
        (m_CurrentTool==ID_TOOLS_FCIRCLE ? 1 : 0);
}

void CScratchBookApp::OnLineSingle()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentWidth = 1;
}

void CScratchBookApp::OnUpdateLineSingle(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck(m_CurrentWidth == 1 ? 1 : 0);
}

void CScratchBookApp::OnLineDouble()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentWidth = 2;
}

void CScratchBookApp::OnUpdateLineDouble(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck(m_CurrentWidth == 2 ? 1 : 0);
}

void CScratchBookApp::OnLineTriple()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentWidth = 3;
}

void CScratchBookApp::OnUpdateLineTriple(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck(m_CurrentWidth == 3 ? 1 : 0);
}

```

```

void CScratchBookApp::OnUpdateToolsLine(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck(m_CurrentTool == ID_TOOLS_LINE ? 1 : 0);
}

void CScratchBookApp::OnColorBlack()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentColor = RGB (0, 0, 0);
    m_IdxColorCmd = ID_COLOR_BLACK;
}

void CScratchBookApp::OnUpdateColorBlack(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck (m_IdxColorCmd == ID_COLOR_BLACK ? 1 : 0);
}

void CScratchBookApp::OnColorWhite()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentColor = RGB (255, 255, 255);
    m_IdxColorCmd = ID_COLOR_WHITE;
}

void CScratchBookApp::OnUpdateColorWhite(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck(m_IdxColorCmd == ID_COLOR_WHITE ? 1 : 0);
}

void CScratchBookApp::OnColorRed()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentColor = RGB (255, 0, 0);
    m_IdxColorCmd = ID_COLOR_RED;
}

void CScratchBookApp::OnUpdateColorRed(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI ->SetCheck(m_IdxColorCmd == ID_COLOR_RED ? 1 : 0);
}

void CScratchBookApp::OnColorGreen()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentColor = RGB (0, 255, 0);
    m_IdxColorCmd = ID_COLOR_GREEN;
}

void CScratchBookApp::OnUpdateColorGreen(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck(m_IdxColorCmd == ID_COLOR_GREEN ? 1 : 0);
}

```

```

void CScratchBookApp::OnColorBlue()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentColor = RGB (0, 0, 255);
    m_IdxColorCmd = ID_COLOR_BLUE;
}

void CScratchBookApp::OnUpdateColorBlue(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck(m_IdxColorCmd == ID_COLOR_BLUE ? 1 : 0);
}

void CScratchBookApp::OnColorYellow()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentColor = RGB (255, 255, 0);
    m_IdxColorCmd = ID_COLOR_YELLOW;
}

void CScratchBookApp::OnUpdateColorYellow(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck (m_IdxColorCmd == ID_COLOR_YELLOW ? 1 : 0);
}

void CScratchBookApp::OnColorCyan()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentColor = RGB (0, 255, 255);
    m_IdxColorCmd = ID_COLOR_CYAN;
}

void CScratchBookApp::OnUpdateColorCyan(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck(m_IdxColorCmd == ID_COLOR_CYAN ? 1 : 0);
}

void CScratchBookApp::OnColorMagenta()
{
    // TODO: Добавьте сюда собственный код обработчика
    m_CurrentColor = RGB (255, 0, 255);
    m_IdxColorCmd = ID_COLOR_MAGENTA;
}

void CScratchBookApp::OnUpdateColorMagenta(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->SetCheck(m_IdxColorCmd == ID_COLOR_MAGENTA ? 1 : 0);
}

void CScratchBookApp::OnColorCustom()
{
    // TODO: Добавьте сюда собственный код обработчика

```



```

        CColorDialog ColorDialog;

        if (ColorDialog.DoModal () == IDOK)
        {
            m_CurrentColor = ColorDialog.GetColor ();
            m_IdxColorCmd = ID_COLOR_CUSTOM;
        }
    }

void CScratchBookApp::OnUpdateColorCustom(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI ->SetCheck(m_IdxColorCmd == ID_COLOR_CUSTOM ? 1 : 0);
}

```

---

### Листинг 21.3.

```

// ScratchBookDoc.h : интерфейс класса CScratchBookDoc
//

```

```

#pragma once

```

```

class CFigure: public CObject
{
protected:
    COLORREF m_Color;
    DWORD m_X1, m_Y1, m_X2, m_Y2;
    CFigure () {}
    DECLARE_SERIAL (CFigure)

public:
    virtual void Draw (CDC *PDC) {}
    CRect GetDimRect ();
    virtual void Serialize (CArchive& ar);
};

class CLine: public CFigure
{
protected:
    DWORD m_Width;
    CLine () {}
    DECLARE_SERIAL (CLine)

public:
    CLine (int X1, int Y1, int X2, int Y2,
           COLORREF Color, int Width);
    virtual void Draw (CDC *PDC);
    virtual void Serialize (CArchive& ar);
};

class CRectangle: public CFigure
{
protected:
    DWORD m_Width;
    CRectangle () {}
    DECLARE_SERIAL (CRectangle)
}

```

```

public:
    CRectangle (int X1, int Y1, int X2, int Y2,
                COLORREF Color, int Width);
    virtual void Draw (CDC *PDC);
    virtual void Serialize (CArchive& ar);
};

class CRectangle: public CFigure
{
protected:
    DWORD m_Width;
    CRectangle () {}
    DECLARE_SERIAL (CRectangle)

public:
    CRectangle (int X1, int Y1, int X2, int Y2,
                COLORREF Color, int Width);
    virtual void Draw (CDC *PDC);
    virtual void Serialize (CArchive& ar);
};

class CRRectangle: public CFigure
{
protected:
    DWORD m_Width;
    CRRectangle () {}
    DECLARE_SERIAL (CRRectangle)

public:
    CRRectangle (int X1, int Y1, int X2, int Y2,
                COLORREF Color, int Width);
    virtual void Draw (CDC *PDC);
    virtual void Serialize (CArchive& ar);
};

class CFRRectangle: public CFigure
{
protected:
    DWORD m_Width;
    CFRRectangle () {}
    DECLARE_SERIAL (CFRRectangle)

public:
    CFRRectangle (int X1, int Y1, int X2, int Y2,
                COLORREF Color, int Width);
    virtual void Draw (CDC *PDC);
    virtual void Serialize (CArchive& ar);
};

class CCircle: public CFigure
{
protected:
    DWORD m_Width;
    CCircle () {}
    DECLARE_SERIAL (CCircle)

```

```

public:
    CCircle (int X1, int Y1, int X2, int Y2,
             COLORREF Color, int Width);
    virtual void Draw (CDC *PDC);
    virtual void Serialize (CArchive& ar);
};

class CFCircle: public CFigure
{
protected:
    DWORD m_Width;
    CFCircle () {}
    DECLARE_SERIAL (CFCircle)

public:
    CFCircle (int X1, int Y1, int X2, int Y2, COLORREF Color);
    virtual void Draw (CDC *PDC);
    virtual void Serialize (CArchive& ar);
};

class CScratchBookDoc : public CDocument
{
protected:
    CTypedPtrArray<CObArray, CFigure*> m_FigArray;

public:
    int GetNumFigs ();
    void AddFigure (CFigure *PFigure);
    CFigure *GetFigure (int Index);

protected: // используется только для сериализации
    CScratchBookDoc();
    DECLARE_DYNCREATE(CScratchBookDoc)

// Атрибуты
public:

// Операции
public:

// Переопределения
public:
    virtual BOOL OnNewDocument();
    virtual void Serialize(CArchive& ar);

// Реализация
public:
    virtual ~CScratchBookDoc();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif
protected:

```

```

// Сгенерированные функции карты сообщений
protected:
    DECLARE_MESSAGE_MAP()
public:
    virtual void DeleteContents();
    afx_msg void On57633();
    afx_msg void OnUpdate57633(CCmdUI *pCmdUI);
    afx_msg void On57643();
    afx_msg void OnUpdate57643(CCmdUI *pCmdUI);
};

```

---

#### Листинг 21.4.

```

// ScratchBookDoc.cpp : реализация класса CScratchBookDoc
//

#include "stdafx.h"
#include "ScratchBook.h"

#include "ScratchBookDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// CScratchBookDoc

IMPLEMENT_DYNCREATE(CScratchBookDoc, CDocument)

BEGIN_MESSAGE_MAP(CScratchBookDoc, CDocument)
    ON_COMMAND(57633, On57633)
    ON_UPDATE_COMMAND_UI(57633, OnUpdate57633)
    ON_COMMAND(57643, On57643)
    ON_UPDATE_COMMAND_UI(57643, OnUpdate57643)
END_MESSAGE_MAP()

// Конструктор/деструктор класса CScratchBookDoc

CScratchBookDoc::CScratchBookDoc()
{
    // TODO: добавьте сюда код одноразового конструктора
}

CScratchBookDoc::~CScratchBookDoc()
{
}

BOOL CScratchBookDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;

    // TODO: добавьте сюда код повторной инициализации
    // (SDI-документы будут использовать этот документ многократно)
}

```

```

        return TRUE;
    }

    // Сериализация CScratchBookDoc

void CScratchBookDoc::Serialize(CArchive& ar)
{
    if (ar.IsStoring())
    {
        // TODO: добавьте сюда код сохранения
        m_FigArray.Serialize(ar);
    }
    else
    {
        // TODO: добавьте сюда код загрузки
        m_FigArray.Serialize(ar);
    }
}

// Диагностика класса CScratchBookDoc

#ifdef _DEBUG
void CScratchBookDoc::AssertValid() const
{
    CDocument::AssertValid();
}

void CScratchBookDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}
#endif //_DEBUG

// Команды класса CScratchBookDoc

void *CScratchBookDoc::AddFigure (CFigure *PFigure)
{
    m_FigArray.Add (PFigure);
    SetModifiedFlag( );
}

void CScratchBookDoc::DeleteContents()
{
    // TODO: Добавьте сюда собственный код или
    // вызов базового класса
    int Index = m_FigArray.GetSize ();
    while (Index--)
        delete m_FigArray.GetAt (Index);
    m_FigArray.RemoveAll ();

    CDocument::DeleteContents();
}

void CScratchBookDoc::On57633()
{

```

```

        // TODO: Добавьте сюда собственный код обработчика
        DeleteContents ();
        UpdateAllViews (0);
        SetModifiedFlag ( );
    }

void CScratchBookDoc::OnUpdate57633(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика

    pCmdUI->Enable (m_FigArray.GetSize ());
}

void CScratchBookDoc::On57643()
{
    // TODO: Добавьте сюда собственный код обработчика
    int Index = m_FigArray.GetUpperBound ();
    if (Index > -1)
    {
        delete m_FigArray.GetAt (Index);
        m_FigArray.RemoveAt (Index);
    }
    UpdateAllViews (0);
    SetModifiedFlag ();
}

void CScratchBookDoc::OnUpdate57643(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика
    pCmdUI->Enable (m_FigArray.GetSize ());
}

// реализация классов фигур

IMPLEMENT_SERIAL (CFigure, CObject, 3)

CRect CFigure::GetDimRect ()
{
    return CRect
        (min (m_X1, m_X2), min (m_Y1, m_Y2),
         (max (m_X1, m_X2))+1, (max (m_Y1, m_Y2))+1);
}

void CFigure::Serialize (CArchive& ar)
{
    if (ar.IsStoring ())
        ar << m_X1 << m_Y1 << m_X2 << m_Y2 << m_Color;
    else
        ar >> m_X1 >> m_Y1 >> m_X2 >> m_Y2 >> m_Color;
}

IMPLEMENT_SERIAL (CLine, CFigure, 3);

CLine::CLine (int X1, int Y1, int X2, int Y2,
              COLORREF Color, int Width)
{

```

```

        m_X1 = X1;
        m_Y1 = Y1;
        m_X2 = X2;
        m_Y2 = Y2;
        m_Color = Color;
        m_Width = Width;
    }

void CLine::Serialize (CArchive &ar)
{
    CFigure::Serialize (ar);
    if (ar.IsStoring ())
        ar << m_Width;
    else
        ar >> m_Width;
}

void CLine::Draw (CDC *PDC)
{
    CPen Pen, *POldPen;

    // выбор пера/кисти
    Pen.CreatePen (PS_SOLID, m_Width, m_Color);
    POldPen = PDC->SelectObject (&Pen);

    // рисование фигуры
    PDC->MoveTo (m_X1, m_Y1);
    PDC->LineTo (m_X2, m_Y2);

    // восстановление пера/кисти
    PDC->SelectObject (POldPen);
}

IMPLEMENT_SERIAL (CRectangle, CFigure, 3);

CRectangle::CRectangle (int X1, int Y1, int X2, int Y2,
                        COLORREF Color, int Width)
{
    m_X1 = X1;
    m_Y1 = Y1;
    m_X2 = X2;
    m_Y2 = Y2;
    m_Color = Color;
    m_Width = Width;
}

void CRectangle::Serialize (CArchive &ar)
{
    CFigure::Serialize (ar);
    if (ar.IsStoring ())
        ar << m_Width;
    else
        ar >> m_Width;
}

void CRectangle::Draw (CDC *PDC)

```

```

{
    CPen Pen, *PoldPen;

    // выбор пера/кисти
    Pen.CreatePen (PS_INSIDEFRAME, m_Width, m_Color);
    PoldPen = PDC->SelectObject (&Pen);
    PDC->SelectStockObject (NULL_BRUSH);

    // рисование фигуры
    PDC->Rectangle (m_X1, m_Y1, m_X2, m_Y2);

    // восстановление пера/кисти
    PDC->SelectObject (PoldPen);
}

IMPLEMENT_SERIAL (CRectangle, CFigure, 3);

CRectangle::CRectangle (int X1, int Y1, int X2, int Y2,
                        COLORREF Color, int Width)
{
    m_X1 = X1;
    m_Y1 = Y1;
    m_X2 = X2;
    m_Y2 = Y2;
    m_Color = Color;
    m_Width = Width;
}

void CRectangle::Serialize (CArchive &ar)
{
    CFigure::Serialize (ar);
    if (ar.IsStoring ())
        ar << m_Width;
    else
        ar >> m_Width;
}

void CRectangle::Draw (CDC *PDC)
{
    CPen Pen, *PoldPen;

    // выбор пера/кисти
    Pen.CreatePen (PS_INSIDEFRAME, m_Width, m_Color);
    PoldPen = PDC->SelectObject (&Pen);
    PDC->SelectStockObject (NULL_BRUSH);

    // рисование фигуры
    PDC->Rectangle (m_X1, m_Y1, m_X2, m_Y2);

    // восстановление пера/кисти
    PDC->SelectObject (PoldPen);
}

IMPLEMENT_SERIAL (CFRRectangle, CFigure, 3);

CFRRectangle::CFRRectangle (int X1, int Y1, int X2, int Y2,
                            COLORREF Color, int Width)
{

```



```

        m_X1 = X1;
        m_Y1 = Y1;
        m_X2 = X2;
        m_Y2 = Y2;
        m_Color = Color;
        m_Width = Width;
    }

void CFRRectangle::Draw (CDC *PDC)
{
    CBrush Brush, *PoldBrush;
    CPen Pen, *PoldPen;

    // выбор пера/кисти
    Pen.CreatePen (PS_INSIDEFRAME, 1, m_Color);
    PoldPen = PDC->SelectObject (&Pen);
    Brush.CreateSolidBrush (m_Color);
    PoldBrush=PDC->SelectObject (&Brush);

    // рисование фигуры
    int SizeRound = (m_X2 - m_X1, m_Y2 - m_Y1) / 8;
    PDC->RoundRect (m_X1, m_Y1, m_X2, m_Y2, SizeRound, SizeRound);

    // восстановление пера/кисти
    PDC->SelectObject (PoldPen);
    PDC->SelectObject (PoldBrush);
}

IMPLEMENT_SERIAL (CRRectangle, CFigure, 3);

CRRectangle::CRRectangle (int X1, int Y1, int X2, int Y2,
                           COLORREF Color, int Width)
{
    m_X1 = X1;
    m_Y1 = Y1;
    m_X2 = X2;
    m_Y2 = Y2;
    m_Color = Color;
    m_Width = Width;
}

void CRRectangle::Serialize (CArchive &ar)
{
    CFigure::Serialize (ar);
    if (ar.IsStoring ())
        ar << m_Width;
    else
        ar << m_Width;
}

void CRRectangle::Draw (CDC *PDC)
{
    CPen Pen, *PoldPen;

    // выбор пера/кисти
    Pen.CreatePen (PS_INSIDEFRAME, m_Width, m_Color);

```

```

    POldPen = PDC->SelectObject (&Pen);
    PDC->SelectStockObject (NULL_BRUSH);

    // рисование фигуры
    int SizeRound = (m_X2 - m_X1 + m_Y2 - m_Y1) / 8;
    PDC->RoundRect (m_X1, m_Y1, m_X2, m_Y2, SizeRound, SizeRound);

    // восстановление пера/кисти
    PDC->SelectObject (POldPen);
}

IMPLEMENT_SERIAL (CFCircle, CFigure, 3);

CFCircle::CFCircle (int X1, int Y1, int X2, int Y2, COLORREF Color)
{
    m_X1 = X1;
    m_Y1 = Y1;
    m_X2 = X2;
    m_Y2 = Y2;
    m_Color = Color;
}

void CFCircle::Draw (CDC *PDC)
{
    CBrush Brush, *POldBrush;
    CPen Pen, *POldPen;

    // выбор пера/кисти
    Pen.CreatePen (PS_INSIDEFRAME, m_Width, m_Color);
    POldPen = PDC->SelectObject (&Pen);
    Brush.CreateSolidBrush (m_Color);
    POldBrush = PDC->SelectObject (&Brush);

    // рисование фигуры
    PDC->Ellipse (m_X1, m_Y1, m_X2, m_Y2);

    // восстановление пера/кисти
    PDC->SelectObject (POldPen);
    PDC->SelectObject (POldBrush);
}

IMPLEMENT_SERIAL (CCircle, CFigure, 3);

CCircle::CCircle (int X1, int Y1, int X2, int Y2,
                  COLORREF Color, int Width)
{
    m_X1 = X1;
    m_Y1 = Y1;
    m_X2 = X2;
    m_Y2 = Y2;
    m_Color = Color;
    m_Width = Width;
}

void CCircle::Serialize (CArchive &ar)
{

```

```

        CFigure::Serialize (ar);
        if (ar.IsStoring ())
            ar << m_Width;
        else
            ar >> m_Width;
    }

void CCircle::Draw (CDC *PDC)
{
    CPen Pen, *PoldPen;

    // выбор пера/кисти
    Pen.CreatePen (PS_INSIDEFRAME, m_Width, m_Color);
    PoldPen = PDC->SelectObject (&Pen);
    PDC->SelectStockObject (NULL_BRUSH);

    // рисование фигуры
    PDC->Ellipse (m_X1, m_Y1, m_X2, m_Y2);

    // восстановление пера/кисти
    PDC->SelectObject (PoldPen);
}

CFigure *CScratchBookDoc::GetFigure (int Index)
{
    if (Index < 0 || Index > m_FigArray.GetUpperBound ())
        return 0;
    return (CFigure *)m_FigArray.GetAt (Index);
}

int CScratchBookDoc::GetNumFigs ()
{
    return m_FigArray.GetSize ();
}

```

---

### Листинг 21.5.

```

// ScratchBookView.h : интерфейс класса CScratchBookView
//

const int DRAWWIDTH = 4000; //ширина рисунка
const int DRAWHEIGHT = 6000; //высота рисунка

#pragma once

class CScratchBookView : public CScrollView
{
protected:
    CString m_ClassName;
    int m_Dragging;
    HCURSOR m_HCross;
    HCURSOR m_HArrow;
    CPoint m_PointOld;
    CPoint m_PointOrigin;
    CPen m_PenDotted;
    int m_NumCols, m_NumRows;

```

```

        int m_PageHeight, m_PageWidth;

protected: // используется только для сериализации
    CScratchBookView();
    DECLARE_DYNCREATE(CScratchBookView)

// Атрибуты
public:
    CScratchBookDoc* GetDocument() const;

// Операции
public:

// Переопределения
    public:
        virtual void OnDraw(CDC* pDC);
        // переопределена для прорисовки этого вида
virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
protected:

// Реализация
public:
    virtual ~CScratchBookView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Сгенерированные функции карты сообщений
protected:
    DECLARE_MESSAGE_MAP()
public:
    afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
    afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
    afx_msg void OnMouseMove(UINT nFlags, CPoint point);
    virtual void OnInitialUpdate();
protected:
    virtual void OnUpdate(CView* /*pSender*/, LPARAM /*lHint*/,
                        CObject* /*pHint*/);
    virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
    virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
public:
    virtual void OnPrepareDC(CDC* pDC, CPrintInfo* pInfo = NULL);
};

#ifdef _DEBUG // отладочная версия в файле ScratchBookView.cpp
inline CScratchBookDoc* CScratchBookView::GetDocument() const
    { return (CScratchBookDoc*)m_pDocument; }
#endif

```

---

## Листинг 21.6.

```

// ScratchBookView.cpp : реализация класса CScratchBookView
//

```

```

#include "stdafx.h"
#include "ScratchBook.h"

#include "ScratchBookDoc.h"
#include "ScratchBookView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// CScratchBookView

IMPLEMENT_DYNCREATE(CScratchBookView, CScrollView)

BEGIN_MESSAGE_MAP(CScratchBookView, CScrollView)
    ON_WM_LBUTTONDOWN()
    ON_WM_LBUTTONUP()
    ON_WM_MOUSEMOVE()
    ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview);
END_MESSAGE_MAP()

// Конструктор/деструктор класса CScratchBookView

CScratchBookView::CScratchBookView()
{
    // TODO: добавьте сюда собственный код конструктора

    m_Dragging = 0;
    m_HCross = AfxGetApp()->LoadStandardCursor (IDC_CROSS);
    m_HArrow = AfxGetApp()->LoadStandardCursor (IDC_ARROW);
    m_PenDotted.CreatePen (PS_DOT, 1, RGB (0, 0, 0));
}

CScratchBookView::~CScratchBookView()
{
}

BOOL CScratchBookView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Модифицируйте класс или стили окна,
    // добавляя и изменяя поля структуры cs

    m_ClassName = AfxRegisterWndClass
        (CS_HREDRAW | CS_VREDRAW,    // стили окна
        0,                          // без указателя
        (HBRUSH)::GetStockObject (WHITE_BRUSH),
        0);                          // задать белый фон
    cs.lpszClass = m_ClassName;

    return CScrollView::PreCreateWindow(cs);
}

```

```

// Прорисовка CScratchBookView

void CScratchBookView::OnDraw(CDC* pDC)
{
    CScratchBookDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    // TODO: вставьте сюда код прорисовки собственных данных

    if (pDC->GetDeviceCaps (TECHNOLOGY) == DT_RASDISPLAY)
    {
        CSize ScrollSize = GetTotalSize ();
        pDC->MoveTo (ScrollSize.cx, 0);
        pDC->LineTo (ScrollSize.cx, ScrollSize.cy);
        pDC->LineTo (0, ScrollSize.cy);
    }

    CRect ClipRect;
    CRect DimRect;
    CRect IntRect;
    CFigure *PFigure;
    pDC->GetClipBox (&ClipRect);

    int NumFigs = pDoc->GetNumFigs ();
    for (int Index = 0; Index < NumFigs; ++Index)
    {
        PFigure = pDoc->GetFigure (Index);
        DimRect = PFigure->GetDimRect ();
        if (IntRect.IntersectRect (DimRect, ClipRect))
            PFigure->Draw(pDC);
    }
}

// Диагностика класса CScratchBookView

#ifdef _DEBUG
void CScratchBookView::AssertValid() const
{
    CScrollView::AssertValid();
}

void CScratchBookView::Dump(CDumpContext& dc) const
{
    CScrollView::Dump(dc);
}

CScratchBookDoc* CScratchBookView::GetDocument() const
// не отладочная версия встроена
{
    ASSERT(m_pDocument->IsKindOf (RUNTIME_CLASS(CScratchBookDoc)));
    return (CScratchBookDoc*)m_pDocument;
}
#endif // _DEBUG

// Обработчики сообщений класса CScratchBookView

```

```

void CScratchBookView::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: Вставьте сюда собственный код обработчика
    // или вызов стандартного
    CClientDC clientDC (this);
    OnPrepareDC (&clientDC);
    clientDC.DPtoLP (&point);

    // проверка нахождения курсора внутри области
    // окна представления
    CSize ScrollSize = GetTotalSize ();
    CRect ScrollRect (0, 0, ScrollSize.cx, ScrollSize.cy);
    if (!ScrollRect.PtInRect (point))
        return;

    // сохранение позиции курсора, захват мыши и
    // установка флага перемещения
    m_PointOrigin = point;
    m_PointOld = point;
    SetCapture ();
    m_Dragging = 1;

    // ограничение перемещений курсора мыши
    clientDC.LPtoDP (&ScrollRect);
    CRect ViewRect;
    GetClientRect (&ViewRect);
    CRect IntRect;
    IntRect.IntersectRect (&ScrollRect, &ViewRect);
    ClientToScreen (&IntRect);
    ::ClipCursor (&IntRect);

    CScrollView::OnLButtonDown(nFlags, point);
}

void CScratchBookView::OnLButtonUp(UINT nFlags, CPoint point)
{
    int SizeRound;
    // TODO: Вставьте сюда собственный код обработчика
    // или вызов стандартного
    if (!m_Dragging) return;

    m_Dragging = 0;
    ::ReleaseCapture ();
    ::ClipCursor (NULL);

    CClientDC clientDC(this);
    OnPrepareDC (&clientDC);
    clientDC.DPtoLP (&point);
    clientDC.SetROP2 (R2_NOT);
    clientDC.SelectObject (&m_PenDotted);
    clientDC.SetBkMode (TRANSPARENT);
    clientDC.SelectStockObject (NULL_BRUSH);

    CScratchBookApp *PApp = (CScratchBookApp *)AfxGetApp ();
    CFigure *PFigure;

```

```

switch (PApp->m_CurrentTool)
{
case ID_TOOLS_LINE:
    ClientDC.MoveTo (m_PointOrigin);
    ClientDC.LineTo (m_PointOld);
    PFigure = new CLine
        (m_PointOrigin.x, m_PointOrigin.y, point.x, point.y,
         PApp->m_CurrentColor, PApp->m_CurrentWidth);
    break;

case ID_TOOLS_RECTANGLE:
    ClientDC.Rectangle (m_PointOrigin.x, m_PointOrigin.y,
        m_PointOld.x, m_PointOld.y);
    PFigure = new CRectangle
        (m_PointOrigin.x, m_PointOrigin.y,
        point.x, point.y,
        PApp->m_CurrentColor, PApp->m_CurrentWidth);
    break;

case ID_TOOLS_FRECTANGLE:
    ClientDC.Rectangle (m_PointOrigin.x, m_PointOrigin.y,
        m_PointOld.x, m_PointOld.y);
    PFigure = new CFRectangle
        (m_PointOrigin.x, m_PointOrigin.y,
        point.x, point.y,
        PApp->m_CurrentColor, PApp->m_CurrentWidth);
    break;

case ID_TOOLS_RRECTANGLE:
    SizeRound = (abs(m_PointOld.x - m_PointOrigin.x) +
        abs (m_PointOld.y - m_PointOrigin.y)) / 8;
    ClientDC.RoundRect (m_PointOrigin.x, m_PointOrigin.y,
        m_PointOld.x, m_PointOld.y, SizeRound, SizeRound);
    PFigure = new CRRectangle
        (m_PointOrigin.x, m_PointOrigin.y,
        point.x, point.y,
        PApp->m_CurrentColor, PApp->m_CurrentWidth);
    break;

case ID_TOOLS_FRRECTANGLE:
    SizeRound = (abs(m_PointOld.x - m_PointOrigin.x) +
        abs (m_PointOld.y - m_PointOrigin.y)) / 8;
    ClientDC.RoundRect (m_PointOrigin.x, m_PointOrigin.y,
        m_PointOld.x, m_PointOld.y, SizeRound, SizeRound);
    PFigure = new CFRRectangle
        (m_PointOrigin.x, m_PointOrigin.y,
        point.x, point.y,
        PApp->m_CurrentColor, PApp->m_CurrentWidth);
    break;

case ID_TOOLS_CIRCLE:
    ClientDC.Ellipse (m_PointOrigin.x, m_PointOrigin.y,
        m_PointOld.x, m_PointOld.y);
    PFigure = new CCircle
        (m_PointOrigin.x, m_PointOrigin.y, point.x, point.y,
        PApp->m_CurrentColor, PApp->m_CurrentWidth);
    break;
}

```



```

        case ID_TOOLS_FCIRCLE:
            ClientDC.Ellipse (m_PointOrigin.x, m_PointOrigin.y,
                             m_PointOld.x, m_PointOld.y);
            PFigure = new CFCircle
            (m_PointOrigin.x, m_PointOrigin.y, point.x, point.y,
             PApp->m_CurrentColor);
            break;

            ClientDC.SetROP2 (R2_COPYPEN);
            PFigure->Draw (&ClientDC);
            CScratchBookDoc *PDoc = GetDocument ();
            PDoc->AddFigure (PFigure);

            PDoc->UpdateAllViews (this, 0, PFigure);

    CScrollView::OnLButtonUp(nFlags, point);
}
}

void CScratchBookView::OnMouseMove(UINT nFlags, CPoint point)
{
    // TODO: Вставьте сюда собственный код обработчика
    // или вызов стандартного

    int SizeRound;
    CClientDC ClientDC (this);
    OnPrepareDC (&ClientDC);
    ClientDC.DPtoLP (&point);

    if (!m_Dragging)
    {
        CSize ScrollSize = GetTotalSize ();
        CRect ScrollRect (0, 0, ScrollSize.cx, ScrollSize.cy);
        if (ScrollRect.PtInRect (point))
            ::SetCursor (m_HCross);
        else
            ::SetCursor (m_HArrow);
        return;
    }

    ClientDC.SetROP2 (R2_NOT);
    ClientDC.SelectObject (&m_PenDotted);
    ClientDC.SetBkMode (TRANSPARENT);
    ClientDC.SelectStockObject (NULL_BRUSH);

    switch (((CScratchBookApp *)AfxGetApp ())->m_CurrentTool)
    {
        case ID_TOOLS_LINE:
            ClientDC.MoveTo (m_PointOrigin);
            ClientDC.LineTo (m_PointOld);
            ClientDC.MoveTo (m_PointOrigin);
            ClientDC.LineTo (point);
            break;

        case ID_TOOLS_RECTANGLE:

```

```

case ID_TOOLS_FRECTANGLE:
    ClientDC.Rectangle (m_PointOrigin.x, m_PointOrigin.y,
        m_PointOld.x, m_PointOld.y);
    ClientDC.Rectangle (m_PointOrigin.x, m_PointOrigin.y,
        point.x, point.y);
    break;

case ID_TOOLS_RRECTANGLE:
case ID_TOOLS_FRRECTANGLE:
    {
        SizeRound = (abs(m_PointOld.x - m_PointOrigin.x) +
            abs (m_PointOld.y - m_PointOrigin.y)) / 8;
        ClientDC.RoundRect
            (m_PointOrigin.x, m_PointOrigin.y,
            m_PointOld.x, m_PointOld.y,
            SizeRound, SizeRound);
        SizeRound = (abs(point.x - m_PointOrigin.x) +
            abs (point.y - m_PointOrigin.y)) / 8;
        ClientDC.RoundRect
            (m_PointOrigin.x, m_PointOrigin.y,
            point.x, point.y, SizeRound, SizeRound);

        case ID_TOOLS_CIRCLE:
        case ID_TOOLS_FCIRCLE:
            ClientDC.Ellipse (m_PointOrigin.x, m_PointOrigin.y,
                m_PointOld.x, m_PointOld.y);
            ClientDC.Ellipse (m_PointOrigin.x, m_PointOrigin.y,
                point.x, point.y);

        break;
    }
    m_PointOld = point;

    CScrollView::OnMouseMove(nFlags, point);
}

void CScratchBookView::OnInitialUpdate()
{
    CScrollView::OnInitialUpdate();

    // TODO: Добавьте сюда собственный код
    // или вызов базового класса

    SIZE Size = {DRAWWIDTH, DRAWHEIGHT};
    SetScrollSizes (MM_TEXT, Size);
}

void CScratchBookView::OnUpdate(CView* pSender, LPARAM lHint,
                                CObject* pHint)
{
    // TODO: Добавьте сюда собственный код или
    // вызов базового класса
    if (pHint != 0)
    {
        CRect InvalidRect = ((CLine *)pHint)->GetDimRect ();
        CClientDC ClientDC (this);
    }
}

```

```

        OnPrepareDC (&ClientDC);
        ClientDC.LPtoDP (&InvalidRect);
        InvalidateRect (&InvalidRect);
    }
    else
        CScrollView::OnUpdate (pSender, lHint, pHint);
}

BOOL CScratchBookView::OnPreparePrinting(CPrintInfo* pInfo)
{
    // TODO: вызовите DoPreparePrinting для вызова окна Print

    return DoPreparePrinting (pInfo);
}

void CScratchBookView::OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo)
{
    // TODO: Добавьте сюда собственный код обработчика
    // или вызов базового класса

    m_PageHeight = pDC->GetDeviceCaps (VERTRES);
    m_PageWidth = pDC->GetDeviceCaps (HORZRES);

    m_NumRows = DRAWHEIGHT / m_PageHeight + (DRAWHEIGHT %
        m_PageHeight > 0);
    m_NumCols = DRAWWIDTH / m_PageWidth + (DRAWWIDTH %
        m_PageWidth > 0);
    pInfo->SetMinPage (1);
    pInfo->SetMaxPage (m_NumRows * m_NumCols);

    CScrollView::OnBeginPrinting(pDC, pInfo);
}

void CScratchBookView::OnPrepareDC(CDC* pDC, CPrintInfo* pInfo)
{
    // TODO: Добавьте сюда собственный код
    // обработчика или вызов базового класса

    CScrollView::OnPrepareDC(pDC, pInfo);

    if (pInfo == NULL) return;

    int CurRow = pInfo->m_nCurPage / m_NumCols +
        (pInfo->m_nCurPage % m_NumCols > 0);
    int CurCol = (pInfo->m_nCurPage - 1) % m_NumCols + 1;

    pDC->SetViewportOrg (-m_PageWidth * (CurCol - 1),
        -m_PageHeight * (CurRow - 1));
}

```

---

#### Листинг 21.7.

```

// MainFrm.h : интерфейс класса CMainFrame
//

```

```

#pragma once
class CMainFrame : public CFrameWnd
{
protected:
    CSplitterWnd m_SplitterWnd;

protected: // используется только для сериализации
    CMainFrame();
    DECLARE_DYNCREATE(CMainFrame)

// Атрибуты
public:

// Операции
public:

// Переопределения
public:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);

// Реализация
public:
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected: // встроенные элементы управления
    CStatusBar m_wndStatusBar;
    CToolBar m_wndToolBar;

// Сгенерированные функции карты сообщений
protected:
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    DECLARE_MESSAGE_MAP()
    virtual BOOL OnCreateClient(LPCREATESTRUCT lpcs,
                               CCreateContext* pContext);
};

```

---

#### Листинг 21.8.

```

// MainFrm.cpp : реализация класса CMainFrame
//

#include "stdafx.h"
#include "ScratchBook.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

```

```

// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
    ON_WM_CREATE()
END_MESSAGE_MAP()

static UINT indicators[] =
{
    ID_SEPARATOR,           // индикатор строки состояния
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
};

// Конструктор/деструктор класса CMainFrame

CMainFrame::CMainFrame()
{
    // TODO: добавьте сюда код инициализации элементов
}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT,
        WS_CHILD | WS_VISIBLE | CBRS_TOP
        | CBRS_GRIPPER | CBRS_TOOLTIPS |
        CBRS_FLYBY | CBRS_SIZE_DYNAMIC) ||
        !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
    {
        TRACE0("Failed to create toolbar\n");
        return -1;      // не удалось создать панель инструментов
    }

    if (!m_wndStatusBar.Create(this) ||
        !m_wndStatusBar.SetIndicators(indicators,
        sizeof(indicators)/sizeof(UINT)))
    {
        TRACE0("Failed to create status bar\n");
        return -1;      // не удалось создать строку состояния
    }
    // TODO: Удалите три следующие строки, если не хотите,
    // чтобы панель инструментов была паркующей
    m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
    EnableDocking(CBRS_ALIGN_ANY);
    DockControlBar(&m_wndToolBar);

    return 0;
}

```

```

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    if( !CFrameWnd::PreCreateWindow(cs) )
        return FALSE;
    // TODO: Модифицируйте стили или классы окна здесь,
    // добавляя или изменяя поля структуры cs

    return TRUE;
}

// Диагностика класса CMainFrame

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
    CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
    CFrameWnd::Dump(dc);
}

#endif // _DEBUG

// Обработчики сообщений класса CMainFrame

BOOL CMainFrame::OnCreateClient(LPCREATESTRUCT lpcs,
                               CCreateContext* pContext)
{
    // TODO: Добавьте сюда собственный код
    // или вызов базового класса
    return m_SplitterWnd.Create
    (this,          // родительское окно разделенного окна;
     1,             // максимальное число строк;
     2,             // максимальное число столбцов;
     CSize(20, 20), // минимальный размер окна представления;
     pContext);     // информация о контексте устройства
}

```

## Резюме

Рассмотрены способы создания кода для стандартных команд Print..., Print Preview и Print Setup... в меню File.

- *Одностраничная печать.* При создании мастером Application Wizard новой программы в меню File программы можно включить команды Print..., Print Preview и Print Setup..., выбрав в мастере Application Wizard опцию "Printing and Print Preview". Однако начальный код, генерируемый мастером Application Wizard, будет печатать или просматривать только ту часть документа, которая помещается на одной странице. Для обработки команды Print Setup... в класс приложения, вызывающий функцию CWinApp::OnFilePrintSetup, необходимо добавить элемент схемы обработки сообщений. Чтобы обработать команды Print... и Print Preview, в класс представления добавляется элемент схемы обработки сообщений, вызывающий функции CView::OnFilePrint и CView::OnFilePrintPreview. Когда функции OnFilePrint и OnFilePrintPreview

управляют печатью или предварительным просмотром печати, они вызывают ряд функций класса `CView`. Необходимо переопределить виртуальную функцию `OnPreparePrinting`, которая вызывает функцию `CView::OnPreparePrinting`, чтобы для печати или предварительного просмотра создать объект контекста устройства.

В существующую программу можно добавить команды `Print...`, `Print Preview` и `Print Setup...` и изменить код для их выполнения. В редакторе меню команды печати добавляются в меню программы `File`. Кроме того, необходимо выбрать команду `Resource Includes` меню `View` и задать переопределенный файл `Afxprint.rc`, чтобы включить в программу ресурсы, определяемые в этом файле (для печати).

- *Многостраничная печать.* Если нужно напечатать более одной страницы, вызовите функции `SetMinPage` и `SetMaxPage`, чтобы указать номера первой и последней страниц. Обычно эти функции вызываются в переопределенной функции `OnBeginPrinting`, являющейся важной виртуальной функцией, имеющей доступ к объекту контекста устройства. Для поддержки многостраничной печати переопределите виртуальную функцию `OnPrepareDC`, которая вызывается перед печатью каждой страницы. Ваша функция должна настроить начало представления, чтобы функция программы `OnDraw` смогла напечатать содержимое текущей страницы документа. После вызова функции `OnPrepareDC` MFC вызывает виртуальную функцию `OnPrint`; стандартная версия этой функции вызывает функцию `OnDraw`, чтобы отобразить выводимую информацию на текущей странице. Независимый от используемой аппаратуры код функции `OnDraw` предназначен для отображения выводимой информации на различных устройствах. Для подгонки выводимой информации к конкретному устройству в функцию `GetDeviceCaps` передается константа `TECHNOLOGY`.

# Глава 22

## Потоки

---

- Вторичные потоки
- Способы синхронизации потоков
- Многопоточковая программа рисования ImpFractal

Термином *поток (thread)* обозначается выполнение последовательности команд программы. Все программы, ранее рассматриваемые в книге, выполняются в виде одного потока, называемого *первичным*. Однако в программе, написанной для Windows 95 или более поздних версий, а также для Windows NT и более поздних версий, можно запустить один или несколько *вторичных потоков* (или подзадач). Каждый поток независимо выполняет последовательность команд, соответствующих тексту программы. С точки зрения пользователя или программиста, создающего приложение, потоки в программе выполняются одновременно. Операционная система обычно достигает этой кажущейся одновременности за счет быстрого переключения управления с одного потока на другой. В многопроцессорной системе потоки могут выполняться одновременно в буквальном смысле.

Если программа в один момент времени должна выполнить несколько задач (как это часто бывает), то возможность предоставления каждой задаче отдельного потока не только делает программу более эффективной, но и упрощает ее разработку. В этой главе мы покажем, как создавать вторичные потоки и управлять ими. Вы узнаете, как использовать механизмы, предоставляемые Win32 API для синхронизации работы отдельных потоков. Наконец, будет создана многопоточковая версия программы FractalView (см. гл. 19) – программа ImpFractal.

### Вторичные потоки

---

Использование техники многопоточковой обработки особенно полезно в программах с графическим интерфейсом, когда первичный поток выделяется для обработки сообщений, что позволяет программе быстро реагировать на поступающие команды и другие события. Вторичный поток обычно используется для выполнения любой длительной задачи, которая блокировала бы обработку сообщений программы при ее выполнении первичным потоком, например, при рисовании сложных графических изображений, пересчете электронных таблиц, выполнении дисковых операций или связи с последовательным портом. Запуск отдельного *потока* происходит относительно быстро и занимает небольшой объем памяти по сравнению с запуском отдельного *процесса*, описанным в следующей главе. Кроме того, все потоки программы выполняются в одной области памяти и используют *общий набор ресурсов* Windows, следовательно, могут совместно использовать переменные и объекты Windows (окна, перья, распределения памяти, контексты устройств и т.д.). Далее вы увидите, что существует несколько ограничений на совместное использование объектов MFC различными потоками и процессами. Многие приемы многопоточковой работы, рассмотренные в этой главе, проиллюстрированы в приведенной в конце главы программе FractalView.

Новый поток запускается вызовом глобальной MFC-функции `AfxBeginThread`. Функция `AfxBeginThread` инициализирует библиотеку MFC для использования в многопоточковой программе. Затем она вызывает функцию `_beginthreadex` библиотеки периода выполнения, инициализирующую библиотеку программы с несколькими потоками, а затем для запуска потока вызывает функцию Win32 API `::CreateThread`. Если функцию `AfxBeginThread` использовать для запуска вторичного потока программы MFC, то можно без проблем использовать классы MFC, функции



библиотеки периода выполнения и Win32 API. В программах MFC *нельзя* запускать новый поток, непосредственно вызывая библиотечные функции `_beginthread` и `_beginthreadex` или функцию `::CreateThread` интерфейса Win32 API. Синтаксис функции `AfxBeginThread` выглядит так:

```
CWinThread* AfxBeginThread
(
    AFX_THREADPROC pfnThreadProc,
    LPVOID pParam,
    int nPriority = THREAD_PRIORITY_NORMAL,
    UINT nStackSize = 0,
    DWORD dwCreateFlags = 0,
    LPSECURITY_ATTRIBUTES lpSecurityAttrs = NULL);
```

Запустив новый поток, функция `AfxBeginThread` сразу возвращает управление. С этого момента оба потока (созданный и вызвавший функцию `AfxBeginThread`) работают одновременно.

- Первый параметр `pfnThreadProc` задает *функцию потока*, и новый поток начинает ее выполнять. Функция потока может вызывать другие функции, а при выходе из нее новый поток завершается. Параметр `pParam` задает значение, передаваемое в эту функцию. Функция потока определяется так

```
UINT ThreadFunction (LPVOID pParam);
```

и возвращает значение типа `UINT`, называемое *кодом возврата*.

Ниже показано, как код возврата читается другими потоками. При нормальном завершении функция потока возвращает значение 0. При желании можно использовать любое возвращаемое значение (оно читается и интерпретируется только вашей собственной программой), за исключением специального значения, показывающего, что поток продолжает выполняться (это значение `STILL_ACTIVE`, равное `0x00000103L`).

- Второй параметр – указатель `pParam`, передаваемый в функцию потока, может содержать адрес простого значения (типа `int`) или структуры, включающей произвольное количество информации для нового потока. Новый поток передает информацию в начальный поток (в дополнение к коду возврата), присваивая значения переменным, на которые указывает параметр `pParam`. Для выполнения описанной процедуры работа потоков должна быть *синхронизирована* (см. ниже).
- Параметр `nPriority` задает *приоритет* нового потока, который определяет, как часто поток будет выполняться при переключении управления от одного потока к другому. Если задачу необходимо выполнить быстро и эффективно, то потоку присваивается относительно высокий приоритет. При выполнении менее важной задачи (которую можно отложить до того момента, когда другие потоки будут неактивны) присвойте потоку относительно низкий приоритет. В большинстве случаев потоку присваивается стандартное значение `THREAD_PRIORITY_NORMAL` – средний приоритет. Описание различных значений приоритетов приведено в справочной системе.
- Чтобы новый поток начал выполняться немедленно, следует параметру `dwCreateFlags` присвоить стандартное значение 0. Если же присвоено значение `CREATE_SUSPENDED`, то он не запустится до вызова функции `CWinThread::ResumeThread`, описанной ниже.
- Параметрам `nStack` и `lpSecurityAttrs` почти всегда присваиваются стандартные значения, определяющие размер стека для потока и атрибуты защиты.

Приведенный ниже фрагмент программы запускает новый поток, выполняющий функцию `ThreadFunction`.

```
UINT ThreadFunction (LPVOID pParam)
{
    // операторы и вызовы функций, которые должны быть
    // выполнены новым потоком ...
```

```

    return 0; // окончание потока и передача кода возврата 0
}

// ...

void SomeFunction (void)
{
    // ...
    int Code = 1;
    CWinThread *PWinThread;
    PWinThread = AfxBeginThread (ThreadFunction, &Code);
    // ...
}

```

## Завершение потока

В Visual C++ предусмотрены два способа завершения нового потока:

- Первый способ завершения: можно просто *возвратиться из потока в функцию потока* (функция ThreadFunction в примере выше), передавая код возврата. Это наиболее правильный способ завершения потока. Стек, используемый потоком, освобождается и все автоматические данные, созданные потоком, удаляются, т.е. для автоматических объектов вызываются деструкторы.
- Второй способ завершения: поток может *вызвать MFC-функцию AfxEndThread*, передавая ей нужный код возврата:

```
void AfxEndThread (UINT nExitCode);
```

Это удобный способ немедленного завершения потока из вложенной функции (вместо возврата в исходную функцию потока). При этом стек потока освобождается, но *деструкторы для автоматических объектов не вызываются*.

Оба способа завершения потока выполняются самим потоком. При необходимости завершить поток из *другого* потока рекомендуется передать из последнего сигнал в завершаемый поток, по которому он сам себя завершает. Ниже показаны различные способы установки связей между потоками. Для прекращения другого потока можно также использовать функцию Win32 API ::TerminateThread.

## Функции управления потоком

Новым потоком управляет указатель на объект класса CWinThread, который возвращается функцией AfxBeginThread. В приведенных ниже примерах предполагается, что PWinThread содержит указатель типа класса CWinThread, возвращаемый функцией AfxBeginThread.

- Для *временной приостановки* выполнения потока вызывается функция SuspendThread класса CWinThread.

```
PWinThread->SuspendThread ();
```

- Чтобы *продолжить выполнение* приостановленного потока можно вызвать функцию CWinThread::ResumeThread. Функция ResumeThread также вызывается для запуска потока, созданного в приостановленном состоянии значением CREATE\_SUSPEND параметра dwCreateFlags функции AfxBeginThread.

```
PWinThread->ResumeThread ();
```

- Чтобы *получить текущее значение приоритета* потока, следует вызвать функцию `CWinThread::GetThreadPriority`.
- Чтобы *изменить приоритет* потока, изначально заданный в вызове функции `AfxBeginThread`, следует вызвать функцию `CWinThread::SetThreadPriority`. Например, следующая строка задает приоритет выше стандартного.

```
PWinThread->SetThreadPriority (THREAD_PRIORITY_ABOVE_NORMAL);
```

- Чтобы определить, *продолжается ли выполнение потока*, вызывается функция Win32 API `::GetExitCodeThread`. Если выполнение завершено, она выдает код возврата (т.е. значение, возвращенное из функции потока или переданное функции `AfxEndThread`). В первом параметре функции потока `::GetExitCodeThread` необходимо задать дескриптор Windows, который хранится в переменной `m_hThread` класса `CWinThread`. В приведенном ниже примере функция `::GetExitCodeThread` присваивает значение переменной типа `DWORD`, адрес которой передается во втором параметре. Если поток продолжает выполнение, то переменной присваивается значение `STILL_ACTIVE`, равное `0x00000103L`. Если поток завершился – присваивается значение кода возврата.

```
DWORD ExitCode;

::GetExitCodeThread
(PWinThread->m_hThread,      // дескриптор Windows для потока;
 &ExitCode);                // адрес переменной DWORD для
                             // получения кода возврата

if (ExitCode == STILL_ACTIVE)
    // поток продолжает выполнение...
else
    // поток завершен, и переменная ExitCode содержит код возврата
```

После завершения потока объект класса `CWinThread`, поддерживаемый функцией `AfxEndThread`, *автоматически удаляется*. При попытке доступа к объекту класса `CWinThread` после выхода из потока программа сгенерирует ошибку защиты. Однако чтобы определить, завершен ли поток, необходимо получить доступ к переменной `m_hThread` класса `CWinThread`. Решение этой проблемы состоит в *предотвращении* автоматического удаления объекта класса `CWinThread` при завершении потока. Для этого присвойте значение `FALSE` переменной `m_bAutoDelete` класса `CWinThread`, как показано ниже.

```
PWinThread = AfxBeginThread
(ThreadFunction,
 &Code,
 THREAD_PRIORITY_ NORMAL,
 0,
 CREATE_SUSPENDED);
PWinThread->m_bAutoDelete = FALSE;
PWinThread->ResumeThread ();
```

В этом примере, запустив поток в состоянии ожидания, мы получаем возможность *перед* выходом из потока установить значение переменной `m_bAutoDelete`, равное `FALSE`. Если переменной `m_bAutoDelete` присвоено значение `FALSE`, объект класса `CWinThread` будет существовать даже после выхода из потока, а программа сможет вызвать функцию `::GetExitCodeThread`, чтобы определить, выполнен ли поток, и получить код его возврата. Если поток выполнен, то вызов функций класса `CWinThread` и функции `SuspendThread` на работе программы не отразится, но

последняя функция не будет генерировать ошибку защиты! После окончания работы с объектом потока его можно удалить явно, воспользовавшись оператором

```
delete PWinThread;
```

## Ограничения на использование MFC-классов

Выполняемые в одном процессе потоки совместно используют определенную область памяти и глобальные или динамические переменные и объекты. Однако существует два ограничения на использование объектов MFC-классов.

- Два потока не должны *одновременно* получать доступ к *одному объекту MFC*. Например, два потока могут использовать один объект класса `CString` при условии, что они не будут одновременно вызывать функцию класса `CString`. Чтобы запретить одновременный доступ к объекту, можно использовать один из методов синхронизации, рассмотренных в следующем разделе. Два потока могут одновременно получить доступ к разным объектам, принадлежащим даже одному MFC-классу, например, к двум объектам класса `CString`.
- Объект любого из классов `CWnd`, `CDC`, `CMenu`, `CGdiObject` или объект класса, наследуемый от них, доступен только для потока, создавшего объект. Эти классы имеют общее свойство: каждый из них хранит дескриптор для некоторых подчиненных объектов. Например, объект класса `CWnd` хранит дескриптор окна, который хранится в переменной `m_hWnd` объекта. Если поток создает объект одного из этих классов, то при получении доступа к подчиненному элементу из второго потока необходимо передать ему дескриптор, после чего второй поток должен создать свой собственный объект. Например, если поток создает объект класса `CWnd` для управления окном, а рисовать в этом окне требуется с помощью второго потока, то вместо передачи во второй поток объекта класса `CWnd`, ему передается дескриптор окна, хранящийся в переменной `m_hWnd` объекта. Второй поток затем вызывает функцию `FromHandle` класса `CWnd`, чтобы создать собственный объект этого класса `CWnd`.

Описанные в этой главе вторичные потоки, называют *рабочими потоками*. Они выполняют фоновую или вторичную задачу, в то время как главный поток продолжает обрабатывать сообщения. С другой стороны, можно вызвать альтернативную версию функции `AfxWinThread`, использующую указатель `C_RUNTIME_CLASS` в качестве первого параметра, и создать поток *пользовательского интерфейса*, разработанного для создания одного или более дополнительных окон и обработки сообщений, передаваемых в эти окна.

Чтобы вызвать альтернативную версию функции `AfxWinThread`, постройте класс для управления потоком из класса `CWinThread` и переопределите в этом классе виртуальную функцию `InitInstance` класса `CWinThread` (см. документацию на класс `CWinThread` и функцию `AfxBeginThread`). Эта функция должна создавать любые окна, отображаемые потоком, и выполнять любые задачи инициализации. Функция `InitInstance` похожа на одноименную функцию класса приложения. Обратите внимание: класс приложения, наследуемый косвенно от класса `CWinThread`, управляет *первичным* потоком программы – потоком пользовательского интерфейса. Создание потоков пользовательского интерфейса описано в справочной системе.

## Способы синхронизации потоков

Многопоточковый режим для 32-битовых программ, выполняемых в среде Windows 95 (и последующих версий) или Windows NT (и последующих версий), учитывает *приоритет* отдельных потоков. Это означает, что выполнение данного потока может быть прервано в промежутке времени между выполнением любых двух машинных команд, и управление перейдет к другому потоку, причем

невозможно предсказать, когда этот поток будет прерван. Кроме того, потоки выполняются *асинхронно*, т.е. при выполнении одним потоком отдельной команды нельзя предсказать, какой оператор другого потока выполняется в данный момент. При работе с несколькими потоками это может создать проблемы в том случае, если два или более потоков получают доступ к таким общим ресурсам, как глобальные переменные. В дальнейшем термин *ресурсы* используется в широком смысле и означает некие компоненты, являющиеся собственностью процесса. Он может, например, указывать на переменную, объект C++, участок памяти, графический объект или аппаратное устройство. Рассмотрим пример.

```
// глобальное объявление:
int Count = 0;

// ...

// функция, выполняемая двумя потоками
void SharedFunctions ()
{
    // ...
    ++Count;
    cout << Count << '\n';
    // ...
}
```

В этом примере два потока выполняют блок программы, увеличивающий содержимое глобального счетчика на 1, а затем печатающий новое значение счетчика. При повторных вызовах этой функции можно ожидать, что результатом будет печать последовательных целочисленных значений, начиная с 1. Однако, поскольку программа выполняется двумя потоками, события могут развиваться иначе. Например, так. Вначале первый поток наращивает счетчик Count. Затем первый поток приостанавливается, а второй поток получает управление. Второй поток инкрементирует счетчик Count и печатает новое значение. Затем первый поток снова получает управление и печатает значение, уже напечатанное вторым потоком. В результате пропускается значение, которое *нужно* было напечатать. Чтобы предотвратить подобную ошибку, нужно синхронизировать действия отдельных потоков.

Существует много других ситуаций, в которых может понадобиться синхронизация потоков:

- Существуют ресурсы, *не допускающие одновременного доступа* несколькими потоками, например, объекты MFC-классов (как упоминалось ранее), графические объекты, не предназначенные для совместного использования файлы и аппаратные средства.
- Может потребоваться синхронизировать действия *потока-производителя* и *потока-потребителя*. Например, если один поток записывает символы в буфер, другой читает и удаляет символы из буфера, то читающему потоку требуется подождать, пока записывающий поток добавит символ, а записывающему потоку подождать, пока читающий поток удалит символ.

Win32 API предлагает для этих целей набор *объектов синхронизации*, использующихся для синхронизации отдельных потоков (в этом контексте термин *объект* ссылается не на объект C++). Используя эти объекты, можно организовать синхронизацию потоков:

- запрет *одновременного* доступа к ресурсам более одного потока;
- *ограничение* количества потоков, одновременно получающих доступ к ресурсам;
- организацию *сигнализации* между потоками.

Одно из преимуществ использования объекта синхронизации состоит в том, что пока поток ждет освобождения объекта, он блокируется. При блокировании потока операционная система не выполняет его, следовательно, он не занимает процессорное время, которое можно использовать для других потоков.

В состав MFC-классов входят и классы для объектов синхронизации управления, которые используются вместо вызова функций Win32 API, рассмотренных в этой и следующей главах. Свойства этих классов менее понятны, чем функции Win32 API, и они не допускают использования некоторых приемов программирования, например, ожидания прерывания потока или процесса. В MFC существуют такие классы синхронизации: CSyncObject, CSemaphore, CCriticalSection, CMutex, CEvent, CSingleLock, CMultiLock. Использование этих классов описано в справочной системе и в документации на классы.

## Мьютекс и другие объекты синхронизации

К числу наиболее простых и типичных объектов синхронизации относится *мьютекс*. Его название происходит от английского выражения *mutual exclusion* (*взаимное исключение*). Этот объект используется для ограничения одновременного доступа к данному ресурсу *единственным* потоком. При использовании мьютекса вызывается функция `::CreateMutex` Win32 API для создания объекта синхронизации – мьютекса. Это можно сделать в любом потоке программы:

```
HANDLE hMutex;    // объявляется глобально, и все потоки
                  // имеют к нему доступ

//...

void SomeFunction ()
{
    //...
    hMutex = ::CreateMutex
        (NULL,    // присваивает стандартные атрибуты защиты
         // (ненаследуемый дескриптор)
         FALSE,   // мьютекс изначально свободный
         NULL);  // имя мьютексу не присваивается
    //...
}
```

- При создании мьютекса первый параметр задает его *атрибуты защиты*. При передаче значения NULL ему присваиваются стандартные значения атрибутов, а дескриптор мьютекса устанавливается ненаследуемым (наследование дескриптора рассмотрено в гл. 23).
- Во втором параметре описывается *начальное состояние* мьютекса. Передача значения TRUE создает изначально свободный мьютекс (при передаче значения FALSE мьютекс изначально будет занят; эти состояния рассмотрены далее).
- Третий параметр задает *имя* мьютекса. Значение NULL создает мьютекс без имени. Задание имени позволяет получать доступ к мьютексу из другого процесса (гл. 23).

Функция `::CreateMutex` возвращает дескриптор, используемый для ссылки на мьютекс потоками программы.

Для реализации механизмов синхронизации следует добавить вызов функции `::WaitForSingleObject` Win32 API в начало каждого блока, обращающегося к защищаемым ресурсам, и вызов функции Win32 API `::ReleaseMutex` в конец каждого из этих блоков. Например, можно запретить одновременный доступ нескольких потоков к глобальному счетчику так:

```
::WaitForSingleObject
    (hMutex,          // дескриптор мьютекса
     INFINITE);       // ждите столько, сколько нужно
++Count;
cout << Count << '\n';
::ReleaseMutex (hMutex);
```

Мьютекс, как и любой другой объект синхронизации, находится в одном из двух состояний: *свободном (signaled)* или *занятом (nonsignaled)*. Как правило, вновь созданный мьютекс свободен. Если при вызове потоком функции `::WaitForSingleObject` мьютекс свободен, то функция переводит мьютекс в занятое состояние и сразу же возвращает управление обратно. Поток продолжает выполнять защищенный блок, после чего вызывает функцию `::ReleaseMutex` и устанавливает мьютекс снова в свободное состояние. Если мьютекс занят при вызове функции `::WaitForSingleObject` (другой поток в это время выполняется в защищенном блоке), то эта функция сначала *ждет*, когда мьютекс станет свободным, а затем устанавливает мьютекс в занятое состояние и возвращает управление. В результате только один поток может выполнять защищенный блок кода в один момент времени.

Когда объект синхронизации занят, считается, что он *принадлежит* потоку, который перевел его в занятое состояние. Например, если *несколько* блоков используют счетчик Count, нужно окружить каждый блок вызовами функций `::WaitForSingleObject` и `::ReleaseMutex`, задавая один и тот же мьютекс в каждом вызове. В этом случае, если один поток выполняет *любой* такой блок кода, то другой – не может выполнить ни один из этих блоков. Параметры, передаваемые в функции `::WaitForSingleObject` и `::ReleaseMutex`:

- Первый параметр – это *дескриптор мьютекса*, возвращаемый функцией `::CreateMutex`.
- Второй параметр функции `::WaitForSingleObject` – это *период ожидания* в миллисекундах. После ожидания в течение указанного периода времени `::WaitForSingleObject` возвратит управление, даже если мьютекс не свободен. Однако большинство программ присваивают этому параметру специальное значение INFINITE, после чего функция `::WaitForSingleObject` ожидает столько, сколько необходимо для освобождения мьютекса.

Завершив использование мьютекса, программа может вызвать функцию Win32 API `::CloseHandle`, чтобы закрыть его дескриптор. Например:

```
::CloseHandle (hMutex);
```

Система автоматически закрывает дескриптор при выходе из программы, если функция `::CloseHandle` не вызывалась. Когда все дескрипторы для данного объекта будут закрыты, система закроет его и освободит память, которую он использовал. Открытие нескольких дескрипторов для одного объекта рассмотрено в гл. 23.

Другие объекты синхронизации тоже можно использовать в системе Win32. Доступны критические секции, семафоры и события.

- *Критические секции* выполняют те же задачи, что и мьютексы, но для них вызываются другие функции Win32. Критические секции немного эффективнее мьютексов, но их нельзя использовать для синхронизации потоков в различных процессах. Другие объекты синхронизации могут совместно использоваться отдельными процессами.
- *Семафоры* похожи на мьютексы с той лишь разницей, что допускают одновременный доступ к ресурсам *нескольких* потоков. При создании семафора указывается максимальное число допустимых потоков.
- *Событие* – это многофункциональный объект синхронизации, который позволяет одному потоку передавать сигналы одному или нескольким потокам.

В табл. 22.1 перечислены объекты синхронизации Win32, кратко описано назначение каждого из них и приведены функции Win32 API, используемые для создания объекта и управления им.

Функции `::WaitForSingleObject` и `::WaitForMultipleObject` – общие функции, ожидающие любой из объектов синхронизации. Функцию `::WaitForSingleObject` Win32 API можно вызвать для *нескольких* различных мьютексов или других объектов синхронизации. Можно ждать, пока *один* из них станет свободным либо пока *все* объекты станут свободными. При вызове функций `::WaitForSingleObject` или `::WaitForMultipleObject` к ожиданию доступа к

объектам синхронизации может добавиться ожидание доступа к объектам Windows других типов. В этом контексте *объектом* называется элемент Windows, предоставленный дескриптором, но не экземпляр класса C++. В табл. 22.2 перечислены эти объекты. Для каждого из них приведены функции или переменные, с помощью которых можно получить дескриптор объекта, и описание события, освобождающего объект. Для ожидания освобождения объекта передайте дескриптор объекта в функцию `::WaitForSingleObject` или `::WaitForMultipleObjects`.

Табл. 22.1. Объекты синхронизации Win32

Объект синхронизации	Основное назначение объекта	Функции Win32 API
Критическая секция	Запрет доступа нескольким потокам к общим ресурсам. Эффективна, но не может быть использована процессами	<code>::InitializeCriticalSection</code> <code>::EnterCriticalSection</code> <code>::LeaveCriticalSection</code> <code>::DeleteCriticalSection</code>
Мьютекс	Запрет доступа нескольким потокам к общим ресурсам	<code>::CreateMutex</code> <code>::WaitForSingleObject</code> <code>::WaitForMultipleObject</code> <code>::ReleaseMutex</code> <code>::CloseHandle</code>
Семафор	Ограничение числа потоков, которые могут иметь одновременный доступ к общим ресурсам	<code>::CreateSemaphore</code> <code>::WaitForSingleObject</code> <code>::WaitForMultipleObject</code> <code>::ReleaseSemaphore</code> <code>::CloseHandle</code>
Событие	Передача потоком сигналов одному или нескольким другим потокам	<code>::CreateEvent</code> <code>::SetEvent</code> <code>::PulseEvent</code> <code>::ResetEvent</code> <code>::WaitForSingleObject</code> <code>::WaitForMultipleObject</code> <code>::CloseHandle</code>

Табл. 22.2. Дополнительные объекты Windows, вызываемые функциями `::WaitForSingleObject` или `::WaitForMultipleObjects`

Объект Windows	Функция или переменная для получения дескриптора объекта	Событие, по которому объект становится свободным
Ввод с клавиатуры	<code>::CreateFile</code> или <code>::GetStdHandle</code>	Доступен ввод с консоли
Обмен уведомлениями	<code>::FindFirstChangeNotifications</code>	Обмен в специальном каталоге
Поток	<code>CWinThread::m_hThread</code>	Прекращение потока
Процесс	<code>::CreateProcess</code> или <code>::OpenProcess</code>	Прекращение процесса

Для ожидания различных событий потоки могут вызывать несколько дополнительных функций Win32 API:



- Вызывая функцию `::MsgWaitForMultipleObjects`, поток может ждать *либо* освобождения одного или более объектов, *либо* получения одного или нескольких указанных сообщений.
- Поток может вызвать функцию `::Sleep` для ожидания в течение определенного периода времени (в миллисекундах).

Обратите внимание: при ожидании потока вызовом функций `::MsgWaitForMultipleObjects` или `::Sleep` он блокируется и не занимает процессорное время (как при вызове функции `::WaitForSingleObject` или `::WaitForMultipleObjects`). Наконец, поток может передавать сигналы другим потокам или синхронизировать их работу, используя глобальную переменную.

## ***Многопотоковая программа рисования ImpFractal***

В программе `FractalView` (см. гл. 19) первичный поток программы рисует один столбец рекурсивного изображения и обрабатывает любое поступающее сообщение. Затем рисует другой столбец и обрабатывает сообщение и так далее. Эта процедура позволяет программе отвечать на сообщения, *пока* рисуется изображение (эта прорисовка может занимать много времени). В многопотоковой версии программы, названной `ImpFractal`, показано, что в случае, когда программа должна одновременно решать несколько задач, эффективнее и легче задать для каждой главной задачи отдельный поток. В программе `ImpFractal` первичный поток создает сообщения, а вторичный – рисует рекурсивное изображение. Новая программа реагирует на запросы быстрее, чем `FractalView`, так как первичный поток может обработать сообщение, не ожидая окончания рисования текущего столбца изображения. Программа `ImpFractal` проще еще и потому, что код, выполняемый каждым потоком, решает только собственную задачу, не переключаясь на другие задачи. Например, программа рисования рекурсивного изображения просто рисует его целиком, не сохраняя его состояние и не выполняя после рисования каждого столбца возврат.

Многопотоковый режим в среде Windows сопряжен с некоторыми программными сложностями, которыми нужно управлять. Например, вторичному потоку необходимо запретить рисовать в окне, пока первичный поток передвигает или модифицирует окно. Кроме того, отдельные потоки не могут свободно использовать объекты MFC-классов (см. выше).

Программа `ImpFractal` создается с использованием мастера `Application Wizard` для генерации *нового* набора исходных файлов (вместо изменения исходных файлов программы `FractalView`). При необходимости в программу `ImpFractal` всегда можно скопировать фрагмент из программы `FractalView`. При генерации исходных файлов назовите проект именем `ImpFractal`, а на вкладках диалогового окна мастера `Application Wizard` выберите такие же установки, что и в программе `WinHello` (см. гл. 9), но отключите создание строки состояния и панели инструментов.

Сгенерировав исходные коды, откройте файл `ImpFractalView.h` и определите переменную `m_DrawThread` в классе представления, используемую для хранения адреса объекта, управляющего вторичным потоком.

```
class CImpFractalView : public CView
{
public:
    CWinThread *m_DrawThread;
```

В файле `ImpFractalView.cpp` добавьте код для инициализации экземпляра `m_DrawThread` в конструктор класса представления.

```
CImpFractalView::CImpFractalView()
{
    // TODO: добавьте сюда код конструктора
```

```

    m_DrawThread = 0;
}

```

Кроме того, добавьте фрагмент программы деструктора класса для удаления объекта потока.

```

CImpFractalView::~CImpFractalView()
{
    delete m_DrawThread;
}

```

Добавьте следующие определения в начало файла ImpFractalView.cpp.

```

// ImpFractalView.cpp : реализация класса CImpFractalView
//

#include "stdafx.h"
#include "ImpFractal.h"

#include "ImpFractalDoc.h"
#include "ImpFractalView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// константы-параметры фрактала
#define CIMAX 1.2
#define CIMIN -1.2
#define CRMAX 1.0
#define CRMIN -2.0
#define NMAX 128

// глобальные переменные для связи потоков между собой
int ColMax;
int RowMax;
BOOL StopDraw;

```

Для хранения текущих размеров окна представления используются переменные ColMax и RowMax. В программе ImpFractal это – глобальные переменные, а не переменные класса представления. Таким образом оба потока могут иметь к ним свободный доступ (вспомните ограничения на доступ к объектам MFC-классов множества потоков). Константы множества Мандельброта (CIMAX и другие) такие же, как определенные в программе FractalView. Переменная StopDraw – это флажок, используемый первичным потоком для передачи сигнала вторичному потоку (для немедленного возврата без завершения рисования текущего изображения).

В классе ImpFractalView создайте обработчик сообщений WM\_SIZE. Добавьте в созданный обработчик код, приведенный ниже. Функция OnSize получает управление при первом создании окна и изменении его размеров, а также устанавливает значения глобальных переменных RowMax и ColMax, используемых программой рисования для определения общего размера окна представления.

```

void CImpFractalView::OnSize(UINT nType, int cx, int cy)
{
    CView::OnSize(nType, cx, cy);

    // TODO: Добавьте сюда собственный код обработчика

    if (cx <= 1 || cy <= 1) return;
}

```

```

    ColMax = cx;
    RowMax = cy;
}

```

Добавьте следующий фрагмент в функцию OnDraw файла FractalViewView.cpp. Вместо того чтобы рисовать рекурсивное изображение самостоятельно, функция OnDraw запускает вторичный поток, а затем быстро возвращает управление. Таким образом, пока вторичный поток рисует рекурсивное изображение, программа сможет продолжать обработку сообщений в фоновом режиме.

```

void CImpFractalView::OnDraw(CDC* pDC)
{
    CImpFractalDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    // TODO: добавьте сюда код прорисовки собственных данных

    if (m_DrawThread)
    {
        StopDraw = TRUE;
        m_DrawThread->ResumeThread ();
        ::WaitForSingleObject
        (m_DrawThread->m_hThread, INFINITE);
        // получения управления потоком рисования ждать столько,
        // сколько нужно
        delete m_DrawThread;
    }

    m_DrawThread = AfxBeginThread
        (DrawFractal,
         &m_hWnd,
         THREAD_PRIORITY_BELOW_NORMAL,
         0,
         CREATE_SUSPENDED);
    m_DrawThread->m_bAutoDelete = FALSE;
    StopDraw = FALSE;
    m_DrawThread->ResumeThread ();
}

```

Если указатель на объект потока m\_DrawThread имеет ненулевое значение, значит вторичный поток уже был запущен. В этом случае функция OnDraw начинается с *остановки* потока и удаления его объекта. Для этого значение глобального флажка StopDraw устанавливается в TRUE. Функция вторичного потока проверяет этот флажок и завершается, если его значение равно TRUE. Затем для объекта потока вызывается функция ResumeThread, чтобы повторно запустить поток *в случае его приостановки* (позже вы узнаете, как приостанавливается поток). Для ожидания естественного завершения потока вызывается функция ::WaitForSingleObject. Как только поток закончит выполнение, он удаляет свой объект, т.е. объект класса CWinThread, адрес которого сохранен в переменной m\_DrawThread. Обратите внимание: если поток уже завершил выполнение (он автоматически останавливает выполнение после окончания рисования), то описанные действия бесполезны, но не могут привести к ошибке выполнения программы.

Функция OnDraw создает *новый* поток (как только старый поток рисования остановлен), чтобы нарисовать или перерисовать рекурсивное изображение, используя размеры текущего окна представления. При вызове функции AfxBeginThread функция OnDraw присваивает:

- первому параметру – адрес функции вторичного потока DrawFractal;
- второму параметру – дескриптор окна представления (чтобы вторичный поток мог рисовать внутри окна);
- третьему параметру – значение THREAD\_PRIORITY\_BELOW\_NORMAL, чтобы поток рисования имел приоритет меньший, чем у первичного. Это позволяет первичному потоку быстро реагировать на сообщения, а вторичному потоку – рисовать, когда первичный поток не занят;
- пятому параметру (AfxBeginThread) – значение CREATE\_SUSPENDED, что позволяет присвоить значение FALSE переменной m\_bAutoDelete объекта потока перед его завершением. Как уже упоминалось в разделе “Функции управления потоком”, переменной m\_bAutoDelete присваивается значение FALSE, чтобы объект потока не уничтожал себя после окончания выполнения потока. Программе необходим доступ к объекту потока, независимо от продолжения выполнения потока.

Добавьте определение функции рисования DrawFractal в функцию файл ImpFractalView.cpp. Вторичный поток программы выполняет функцию DrawFractal, напоминающую функцию CFractalViewView::DrawCol в программе FractalView. Однако (вместо возврата после рисования каждого столбца изображения) она возвращает управление, когда нарисовано все рекурсивное изображение *или* когда первичный поток программы присвоит флажку StopDraw значение TRUE.

```
// Прорисовка класса CImpFractalView

UINT DrawFractal (LPVOID PHWndView)
{
    float CI;
    CClientDC ClientDC (CWnd::FromHandle (*(HWND *)PHWndView));
    int Col;
    static DWORD ColorTable [6] =
        {0x0000ff,      // красный
         0x00ff00,      // зеленый
         0xff0000,      // синий
         0x00ffff,      // желтый
         0xffff00,      // бирюзовый
         0xff00ff};     // сиреневый
    int ColorVal;
    float CR = (float)CRMIN;
    float DCI = (float)((CIMAX - CIMIN) / (RowMax-1));
    float DCR = (float)((CRMAX - CRMIN) / (ColMax-1));
    float I;
    float ISqr;
    float R;
    int Row;
    float RSqr;

    for (Col = 0; Col < ColMax; ++Col)
    {
        if (StopDraw) break;

        CI = (float)CIMAX;

        for (Row = 0; Row < RowMax; ++Row)
        {
            R = (float)0.0;
```

```

I = (float)0.0;
RSqr = (float)0.0;
ISqr = (float)0.0;
ColorVal = 0;
while (ColorVal < NMAX && RSqr + ISqr < 4)
{
    ++ColorVal;
    RSqr = R * R;
    ISqr = I * I;
    I *= R;
    I += I + CI;
    R = RSqr - ISqr + CR;
}
ClientDC.SetPixelV (Col, Row, ColorTable [ColorVal % 6]);
CI -= DCI;
}

CR += DCR;
}

return (0);
}

```

При возвращении управления функцией DrawFractal вторичный поток прерывается. Так как DrawFractal не является функцией класса представления, то при создании объекта контекста устройства она не может просто передать указатель this в конструктор CClientDC, а передает дескриптор окна представления (содержащийся в параметре функции DrawFractal) в функцию CWnd::FromHandle для получения указателя на новый временный объект окна представления. Затем этот указатель передается в конструктор класса CClientDC. Функция OnDraw передает функции DrawThread дескриптор Windows для окна представления вместо передачи указателя на объект окна представления.

Обратите внимание: при перемещении или изменении размеров окна программы вторичный поток продолжает рисовать в окне в то время, когда главный поток внутри системного кода управляет названными операциями. При выполнении таких операций *результаты в окне не появятся*, а после перемещения или изменения размеров окна изображение будет содержать пробелы. Если хотите это увидеть, скомпилируйте и выполните программу. В некоторых версиях рабочего стола Windows можно установить параметр Show Window Content While Dragging, выбрав команду Folder Options в меню Start подменю Setting панели задач Windows и открыв в появившемся окне вкладку View. При выборе этого параметра графическое изображение, нарисованное в окне FractalView, будет отображаться в нормальном виде, и в нем не будут возникать зазоры, но при этом возможна потеря плавности перемещения окна по экрану (если быстродействие графического процессора недостаточно для быстрой прорисовки).

## Текст программы ImpFractal

Текст программы ImpFractal (многопоточковой версии программы ImpFractal) приведен в листингах 22.1—22.8.

---

### Листинг 22.1.

```

// ImpFractal.h : главный заголовочный файл программы ImpFractal
//

```

```

#pragma once

#ifndef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"          // основные системы

// CImpFractalApp:
// Реализация данного класса - в файле ImpFractal.cpp
//

class CImpFractalApp : public CWinApp
{
public:
    CImpFractalApp();

    // Переопределения
public:
    virtual BOOL InitInstance();

    // Реализация
    afx_msg void OnAppAbout();
    DECLARE_MESSAGE_MAP()
};

extern CImpFractalApp theApp;

```

---

## Листинг 22.2.

```

// ImpFractal.cpp : Определяет поведение классов приложения
//

#include "stdafx.h"
#include "ImpFractal.h"
#include "MainFrm.h"

#include "ImpFractalDoc.h"
#include "ImpFractalView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CImpFractalApp

BEGIN_MESSAGE_MAP(CImpFractalApp, CWinApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
    // Стандартные команды работы с файлами документов
    ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
END_MESSAGE_MAP()

// CImpFractalApp

CImpFractalApp::CImpFractalApp()

```

```

{
    // TODO: добавьте сюда собственный код конструктора.
    // Поместите весь существенный код инициализации
    // в функцию InitInstance
}

// Единственный объект класса CImpFractalApp

CImpFractalApp theApp;

// Инициализация CImpFractalApp

BOOL CImpFractalApp::InitInstance()
{
    CWinApp::InitInstance();

    // Стандартная инициализация.
    // Если вы не используете какие-то из возможностей и хотите
    // уменьшить размер оконечного исполняемого модуля,
    // удалите отдельные процедуры инициализации из последующего
    // кода.
    // Измените строку-аргумент функции (ключ в реестре,
    // под которым хранятся в реестре ваши установки).
    // TODO: измените эту строку на что-нибудь подходящее,
    // например, название вашей компании или организации
    SetRegistryKey(_T("Local AppWizard-Generated Applications"));
    LoadStdProfileSettings(4); // Загрузка установок из INI-файла
                               // (включая MRU)
    // Регистрация шаблонов документов приложения. Шаблоны
    // документов служат связью между документами, окнами документов
    // и окнами представлений
    CSingleDocTemplate* pDocTemplate;
    pDocTemplate = new CSingleDocTemplate(
        IDR_MAINFRAME,
        RUNTIME_CLASS(CImpFractalDoc),
        RUNTIME_CLASS(CMainFrame), // главное окно
                                   // SDI-приложения
        RUNTIME_CLASS(CImpFractalView));
    AddDocTemplate(pDocTemplate);
    // Поиск в командной строке команд управления, DDE,
    // открытия файлов
    CCommandLineInfo cmdInfo;
    ParseCommandLine(cmdInfo);
    // Выполнение команд, указанных в командной строке.
    // Вернет FALSE, если приложение запускалось с /RegServer,
    // /Register, /Unregserver или /Unregister.
    if (!ProcessShellCommand(cmdInfo))
        return FALSE;
    // Прорисовка и обновление единственного
    // проинициализированного окна
    m_pMainWnd->ShowWindow(SW_SHOW);
    m_pMainWnd->UpdateWindow();
    return TRUE;
}

// CAboutDlg диалог, используемый в App About

```

```

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

    // Данные для диалога
    enum { IDD = IDD_ABOUTBOX };

protected:
    virtual void DoDataExchange(CDataExchange* pDX);
    // поддержка DDX/DDV

    // Реализация
protected:
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
END_MESSAGE_MAP()

// Команда приложения на запуск диалога
void CImpFractalApp::OnAppAbout()
{
    CAboutDlg aboutDlg;
    aboutDlg.DoModal();
}

// Обработчики сообщений класса CImpFractalApp

```

---

### Листинг 22.3.

```

// ImpFractalDoc.h : интерфейс класса CImpFractalDoc
//

#pragma once

class CImpFractalDoc : public CDocument
{
protected: // используется только для сериализации
    CImpFractalDoc();
    DECLARE_DYNCREATE(CImpFractalDoc)

    // Атрибуты
public:

    // Операции

```



```

public:

// Переопределения
public:
    virtual BOOL OnNewDocument();
    virtual void Serialize(CArchive& ar);

// Реализация
public:
    virtual ~CImpFractalDoc();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Сгенерированные функции карты сообщений
protected:
    DECLARE_MESSAGE_MAP()
};

```

---

#### Листинг 22.4.

```

// ImpFractalDoc.cpp : реализация класса CImpFractalDoc
//

#include "stdafx.h"
#include "ImpFractal.h"

#include "ImpFractalDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CImpFractalDoc

IMPLEMENT_DYNCREATE(CImpFractalDoc, CDocument)

BEGIN_MESSAGE_MAP(CImpFractalDoc, CDocument)
END_MESSAGE_MAP()

// Конструктор/деструктор класса CImpFractalDoc

CImpFractalDoc::CImpFractalDoc()
{
    // TODO: добавьте сюда одноразовый код конструктора
}

CImpFractalDoc::~CImpFractalDoc()
{
}

BOOL CImpFractalDoc::OnNewDocument()

```

```

{
    if (!CDocument::OnNewDocument())
        return FALSE;

    // TODO: добавьте сюда код реинициализации
    // (SDI-документы будут использовать этот документ
    // многократно)

    return TRUE;
}

// Сериализация класса CImpFractalDoc

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

// Диагностика CImpFractalDoc

#ifdef _DEBUG
void CImpFractalDoc::AssertValid() const
{
    CDocument::AssertValid();
}

void CImpFractalDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}
#endif // _DEBUG

// Команды CImpFractalDoc

```

---

### Листинг 22.5.

```

// ImpFractalView.h : интерфейс класса CImpFractalView
//

#pragma once

class CImpFractalView : public CView
{
public:
    CWinThread *m_DrawThread;

protected: // используется только для сериализации
    CImpFractalView();
    DECLARE_DYNCREATE(CImpFractalView)

```

```

// Атрибуты
public:
    CImpFractalDoc* GetDocument() const;

// Операции
public:

// Переопределения
public:
    virtual void OnDraw(CDC* pDC);
    // переопределена для прорисовки этого вида
virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
protected:

// Реализация
public:
    virtual ~CImpFractalView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Сгенерированные функции карты сообщений
protected:
    DECLARE_MESSAGE_MAP()
public:
    afx_msg void OnSize(UINT nType, int cx, int cy);
};

#ifdef _DEBUG // отладочная версия в файле ImpFractalView.cpp
inline CImpFractalDoc* CImpFractalView::GetDocument() const
    { return reinterpret_cast<CImpFractalDoc*>(m_pDocument); }
#endif

```

---

## Листинг 22.6.

```

// ImpFractalView.cpp : реализация класса CImpFractalView
//

#include "stdafx.h"
#include "ImpFractal.h"

#include "ImpFractalDoc.h"
#include "ImpFractalView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// константы-параметры фрактала
#define CIMAX 1.2
#define CIMIN -1.2
#define CRMAX 1.0

```

```

#define CRMIN -2.0
#define NMAX 128

// глобальные переменные для связи потоков между собой
int ColMax;
int RowMax;
BOOL StopDraw;

// CImpFractalView

IMPLEMENT_DYNCREATE(CImpFractalView, CView)

BEGIN_MESSAGE_MAP(CImpFractalView, CView)
    ON_WM_SIZE()
END_MESSAGE_MAP()

// Конструктор/деструктор класса CImpFractalView

CImpFractalView::CImpFractalView()
{
    // TODO: добавьте сюда код конструктора

    m_DrawThread = 0;
}

CImpFractalView::~CImpFractalView()
{
    delete m_DrawThread;
}

BOOL CImpFractalView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Измените класс или стили окна здесь,
    // изменяя или добавляя поля структуры cs

    return CView::PreCreateWindow(cs);
}

// Прорисовка класса CImpFractalView

UINT DrawFractal (LPVOID PHWndView)
{
    float CI;
    CClientDC ClientDC (CWnd::FromHandle (*(HWND *)PHWndView));
    int Col;
    static DWORD ColorTable [6] =
        {0x0000ff,    // красный
        0x00ff00,    // зеленый
        0xff0000,    // синий
        0x00ffff,    // желтый
        0xffff00,    // бирюзовый
        0xff00ff};   // сиреневый
    int ColorVal;
    float CR = (float)CRMIN;
    float DCI = (float)((CIMAX - CIMIN) / (RowMax-1));

```

```

float DCR = (float)((CRMAX - CRMIN) / (ColMax-1));
float I;
float ISqr;
float R;
int Row;
float RSqr;

for (Col = 0; Col < ColMax; ++Col)
{
    if (StopDraw) break;

    CI = (float)CIMAX;

    for (Row = 0; Row < RowMax; ++Row)
    {
        R = (float)0.0;
        I = (float)0.0;
        RSqr = (float)0.0;
        ISqr = (float)0.0;
        ColorVal = 0;
        while (ColorVal < NMAX && RSqr + ISqr < 4)
        {
            ++ColorVal;
            RSqr = R * R;
            ISqr = I * I;
            I *= R;
            I += I + CI;
            R = RSqr - ISqr + CR;
        }
        ClientDC.SetPixelV (Col, Row, ColorTable [ColorVal % 6]);
        CI -= DCI;
    }

    CR += DCR;
}

return (0);
}

void CImpFractalView::OnDraw(CDC* pDC)
{
    CImpFractalDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    // TODO: добавьте сюда код прорисовки собственных данных

    if (m_DrawThread)
    {
        StopDraw = TRUE;
        m_DrawThread->ResumeThread ();
        ::WaitForSingleObject
        (m_DrawThread->m_hThread, INFINITE);
        // получения управления потоком рисования
        // ждать столько, сколько нужно
        delete m_DrawThread;
    }
}

```

```

    m_DrawThread = AfxBeginThread
        (DrawFractal,
         &m_hWnd,
         THREAD_PRIORITY_BELOW_NORMAL,
         0,
         CREATE_SUSPENDED);
    m_DrawThread->m_bAutoDelete = FALSE;
    StopDraw = FALSE;
    m_DrawThread->ResumeThread ();
}

// CImpFractalView diagnostics

#ifdef _DEBUG
void CImpFractalView::AssertValid() const
{
    CView::AssertValid();
}

void CImpFractalView::Dump(CDumpContext& dc) const
{
    CView::Dump(dc);
}

CImpFractalDoc* CImpFractalView::GetDocument() const
// не отладочная версия встроена
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CImpFractalDoc)));
    return (CImpFractalDoc*)m_pDocument;
}
#endif // _DEBUG

// Обработчики сообщений класса CImpFractalView

void CImpFractalView::OnSize(UINT nType, int cx, int cy)
{
    CView::OnSize(nType, cx, cy);

    // TODO: Добавьте сюда собственный код обработчика

    if (cx <= 1 || cy <= 1) return;

    ColMax = cx;
    RowMax = cy;
}

```

---

## Листинг 22.7.

```

// MainFrm.h : интерфейс класса CMainFrame
//

#pragma once
class CMainFrame : public CFrameWnd
{

```

```

protected: // используется только для сериализации
    CMainFrame();
    DECLARE_DYNCREATE(CMainFrame)

// Атрибуты
public:

// Операции
public:

// Переопределения
public:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);

// Реализация
public:
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

// Сгенерированные функции карты сообщений
protected:
    DECLARE_MESSAGE_MAP()
};

```

---

#### Листинг 22.8.

```

// MainFrm.cpp : реализация класса CMainFrame
//

#include "stdafx.h"
#include "ImpFractal.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
END_MESSAGE_MAP()

// Конструктор/деструктор класса CMainFrame

CMainFrame::CMainFrame()
{
    // TODO: добавьте сюда код инициализации
}

```

```

CMainFrame::~CMainFrame()
{
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    if( !CFrameWnd::PreCreateWindow(cs) )
        return FALSE;
    // TODO: Измените класс или стили окна здесь,
    // изменяя и добавляя поля структуры cs

    return TRUE;
}

// Диагностика CMainFrame

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
    CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
    CFrameWnd::Dump(dc);
}

#endif // _DEBUG

// Обработчики сообщений класса CMainFrame

```

## Резюме

---

Рассмотрен запуск и управление вторичными потоками в MFC-программе и способы их синхронизации.

- *Первичный и вторичный поток.* Поток – это выполняемая последовательность команд программы. Все программы начинаются с выполнения первичного *потока*. Затем программа может запустить один или более *вторичных потоков*, каждый из которых выполняется независимо.
- *Запуск потока.* Запуск нового потока требует относительно небольших затрат. Все потоки одной программы могут совместно использовать ее ресурсы (глобальные или статические переменные, распределение памяти, окна, открытые файлы и прочее). Чтобы запустить новый поток в MFC-программе, вызовите глобальную функцию `AfxBeginThread` MFC-класса и присвойте ее первым двум параметрам адрес *функции потока* (функции, выполняемой потоком) и значение параметра, передаваемого в эту функцию. Остальные параметры функции `AfxBeginThread`, которые имеют стандартные значения, обычно принимаются без изменений. Функция `AfxBeginThread` возвращает управление, после чего вызывающий и новый поток выполняются одновременно и асинхронно.
- *Приостановка потока.* Поток можно приостановить на заданное число миллисекунд после вызова функции `::Sleep` Win32 API. Если любая из функций синхронизации Win32 переводит поток в режим ожидания, то выполнение приостанавливается, и во время ожидания поток не занимает ресурсы процессора.



- *Завершение потока.* Вторичный поток завершается при возврате из функции потока или при вызове глобальной функции `AfxBeginThread`. При вызове функции `AfxBeginThread` возвращается указатель на объект класса `CWinThread`. Эти объекты используются для приостановки или завершения потока, изменения его приоритета и управления потоком.
- *Синхронизация потоков.* Иногда работу отдельных потоков необходимо синхронизировать, например, при работе с таким общим ресурсом, как глобальная переменная. Среда Win32 API содержит ряд объектов, применяемых для синхронизации потоков – мьютексы, критические секции, семафоры и события. *Мьютекс* или *критическая секция* запрещают одновременный доступ к общему ресурсу более чем одного потока. *Семафор* используется для предотвращения одновременного доступа к общему ресурсу потоков, количество которых превышает заданное. *Событие* предназначено для того, чтобы заставить один поток передавать сигналы одному или более дополнительным потокам. Действие потока синхронизируется вызовом функции `:WaitForSingleObject`, `:WaitForMultipleObject` или `:MsgWaitForMultipleObject` Win32 API, что позволяет ожидать окончания потока, процесса или какого-либо другого события.

# Глава 23

## Процессы

---

- Запуск процесса
- Дескрипторы общих объектов
- Каналы и почтовые ящики
- Файлы памяти и совместный доступ
- Буфер обмена

Когда исполняемый файл запускается, система создает новый *процесс*, который можно рассматривать как исполняемый экземпляр *программы*. Термины *программа* и *процесс* похожи по смыслу, но есть и отличие: *программа* обозначает исполняемый файл или его исходный текст, а *процесс* – то, что происходит при выполнении программы (если подходить с такой позиции, то программа может порождать несколько процессов). В предыдущей главе вы научились создавать множество *потоков*, выполняемых внутри одного процесса, и управлять ими. В этой главе научитесь создавать процессы, управлять ими и связывать их. Узнаете, как осуществляется запуск нового процесса из программы, а также как управлять процессом и ожидать его завершения. Увидите, как используются объекты синхронизации для координации работы потоков внутри отдельных процессов и дескрипторы разных процессов Windows при их совместном применении. Узнаете, как используются два механизма обмена данными между процессами: каналы (*pipes*) и общая память. Технические приемы синхронизации процессов и обмена данными по каналам и через общую память особенно полезны для использования в пакете программ, разработанных для совместной работы.

В заключительном разделе главы описано, как можно обмениваться данными через буфер системы Windows, позволяющий даже не родственным приложениям выполнять простой перенос данных в стандартных форматах. В гл. 24 вы научитесь использовать более сложные механизмы, предоставляемые протоколом OLE, позволяющим практически без ограничений на форматы переносить данные между различными программами.

### Запуск процесса

---

В некоторых случаях вместо создания уже знакомых вам по предыдущей главе отдельных потоков внутри процесса может потребоваться разделение приложений на отдельные *процессы*. Код для каждого из них содержится в отдельном исполняемом файле. Запуск отдельного процесса медленнее и расходует больше системных ресурсов, чем запуск потока, а обмен информацией между отдельными процессами более сложен, чем между потоками. Однако, поскольку каждый процесс выполняется в своем собственном адресном пространстве, то для отдельных процессов менее вероятно вмешательство одного в работу другого, чего не скажешь о работе потоков.

Использование отдельных процессов позволяет разрабатывать наборы родственных программ или разделять большое приложение на отдельные модули. Например, модуль обработки платежной ведомости для пакета бухгалтерских программ может быть разработан и размещен в отдельном исполняемом файле, и выполняться, как отдельный процесс. Когда приложение состоит из отдельных процессов, один из них (обычно, главный модуль программы) может запускать один или несколько дополнительных. Следовательно, нужно запустить только главный исполняемый файл программы, а управлять запуском исполняемого файла для каждого дополнительного процесса не нужно. Например,

если в бухгалтерском приложении с несколькими процессами выбрать команду расчетов с клиентами, то программа может запустить исполняемый файл модуля выполнения расчетов с клиентами. Для запуска такого файла (и запуска нового процесса) в программе можно вызвать функцию Win32 API ::CreateProcess. Ее синтаксис выглядит так:

```

BOOL CreateProcess
(LPCTSTR lpszImageName,           // путь доступа к
                                // исполняемому файлу;
LPTSTR lpszCommandLine,          // командная строка;
LPSECURITY_ATTRIBUTES lpsaProcess, // атрибуты защиты процесса;
LPSECURITY_ATTRIBUTES lpsaThread,  // атрибуты защиты потока;
BOOL finheritHandles,             // наследует ли новый процесс
                                // дескрипторы?
DWORD fdwCreate,                  // флаги создания процесса;
LPVOID lpvEnvironment,           // блок окружения нового
                                // процесса;
LPCTSTR lpszCurDir,              // текущий каталог
                                // процесса;
LPSTARTUPINFO lpsiStartInfo,      // описывает функции окна
                                // процесса;
LPPROCESS_INFORMATION lppiProcInfo); // получает информацию о
                                // новом процессе

```

Функция ::CreateProcess вызывается *родительским* процессом, а запускаемый с ее помощью процесс называется *дочерним*. Функция ::CreateProcess используется для выполнения программ любого типа, поддерживаемых операционной системой, например, 32-битовой программы с графическим интерфейсом, консольной программы Windows, 16-битовой программы Windows 3.1 или 16-битовой программы MS-DOS. Например, следующий вызов запускает как отдельный процесс программу Payroll.exe.

```

STARTUPINFO StartupInfo;
memset (&StartupInfo, 0,           // использует стандартные
      sizeof (STARTUPINFO));        // компоненты окна;
StartupInfo.cb = sizeof (STARTUPINFO); // это поле должно
                                // быть заполнено

PROCESS_INFORMATION ProcessInfo;

::CreateProcess
("PAYROLL.EXE", // запускает исполняемый файл "PAYROLL.EXE";
NULL,           // командная строка не задана;
NULL,           // стандартные атрибуты защиты процесса
NULL,           // стандартные атрибуты защиты потока;
FALSE,          // процесс не наследует дескрипторы;
0,              // флажки не создаются (используются стандартные);
NULL,           // использует среду вызывающего процесса;
NULL,           // тот же текущий каталог, что и у
                // вызывающего процесса;
&StartupInfo,   // описывает компоненты окна процесса;
&ProcessInfo); // получает информацию от процесса

```

Рассмотренные в приведенном примере параметры можно истолковать следующим образом:

- *Первый параметр* содержит только простое имя исполняемого файла, значит, этот файл должен быть расположен в текущем каталоге.

- Второму параметру присвоено значение NULL, поэтому командная строка, передаваемая в дочерний процесс, состоит только из имени исполняемого файла PAYROLL.EXE. Второй параметр можно использовать для передачи аргументов командной строки в дочерний процесс. Последний получает доступ к полной командной строке с помощью вызова функции Win32 API `::GetCommandLine` либо с помощью параметра `lpCmdLine`, передаваемого в функцию `WinMain`.
- Для параметров с *третьего по девятый*, в нашем примере, передаются стандартные значения.
- Десятый (последний) параметр задает адрес неинициализированной структуры `ProcessInfo` типа `PROCESS_INFORMATION`. Функция `::CreateProcess` присваивает этой структуре дескриптор, а также идентификаторы дочернего процесса и его первичного потока. После того как родительский процесс вызовет функцию `::CreateProcess` для запуска дочернего процесса, он может вызвать функцию Win32 API `::WaitForInputIdle` для ожидания завершения инициализации этого процесса (например, создания окна программы) и не задерживать сообщения. Например, если родительскому процессу нужно передать сообщение в окно дочернего процесса, то чтобы подождать завершения создания окна дочерним процессом, можно вызвать функцию `WaitForInputIdle`.

Запустив процесс, функция `::CreateProcess` возвращает управление. С этого момента дочерний и родительский процессы выполняются одновременно. Как и любой процесс, дочерний завершается при выходе из вызывающей его функции (`WinMain` для программы с графическим интерфейсом) или при вызове функции Win32 API `::ExitProcess`. Значение, которое дочерний процесс возвращает или присваивает функции `::ExitProcess`, называется *кодом возврата*. Родительский или любой другой процесс, имеющий корректный дескриптор для дочернего процесса (см. ниже), может получить этот код, вызвав функцию Win32 API `::GetExitCodeProcess`, как в следующем примере. Здесь `ProcessInfo` является структурой типа `PROCESS_INFORMATION`, которая заполняется с помощью вызова функции `::CreateProcess`. Заметим: если процесс все еще выполняется, то функция `::GetExitCodeProcess` предоставляет в качестве кода возврата специальное значение `STILL_ACTIVE`.

```
DWORD ExitCode;

::GetExitCodeProcess
    (ProcessInfo.hProcess, // дескриптор процесса, предоставленный
                                // функцией CreateProcess;
    &ExitCode);             // адрес переменной DWORD,
                                // принимающей код возврата
if (ExitCode == STILL_ACTIVE)
    // процесс продолжает выполняться ...
else
    // процесс завершен и ExitCode содержит его код возврата
```

Чтобы *подождать* завершения дочернего процесса (см. гл. 22), программа передает в функцию Win32 API `::WaitForSingleObject` его дескриптор. Программа может передавать дескриптор дочернего процесса другим функциям Win32 API, предназначенным для управления им, например, функции `::SetPriorityClass` для изменения приоритета процесса или функции `::TerminateProcess` для его немедленного прекращения. Дескриптор процесса остается корректным даже после окончания процесса. Когда программа завершится с использованием дескриптора, она может закрыть его, передавая дескриптор в функцию Win32 API `::CloseHandle`. Если программа не закрыла дескриптор, то он закроется автоматически при завершении программы. Когда *все* дескрипторы данного процесса будут закрыты, Windows освобождает память от связанной с процессом информации.

Запуск дочернего процесса не сопровождается автоматическим установлением отношений между родительским и дочерним процессами. Например, при завершении родительского процесса запущенные им дочерние процессы в системе Windows *не* завершаются автоматически. Кроме того, *любой* процесс

(не только родительский) может вызвать функцию Win32 API (например, `::WaitForInputIdle`, `::GetExitCodeProcess`, `::WaitForSingleObject`, `::SetPriorityClass` или `::TerminateProcess`) для управления или ожидания окончания дочернего процесса при условии, что он имеет дескриптор этого процесса. Однако каждый процесс должен получить свой собственный дескриптор для дочернего, используя один из описанных ниже методов совместного использования дескрипторов.

## Дескрипторы общих объектов

Четыре типа объектов синхронизации Win32, которые используются для синхронизации действия отдельных потоков внутри одного процесса, рассматривались в гл. 22. Три из них – мьютексы, семафоры и события – можно применять для синхронизации действий потоков между *разными процессами*. Вспомните: при использовании объекта синхронизации нужно передавать его дескриптор соответствующей функции Win32 API. Но, как правило, дескриптор Windows, полученный процессом (например, дескриптор мьютекса, полученный при вызове функции `::CreateMutex`), не может использоваться другим процессом. Вместо этого каждый процесс должен получить свой собственный дескриптор этого объекта. Исключением является дескриптор, *унаследованный* дочерним процессом (см. ниже).

Рассмотрим пример, в котором (и в следующих за ним) предполагается, что `hMutex` – глобальная переменная типа `HANDLE`.

```
// в первом процессе:
hMutex = ::CreateMutex
(NULL, // присваивает стандартные атрибуты
    // защиты; не наследует дескриптор;
FALSE, // мьютекс изначально свободен;
"Acme Accounting Mutex"); // имя мьютекса
```

Второй процесс может получить дескриптор *того же* мьютекса, выполняя идентичный вызов функции и задавая для мьютекса то же самое имя.

```
// во втором процессе:
hMutex = ::CreateMutex
(NULL, // присваивает стандартные атрибуты
    // защиты; не наследует дескриптор;
FALSE, // мьютекс изначально свободен;
"Acme Accounting Mutex"); // имя мьютекса
```

- Если мьютекс при вызове функции `::CreateMutex` *не существует*, то эта функция создает объект и возвращает дескриптор, который используется текущим процессом.
- Если мьютекс уже *существует*, то функция `::CreateMutex` просто возвращает новый дескриптор того же объекта, который можно использовать в текущем процессе.

Убедитесь, что мьютексу присвоено имя, которое не используется в другой программе, и что присвоенное каждой функции API оно точно совпадает с требуемым, включая регистр каждой буквы. Объекты синхронизации (а также объекты файлов памяти, рассматриваемые далее) совместно используют одинаковые имена. Таким образом, если имя, передаваемое в функции `::Create...` или `::Open...` совпадает с именем объекта синхронизации (или файла памяти) другого типа, возникает ошибка.

Кроме того, несколько процессов могут совместно использовать один семафор или событие, присваивая им имя при вызове функции `::CreateSemaphore` или функции `::CreateEvent`. Как только один или несколько процессов получают дескрипторы общего объекта синхронизации, этот объект может использоваться отдельными потоками внутри одного процесса (см. гл. 22). Например,

мьютекс можно использовать, чтобы запретить одновременный доступ к общему блоку памяти (см. ниже) более чем одному процессу.

Для совместного использования дескрипторов именованных объектов синхронизации можно вызвать функцию Win32 `::Open...` Например, если один процесс уже создал мьютекс посредством вызова функции `::CreateMutex` (см. пример выше в этом разделе), то другой процесс, вызывая функцию `::OpenMutex...`, мог бы получить дескриптор того же мьютекса:

```
// во втором процессе:
HMutex = ::OpenMutex
(MUTEX_ALL_ACCESS, // флаг доступа: разрешен
                    // максимальный доступ;
FALSE,             // флаг наследования: дескриптор
                    // мьютекса не наследуется;
"Acme Accounting Mutex"); //имя мьютекса
```

Рассмотрим ситуацию, когда другой процесс еще не создал мьютекс с таким же именем. В этом случае функция `::OpenMutex` завершается неудачно и возвращает NULL. Следовательно, при использовании этого метода доступа к мьютексу необходимо убедиться, что процесс, вызывающий функцию `::CreateMutex`, сделает это раньше, чем другой процесс вызовет функцию `::OpenMutex`. Если оба процесса получают дескриптор, вызывая функцию `::OpenMutex`, то они могут делать это в любом порядке. Таким же образом процесс, вызывая функцию `::OpenSemaphore` или `::OpenEvent`, может получить дескриптор существующего семафора или события.

Для получения дескриптора другого процесса (даже если он не создает его), процесс также может вызвать функцию Win32 `::OpenProcess`. В этом случае в функцию `::OpenProcess` нужно передать *идентификатор* процесса, а не имя (в отличие от объектов синхронизации процессам нельзя задавать имя):

```
HANDLE HProcess;
DWORD IDProcess;

// получить идентификатор процесса и присвоить его
// переменной IDProcess ...

HProcess = ::OpenProcess
(PROCESS_ALL_ACCESS, // флаг доступа: позволяет
                    // максимальный доступ;
FALSE,             // флаг наследования: процесс
                    // не наследует дескриптор;
IDProcess);        // идентификатор процесса
```

Хотя программа может заранее знать *имя* объекта, *идентификатор* объекта она должна получить во время выполнения. В приведенном примере программа должна получить идентификатор процесса, которым нужно управлять. Этот дескриптор можно получить из родительского процесса или из текущего процесса, используя такие способы совместного использования данных, как каналы или общие блоки памяти (см. ниже):

- родительский процесс может получить дескриптор из поля `dwProcessId` структуры `LPPROCESS_INFORMATION`, заполняемого функцией `::CreateProcess`;
- текущий процесс может получить свой идентификатор, вызвав функцию Win32 API `::GetCurrentProcessId`.

Как только программа получит дескриптор процесса, его можно использовать для вызова любой функции Win32 API, для ожидания завершения процесса или управления им. Объект Windows не уничтожается, пока не закрыты *все* дескрипторы. Дескриптор можно закрыть явным вызовом функции Win32 API `::CloseHandle` или путем завершения процесса, который этим дескриптором владеет.

Кроме рассмотренных выше существует два других способа совместного использования дескрипторов Windows различными процессами:

- *Первый способ* основан на том, что дочерний процесс может *наследовать* один или несколько дескрипторов из родительского.
- 1. Чтобы дочерний процесс получил дескриптор, родительский процесс при создании или открытии объекта должен указать, что этот дескриптор является *наследуемым* (в гл. 22 и 23 дескрипторы сделаны *ненаследуемыми*). Например, если при создании мьютекса полю `bInheritHandle` структуры `SECURITY_ATTRIBUTES` присвоить значение `TRUE`, то дескриптор станет наследуемым:

```
SECURITY_ATTRIBUTES Security =
    (sizeof (SECURITY_ATTRIBUTES),
     NULL,           // задает стандартную защиту;
     TRUE);          // ДЕЛАЕТ ДЕСКРИПТОР НАСЛЕДУЕМОМ

HMutex = ::CreateMutex
    (&Security,      // задает защиту и наследуемость дескриптора;
     FALSE,          // мьютекс изначально свободен;
     NULL);          // мьютекс не имеет имени
```

- 2. Родительский процесс, создавая дочерний, должен присвоить значение `TRUE` пятому параметру – `::CreateProcess (fInheritHandles)`, позволяющему дочернему процессу наследовать любой *наследуемый* дескриптор, принадлежащий родительскому процессу.
- 3. Наконец, родительский процесс предоставляет дочернему процессу значение фактически существующего дескриптора с помощью подходящего способа организации связи между процессами, например, канала или общего блока памяти.
- *Второй способ* основан на том, что процесс, имеющий дескриптор отдельного объекта, может вызвать функцию Win32 API `::DuplicateHandle` для генерирования нового дескриптора того же объекта (*дубликата*). Этот дескриптор может использоваться *любым* указанным процессом, в том числе текущим процессом.

Два описанных способа использования дескрипторов не так удобны, как вызовы функций `::Create...` и `::Open...`, потому что требуют обмена информацией (дескрипторами или идентификаторами) между процессами во время исполнения. Однако эти методы применимы для совместного использования широкого набора дескрипторов различных типов, в частности – объектов синхронизации, потоков, процессов, каналов, объектов файлов памяти и любых других, созданных функцией Win32 API `::CreateFile`. При этом следует помнить, что в системе Windows разным процессам *нельзя* совместно использовать дескрипторы ряда разновидностей объектов, которые всегда остаются собственностью процесса. К ним относятся:

- выделенная память;
- графические объекты;
- окна.

## Каналы и почтовые ящики

Для передачи данных от одного процесса другому в Windows используется специальный механизм, называемый *каналом* (*pipe*). Средства Win32 предусматривают возможность использования каналов как *именованных*, так и *анонимных*. Анонимный канал чаще всего используется для обмена данными между родительским и дочерним или между дочерними процессами.

1. Родительский процесс создает канал, вызывая функцию Win32 API `::CreatePipe`, предоставляющую два дескриптора: один – для записи в канал и другой – для чтения из канала. При вызове функции `::CreatePipe` родительский процесс должен сделать дескрипторы канала наследуемыми. Перед созданием дочернего процесса родительский процесс может сделать доступными для дочернего процесса дескрипторы чтения или записи, вызвав функцию Win32 API `::SetStdHandle`. Эта функция модифицирует стандартные дескрипторы ввода, вывода и стандартный дескриптор ошибок для родительского и дочернего процессов. Если родительский процесс хочет передать данные дочернему, то он должен использовать функцию `::SetStdHandle`, чтобы присвоить дескриптор чтения канала стандартному дескриптору ввода. Однако, если родительский процесс хочет получить данные из дочернего процесса, он должен использовать функцию `::SetStdHandle`, чтобы присвоить дескриптор записи канала стандартному дескриптору вывода.
2. Родительский процесс вызывает функцию `::CreateProcess` для создания дочернего процесса. При этом он должен разрешить наследование дескриптора.
3. Если родительский объект посылает данные дочернему объекту, то для записи данных в канал используется функция Win32 API `::WriteFile`. Затем родительский процесс вызывает функцию `::ReadFile` для чтения данных из канала. Если родительский канал получает данные из дочернего канала, то последний вызывает функцию `::GetStdHandle` для получения дескриптора записи канала, а для записи данных в канал – функцию `::WriteFile`. Затем родительский канал вызывает функцию `::ReadFile`, чтобы получить данные из канала, передавая ей дескриптор чтения канала.

В среде Win32 предусмотрен и другой механизм для межпроцессорной связи, называемый *почтовым ящиком (mailslot)*.

- Процесс (называемый *клиентом*) может послать сообщение в отдельный почтовый ящик, распознаваемый по имени.
- Сообщение будет получено другим процессом (называемым *сервером*), который создает почтовый ящик с этим именем.

Почтовые ящики удобны, так как позволяют процессу передавать сообщения целой группе других процессов. Однако передающий процесс *не получает подтверждения* о получении сообщения. Канал же позволяет передать данные только одному процессу и проверить, получено ли сообщение. Информацию о почтовых ящиках смотрите в справочной системе.

## Файлы памяти и совместный доступ

Windows не дает, как правило, процессу возможности читать или изменять данные, т. е. переменные, объекты или блоки памяти, принадлежащие другому процессу. Однако при использовании входящих в средства Win32 *файлов памяти* можно организовать доступ к общему блоку памяти для двух и более процессов. В общем, файлы памяти предоставляют одному или более процессам доступ к файлу для чтения или записи копии (или *отображения*) этого файла. Однако когда файл памяти используется просто для организации совместного использования блоков памяти различными процессами, не нужно открывать или создавать файл на диске. Как и для любого блока памяти, операционная система будет подкачивать память по мере необходимости в собственный файл размещения страниц памяти. Для выделения совместно используемого блока памяти и организации доступа к нему для двух процессов с применением файла памяти выполняются следующие действия.

1. Каждый процесс вызывает функцию Win32 API `::CreateFileMapping`, например, так:

```
// код внутри КАЖДОГО из процессов:  
HANDLE hFileMapping;
```



```

HFileMapping = ::CreateFileMapping
((HANDLE)0xFFFFFFFF, // новый файл отсутствует (используется
                      // системный файл размещения);
(LPSECURITY_ATTRIBUTES) NULL, // стандартная защита; дескриптор
                              // не наследуемый;
PAGE_READWRITE, // разрешает доступ для чтения/записи;
0, // 32-битовый (старшие разряды),
   // максимальный размер : 0;
1024, // 32-битовый (младшие разряды),
      // максимальный размер : 1K;
"MyFileMappingObject"); // имя объекта файла памяти

```

Когда в одном из процессов выполняется первый вызов функции `::CreateFileMapping`, то создается *объект файла памяти* и возвращается его дескриптор. При втором вызове функции `::CreateFileMapping` (в другом процессе) возвращается новый дескриптор существующего объекта файла памяти, действительный внутри текущего процесса. Приведенные в примере вызовы ограничивают размер создаваемых блоков памяти 1 Кбайтом. В результате этих вызовов каждый из процессов получает дескриптор своего объекта файла памяти.

2. Каждый процесс вызывает функцию Win32 API `::MapViewOfFile`, чтобы выделить блок памяти, называемый *представлением файла*, передавая дескриптор объекта файла памяти и указывая *то же самое* имя ("MyFileMappingObject") и размер (1024 байта), которые задавались в вызове функции `::CreateFileMapping`. Это может выглядеть, например, так:

```

// код внутри КАЖДОГО из процессов:
char *PtrSharedMemory;

PtrSharedMemory = (char *)::MapViewOfFile
(HFileMapping, // дескриптор объекта файла памяти
FILE_MAP_ALL_ACCESS, // разрешает доступ для чтения/записи
0, // 32-битовый (старшие разряды) со смещением
   // совместно используемого блока: 0;
0, // 32-битовый (младшие разряды) со смещением
   // совместно используемого блока: 0;
1024); // 1 К в совместно используемом блоке

```

Когда в одном из процессов выполняется первый вызов функции `::MapViewOfFile`, то выделяется блок памяти и возвращается указатель на него. При втором вызове (в другом процессе) – возвращается указатель на *тот же* блок памяти. Теперь каждый процесс использует указатель для доступа к блоку совместно используемой памяти так же, как он обращается к любой выделенной памяти. Поскольку оба процесса имеют доступ к *одним* данным, им могут потребоваться объекты синхронизации Win32 для синхронизации доступа.

3. Когда процессы завершают совместное использование блока памяти, каждый из них вызывает функцию Win32 API `::UnmapViewOfFile`, чтобы отменить представление файла, а затем (чтобы закрыть дескриптор объекта файла памяти) – функцию Win32 API `::CloseFile`.

## Буфер обмена

Рассмотрим использование буфера обмена для передачи данных в программу и обмена данными с другими программами. В первом разделе описываются команды, доступные в программе, которая использует буфер обмена. Далее объясняется, как он используется для передачи простого текста, графических изображений в растровом формате и представленных в пользовательском формате данных.

## Команды управления

В меню Edit программ, оперирующих буфером обмена, обычно содержатся команды Cut, Copy и Paste. Если для генерации программы используется мастер Application Wizard, то он добавит команды Cut, Copy и Paste в исходное меню Edit. В табл. 23.1 перечислены свойства команд, определенных мастером Application Wizard и появляющихся в редакторе меню Visual C++.

Табл. 23.1. Свойства команд Cut, Copy и Paste меню Edit, сгенерированных мастером Application Wizard

Идентификатор (ID)	Надпись (Caption)	Интерактивная справка
ID_EDIT_CUT	Cu&tCtrl+X	Cut the selection and put it on the Clipboard\nCut (Удалить в буфер)
ID_EDIT_COPY	&CopytCtrl+C	Copy the selection and put it on the Clipboard\nCopy (Копировать в буфер)
ID_EDIT_PASTE	&Paste\ tCtrl+V	Insert Clipboard contents\nPaste (Вставить из буфера)

По умолчанию, мастер Application Wizard также задает два набора горячих клавиш для команд Cut, Copy и Paste. Первый – соответствует ранее принятым соглашениям о сочетаниях клавиш, а второй – новому соглашению. Можно использовать и старое, и новое сочетание клавиш для активизации этих команд. Заметим: заголовки меню отображают сочетания клавиш, соответствующие новым соглашениям (табл. 23.2).

Табл. 23.2. Клавиатурные сочетания для команд Cut, Copy и Paste, сгенерированных мастером Application Wizard

Команда	Имя команды меню и акселератора	Сочетание клавиш по старому соглашению	Сочетание клавиш по новому соглашению
Cut	ID_EDIT_CUT	Shift+Del	Ctrl+X
Copy	ID_EDIT_COPY	Ctrl+Ins	Ctrl+C
Paste	ID_EDIT_PASTE	Shift+Ins	Ctrl+V

Дескрипторы для команд Cut, Copy и Paste мастером Application Wizard не предоставлены. Следовательно, они изначально недоступны. Как правило, обработчики сообщений для этих команд можно определить, используя окно Properties. Их обычно добавляют в класс представления, отвечающий за интерфейс команд, позволяющих редактировать документ, отображаемый в окне представления. Кроме того, для каждой из команд Cut, Copy и Paste нужно предоставить обработчики COMMAND и UPDATE\_COMMAND\_UI. Обработчик UPDATE\_COMMAND\_UI получает управление, когда меню Edit открывается, после чего обработчик, основываясь на текущем состоянии буфера обмена или окна представления, делает команду либо доступной, либо недоступной. Обработчик COMMAND получает управление, когда выбирается команда, выполняющая операцию с буфером обмена. Если в диалоговом окне Application Wizard установить опцию создания панели инструментов, то мастер добавит панель инструментов с кнопками команд Cut, Copy и Paste. Как и соответствующие команды меню, эти кнопки будут недоступны до тех пор, пока не будут определены обработчики сообщений. Поскольку кнопкам присваиваются *те же* идентификаторы, что и соответствующим командам меню, нужно определить один набор обработчиков сообщений для обработки команд меню Edit и соответствующих кнопок панели инструментов. В табл. 23.3 приведены стандартные имена обработчиков, генерируемых для этих команд в Visual Studio.

Определенные для команд Cut, Copy и Paste обработчики сообщения COMMAND передают выбранные данные (текст или графику) в буфер обмена и из него. Кроме того, обработчик сообщения COMMAND команды Cut удаляет выбранные данные из документа. В меню Edit для удаления выбранных данных без использования буфера обмена можно включить команду Delete или Clear, которые, хотя и не используют буфер, в меню Edit обычно группируются с командами работы с буфером. Обработ-

чики сообщения `UPDATE_COMMAND_UI` команд `Cut` и `Copy` доступны только в том случае, если в окне представления выбраны данные документа.

Табл. 23.3. Обработчики сообщений, предлагаемые по умолчанию для команд `Cut`, `Copy` и `Paste` меню `Edit`

Команда	Идентификатор	Тип сообщения	Обработчик сообщения
Cut	<code>ID_EDIT_CUT</code>	<code>COMMAND</code>	<code>OnEditCut</code>
	<code>ID_EDIT_CUT</code>	<code>UPDATE_COMMAND_UI</code>	<code>OnUpdateEditCut</code>
Copy	<code>ID_EDIT_COPY</code>	<code>COMMAND</code>	<code>OnEditCopy</code>
	<code>ID_EDIT_COPY</code>	<code>UPDATE_COMMAND_UI</code>	<code>OnUpdateEditCopy</code>
Paste	<code>ID_EDIT_PASTE</code>	<code>COMMAND</code>	<code>OnEditPaste</code>
	<code>ID_EDIT_PASTE</code>	<code>UPDATE_COMMAND_UI</code>	<code>OnUpdateEditPaste</code>

Данные из буфера можно поместить в любое место в этом документе, в другой документ этой программы или в документ другой программы. Программой-получателем может быть программа Windows с графическим интерфейсом или консольная программа, 16-битовая программа Windows 3.1 или MS DOS. В консольную программу или программу MS DOS можно вставить только текстовые данные. Обработчик сообщения `COMMAND` команды `Paste` вставляет содержимое буфера обмена в текущий документ. Данные могут быть получены из текущего документа, из другого документа внутри этой программы или из другой программы (с графическим интерфейсом или консольной, 16-битовой программы Windows 3.1 или MS DOS). Текстовые или графические данные обычно вставляются в той позиции, где курсор отмечает точку вставки текста в окне представления (например, в приложении текстового процессора).

Для некоторых программ (например, программ рисования) полезной может быть возможность вставить *графические* данные в произвольное место окна представления и выделить их для перемещения в желаемую позицию.

Если буфер содержит данные в соответствующем формате, то обработчик сообщения `UPDATE_COMMAND_UI` для команды `Paste` делает ее доступной. Алгоритм функционирования обработчиков `COMMAND` и `UPDATE_COMMAND_UI` зависит от формата переносимых данных. Техника тестирования отдельных форматов рассмотрена ниже.

## Обмен текстом через буфер

Буфера обмена можно использовать для пересылки простого текста, т. е. текста, состоящего из простого потока печатных символов ANSI, не содержащего внедренных форматирующих кодов, например, текста, отображаемого редактором Windows Notepad. Чтобы переслать текст, содержащий собственные коды форматирования (например, форматированный текст, отображаемый в текстовом процессоре), применяются способы, описанные в параграфе “Данные зарегистрированных форматов в буфере обмена”. Вы узнаете, как переслать текст в буфер, а затем получить текст из буфера и вставить его в документ.

Разрабатывая программу, в которой класс представления программы наследуется от MFC-класса `CEditView` (включающего текстовый редактор, рассмотренный в гл. 10), нет необходимости писать строки для поддержки команд `Cut`, `Copy` и `Paste`. Нужно только убедиться, что меню программы `Edit` содержит эти команды, и они имеют идентификаторы, приведенные в табл. 23.1. Класс `CEditView` предоставляет для этих команд полный набор обработчиков сообщений.

## Передача текста в буфер обмена

В буфер обмена текст можно передать командами `Cut` и `Copy`. Для них обработчик сообщений `UPDATE_COMMAND_UI` доступен только в том случае, если выбран блок текста, как показано в примере

ниже. Предполагается, что `m_IsSelection` – это переменная класса представления типа `BOOL`, которой присваивается значение `TRUE` при выборе блока текста.

```
void CProgView::OnUpdateEditCut (CCmdUI* pCmdUI)
{
    // TODO: Здесь добавьте собственный код обработчика
    pCmdUI->Enable (m_IsSelection);
}

void CProgView::OnUpdateEditCopy (CCmdUI* pCmdUI)
{
    // TODO: Здесь добавьте собственный код обработчика
    pCmdUI->Enable (m_IsSelection);
}
```

Выделенный блок текста передается в буфер обмена обработчиками сообщений `COMMAND` для команд `Cut` и `Copy`. При этом выполняется вызов функции `Win32 API ::GlobalAlloc` для выделения блока памяти, достаточного для размещения текста, копируемого в буфер. Затем вызывается функция `Win32 API ::GlobalLock` для блокирования блока памяти и получения его указателя. После этого выполняется собственно копирование выделенного текста в выделенный блок памяти. Затем вызывается функция `Win32 API ::GlobalUnlock` для освобождения выделенной памяти и функция `OpenClipboard` класса `CWnd` для открытия буфера обмена. Далее необходимо вызвать функцию `Win32 API ::EmptyClipboard` для удаления текущего содержимого буфера. Потом вызывается функция `Win32 API ::SetClipboardData` для задания дескриптора выделенного блока памяти буфера и функция `Win32 API ::CloseClipboard` для закрытия буфера. В обработчике сообщения `COMMAND` команды `Cut` выполняется удаление отмеченных данных из документа. Текст, передаваемый в буфер обмена с использованием описанной процедуры, должен соответствовать стандартным текстовым форматам буфера (в противном случае этот текст следует преобразовать перед передачей в буфер обмена):

- текст содержит простой поток печатаемых символов ANSI без внедренных форматирующих или управляющих кодов;
- каждая строка заканчивается символом возврата каретки и перевода строки;
- весь блок текста заканчивается символом `NULL`.

Средства Windows не сохраняют пересылаемые в буфер данные. Вместо этого необходимо явно выделить блок памяти, скопировать в него текст, а затем передать в буфер *дескриптор* этого блока памяти. В буфере хранится только дескриптор, предоставляющийся любой программе, которая получает доступ к буферу. Следовательно, первым действием при передаче текста в буфер должно быть:

1. Выделение блока из динамически распределяемой (*heap*) области памяти вызовом функции `Win32 API ::GlobalAlloc`. Ее синтаксис выглядит так:

```
HGLOBAL GlobalAlloc
(UINT uFlags,          // атрибуты размещения;
DWORD dwBytes);       // число байтов размещения
```

- Первый параметр `uFlags`. Ему присваивается один или несколько флагов, описывающих блок памяти. При выделении этого блока для использования буфером необходимо задать флаги `GMEM_MOVEABLE` и `GMEM_DDESHARE`. Кроме того, при добавлении флага `GMEM_ZEROINIT` блок памяти будет инициализирован нулями. Значения всех задаваемых флагов смотрите в документации на функцию `::GlobalAlloc` в справочной системе.
- Второй параметр `dwBytes` задает желаемый размер блока памяти в байтах. Задавая его, убедитесь в наличии места для символа `NULL`, завершающего текст. В случае удачи функция `::GlobalAlloc`

возвращает дескриптор блока памяти, а при невозможности выделения памяти – возвращается значение NULL.

Рассмотрим случай, когда текст, копируемый в буфер, содержится внутри символьного массива Buffer, заканчивается символом NULL. Тогда память выделяется так:

```
HGLOBAL HMem;

HMem=::GlobalAlloc
(GMEM_MOVEABLE | GMEM_DDESHARE,
strlen (Buffer) + 1);
if (HMem == NULL)
{
    AfxMessageBox ("Error copying data to the Clipboard.");
    return;
}
```

2. Доступ к блоку выделенной памяти можно получить, вызвав функцию Win32 API ::GlobalLock, и задав в качестве параметра hglbMem – дескриптор, получаемый из функции GlobalAlloc. В случае удачного завершения функция ::GlobalLock возвращает указатель на начало блока памяти. В случае ошибки – возвращается NULL. Синтаксис функции:

```
LPVOID GlobalLock (HGLOBAL hglbMem);
```

Следующий код получает указатель на выделенную в предыдущем примере область памяти.

```
char *PMem;

PMem = (char *)::GlobalLock (HMem);
if (PMem == NULL)
{
    ::GlobalFree (HMem);
    AfxMessageBox ("Error copying data to the Clipboard.");
    return;
}
```

Если вызов функции ::GlobalLock завершается ошибкой, то с помощью передачи дескриптора памяти в функцию Win32 API ::GlobalFree перед выходом из функции блок памяти освобождается.

3. В выделенный блок памяти необходимо скопировать текст, пересылаемый в буфер обмена. Для этого используется функция Win32 API ::lstrcp. Функция ::lstrcp копирует строку, заканчивающуюся символом NULL, из области, заданной вторым параметром, в область, заданную первым.

```
::lstrcp (PMem, Buffer);
```

Если текст, добавляемый в буфер, не соответствует формату или не содержит завершающий символ – NULL, то для копирования данных можно написать собственную программу с необходимыми преобразованиями.

4. Для передачи в буфер следует подготовить дескриптор выделенной памяти, вызвав функцию Win32 API ::GlobalUnlock.

```
::GlobalUnlock (HMem);
```

5. Откройте буфер обмена для получения доступа к нему, вызвав функцию OpenClipboard класса CWnd. Функция OpenClipboard принадлежит классу CWnd, поэтому предполагается, что код внутри функции класса, наследуемого от CWnd, такой же, как и у функции класса представления

программы. В случае удачного завершения функция `OpenClipboard` возвращает значение `TRUE`. Значение `FALSE` возвращается, если другая программа открыла буфер и еще не закрыла его. В таком случае операцию с буфером нельзя завершить. Например:

```
if (!OpenClipboard ())
{
    ::GlobalFree (HMem);
    AfxMessageBox ("Clipboard not available.");
}
```

6. Открыв буфер, необходимо удалить текущее содержимое буфера, вызвав функцию Win32 API `::EmptyClipboard`. Ее синтаксис:

```
BOOL EmptyClipboard (VOID);
```

7. Выполните передачу текста в буфер, вызывая функцию Win32 API `::SetClipboardData`. Ее синтаксис выглядит так:

```
HANDLE SetClipboardData (UINT uFormat, HANDLE hData);
```

- Первый параметр – `uFormat` – задает *формат данных*, передаваемых в буфер. Для передачи текста в качестве этого параметра задается значение `CF_TEXT`. Смотрите описание других стандартных форматов в документации на эту функцию.
- Второй параметр – `hData` – является *дескриптором выделенной памяти*, содержащей данные. В рассматриваемом примере вызов этой функции будет выглядеть так:

```
::SetClipboardData (CF_TEXT, HMem);
```

Дескриптор памяти после вызова функции `::SetClipboardData` использовать *нельзя*. Не рекомендуются попытки чтения из памяти или записи в память, а также вызовы функции `::GlobalFree` для освобождения блока памяти. Сейчас дескриптор принадлежит системе Windows, автоматически освобождающей соответствующие блоки памяти.

8. Чтобы избежать соблазна в использовании дескриптора, присвойте переменной, содержащей дескриптор, значение `NULL` сразу после вызова функции `::SetClipboardData`. Если после вызова этой функции требуется получить доступ к блоку памяти, то используйте стандартную процедуру получения данных из буфера командой `Paste`.
9. Если для получения доступа к блоку памяти, *не* передаваемому в буфер обмена, используется функция `::GlobalAlloc`, то вначале вызовите функцию `::GlobalUnlock` (если до этого была вызвана функция `::GlobalLock`), а затем, по окончании использования памяти, вызовите функцию `::GlobalFree`.
10. Закройте буфер, вызвав функцию Win32 API `::CloseClipboard`. Поскольку в один момент времени только одна программа может открыть буфер, его необходимо закрыть как можно быстрее. Синтаксис этой функции выглядит так:

```
BOOL CloseClipboard (VOID);
```

Ниже приведен пример обработчика сообщения `COMMAND` для команды `Copy` (определенной как функции класса представления), объединяющий воедино описанные в разделе действия. Обратите внимание, что `Buffer` – это глобальный массив символов, содержащий текст, оканчивающийся символом `NULL` и предназначенный для копирования в буфер обмена.

```
char Buffer [] = "Sample text to be copied to the Clipboard.";

void CProgView::OnEditCopy()
{
```

```

// TODO: Здесь добавьте собственный код обработчика

HGLOBAL HMem;
char *PMem;

// Выделите блок памяти
HMem = (char *)::GlobalAlloc
    (GMEM_MOVEABLE | GMEM_DDESHARE,
     strlen (Buffer) + 1);
if (HMem == NULL)
{
    AfxMessageBox ("Error copying data to the Clipboard.");
    return;
}

// Заблокируйте выделенный блок памяти и получите указатель
PMem = (char *)::GlobalLock (HMem);
if (PMem == NULL)
{
    ::GlobalFree (HMem);
    AfxMessageBox ("Error copying data to the Clipboard.");
    return;
}

// Скопируйте выделенный текст в выделенный блок памяти
::lstrcpy (PMem, Buffer);

// Разблокируйте выделенную память
::GlobalUnlock (HMem);

// Откройте буфер Clipboard:
if (!OpenClipboard ())
{
    ::GlobalFree (HMem);
    AfxMessageBox ("Error copying data to the Clipboard.");
    return;
}

// Удалите текущее содержимое буфера
::EmptyClipboard ();

// Передайте дескриптор памяти в буфер
::SetClipboardData (CF_TEXT, HMem);
HMem = NULL;

// Закройте буфер
::CloseClipboard ();
}

```

В реальном приложении массив Buffer должен содержать заканчивающийся символом NULL выделенный текст.

В отличие от обработчика сообщения COMMAND команды Copy, соответствующий обработчик команды Cut (с помощью функции OnEditCut), вызывает функцию OnEditCopy, а затем удаляет из документа выделенный текст.

## Извлечение текста из буфера обмена

Для программы позволяющей вставить из буфера только стандартный текст обработчик сообщения `UPDATE_COMMAND_UI` команды `Paste` делает команду доступной только тогда, когда буфер содержит стандартный текст. Чтобы определить, содержит ли буфер данные, соответствующие специальному формату, вызовите функцию Win32 API `::IsClipboardFormatAvailable`, задав в параметре желаемый формат. Ее синтаксис выглядит так:

```
BOOL ::IsClipboardFormatAvailable (UINT uFormat);
```

Ниже приведен пример обработчика сообщения `UPDATE_COMMAND_UI` команды `Paste`. В зависимости от того, содержит ли буфер текст, команда будет доступна или нет.

```
void CProgView::OnUpdateEditPaste(CCmdUI* pCmdUI)
{
    //TODO: Здесь добавьте собственный код обработчика

    pCmdUI->Enable (::IsClipboardFormatAvailable (CF_TEXT));
}
```

В буфере одновременно могут содержаться данные нескольких форматов:

- Чтобы определить *все* форматы, доступные в данный момент, вызовите функцию Win32 API `::EnumClipboardFormats`.
- Чтобы определить *лучший* формат из предварительно заданного списка приоритетов, вызовите функцию Win32 API `::GetPriorityClipboardFormat`.

Команде `Paste` необходим обработчик сообщения `COMMAND` для получения текста из буфера и последующей его вставки в документ. В следующих процедурах и примерах обработчик команды `Paste` выделяет временный буфер и копирует в него текст из буфера обмена. После этого буфер обмена закрывается, и программа соответствующим образом обрабатывает текст, добавляет его в документ, а затем освобождает буфер. При альтернативном решении программа может копировать текст непосредственно из буфера обмена в структуру данных, в которой хранится текст документа. При этом предполагается, что операции выполняются быстро и буфер открыт недолго. Обычная процедура получения текста из буфера по команде `Paste` начинается вызовом функции `CWnd::OpenClipboard` для открытия буфера.

Далее необходимо вызвать функцию Win32 API `::GetClipboardData` для получения дескриптора блока памяти, содержащего текст, помещенный в буфер обмена, и выделить временный буфер, чтобы хранить копию текста, помещенного в буфер обмена. Потом следует вызвать функцию `::GlobalLock`, чтобы заблокировать память буфера обмена и получить на нее указатель. Далее выполняется собственно копирование текста из буфера обмена во временный буфер. По завершении копирования следует вызвать функцию `::GlobalUnlock`, чтобы разблокировать память буфера обмена и функцию `::CloseClipboard` для закрытия этого буфера. Теперь можно приступить к обработке текста во временном буфере. После окончания обработки программой текста во временном буфере и добавления текста в документ, необходимо освободить временный буфер. В подробностях эта процедура выглядит так:

1. Следует открыть буфер обмена функцией `OpenClipboard()`.
2. Далее, вызывая функцию Win32 API `::GetClipboardData`, можно получить дескриптор блока памяти, содержащего текст.

```
HANDLE HClipText;

HClipText = ::GetClipboardData (CF_TEXT);
if (HClipText == NULL)
```



```

{
    ::CloseClipboard ();
    AfxMessageBox ("Error obtaining text from Clipboard.");
    return;
}

```

Передаваемый в функцию `::GetClipboardData` параметр определяет требуемый формат данных. Если буфер не содержит данных заданного формата, то возвращается значение `NULL`. Это может случиться, если другой процесс удалил данные из буфера после того, как программе стала доступна команда `Paste`, но до открытия буфера. Если требуются данные в формате `CF_TEXT`, но буфер содержит только данные *других* текстовых форматов (`CF_OEMTEXT` или `CF_UNICODETEXT`), то Windows преобразует их в формат `CF_TEXT` автоматически. Дескриптор, полученный при вызове функции `::GetClipboardData`, остается корректным только до вызова функции `::CloseClipboard`. Данные из блока памяти можно читать или копировать, но *нельзя* изменять или вызывать функцию `::GlobalFree` для освобождения блока памяти. Он должен оставаться неизменным, чтобы буфер обмена мог при последующих запросах какой-либо программы предоставить данные.

- Получив дескриптор блока памяти, содержащего текст, можно выделить временный буфер для хранения его копии. Вызовите функцию Win32 API `::GlobalSize`, чтобы определить размер блока памяти буфера. Например, в следующем фрагменте программы используется оператор `new` для выделения временного буфера достаточного объема. В этом примере функция `::GetClipboardData` возвращает дескриптор памяти `HClipText`.

```

char *PTempBuffer = new char [::GlobalSize (HClipText)];
if (PTempBuffer == 0)
{
    ::CloseClipboard ();
    AfxMessageBox ("Out of memory!");
    return;
}

```

- Далее необходимо вызвать функцию `::GlobalLock` для получения указателя на блок памяти.

```

char *PClipText;

PClipText = (char *)::GlobalLock (HClipText);
if (PClipText == NULL)
{
    ::CloseClipboard ();
    delete [ ] PTempBuffer;
    AfxMessageBox ("Error obtaining text from Clipboard");
    return;
}

```

- Данные из буфера обмена теперь можно прочитать или скопировать, но сделать это нужно как можно быстрее.
- Перед закрытием буфера убедитесь, что для разблокирования памяти буфера была вызвана функция `::GlobalUnlock`.

```

::GlobalUnlock (HClipText);

```

- Закройте буфер обмена. *Нельзя* использовать дескриптор памяти, полученный из буфера обмена, после вызова функции `::CloseClipboard`.

```

::CloseClipboard ();

```

8. Если текст скопирован во временный буфер, то программа в любой момент может обработать текст и очистить буфер.

```
delete [] PTempBuffer;
```

Полный текст обработчика сообщения COMMAND, определенного как функция класса представления и получающего текст из буфера обмена по команде Paste, приведен ниже. Используемая в примере функция InsertText вставляет из буфера в текущий документ текст, заканчивающийся символом NULL.

```
void CProgView::OnEditPaste()
{
    // TODO: Здесь добавьте собственный код обработчика

    HANDLE HClipText;
    char *PClipText;
    char *PTempBuffer;

    // 1. Откройте буфер
    if (!OpenClipboard ())
    {
        AfxMessageBox ("Could not open Clipboard.");
        return;
    }

    // 2. Получите дескриптор данных буфера
    HClipText = ::GetClipboardData (CAF_TEXT);
    if (HClipText == NULL)
    {
        ::CloseClipboard ();
        AfxMessageBox ("Error obtaining text from Clipboard");
        return;
    }

    // 3. Разместите временный буфер для сохранения текста из буфера
    PTempBuffer = new char [::GlobalSize (HClipText)];
    if (PTempBuffer == 0)
    {
        ::CloseClipboard ();
        AfxMessageBox ("Out of memory!");
        return;
    }

    // 4. Заблокируйте дескриптор текста в буфере и получите указатель
    PClipText = (char *)::GlobalLock (HClipText);
    if (PClipText == Null)
    {
        ::CloseClipboard ();
        delete [] PTempBuffer;
        AfxMessageBox ("Error obtaining text from Clipboard");
        return;
    }

    // 5. Скопируйте текст из буфера
    ::lstrcpy (PTempBuffer, PClipText);
}
```

```

// 6. Разблокируйте блок памяти буфера
::GlobalUnlock (HClipText);

// 7. Закройте буфер
::CloseClipboard ();

// 8. Вставьте текст в документ и очистите временный буфер:
InsertText (PTempBuffer);
delete [] PTempBuffer;
}

```

## Обмен графикой через буфер обмена

Используя буфер для обмена растровыми изображениями, можно передать в программу или процесс графическую информацию. В этом разделе вы узнаете, как поместить растровое изображение в буфер и как получить его из буфера. Процедуры, рассмотренные здесь, полезны для разработки программ рисования, текстовых процессоров или других программ, отображающих растровую графику.

### Перемещение растрового изображения в буфер обмена

Перенести растровое изображение в буфер можно с помощью команд Cut и Copy. Обработчики сообщения UPDATE\_COMMAND\_UI этих команд доступны, если растровое изображение выделено. В следующем примере предполагается, что переменной m\_IsSelection типа BOOL класса представления присваивается значение TRUE только при выборе растрового изображения. Если можно перенести в буфер текст, то переменной m\_IsSelection присваивается значение TRUE в случаях, когда выбрано *или* растровое изображение, *или* текст.

```

void CProgView::OnUpdateEditCut (CCmdUI* pCmdUI)
{
    // TODO: Здесь добавьте собственный код обработчика
    pCmdUI->Enable (m_IsSelection);
}

void CProgView::OnUpdateEditCopy (CCmdUI* pCmdUI)
{
    // TODO: Здесь добавьте собственный код обработчика
    pCmdUI->Enable (m_IsSelection);
}

```

Для передачи растрового изображения в буфер следует воспользоваться обработчиками сообщений для команд Cut и Copy. Для этого необходимо вызвать функцию CWnd::OpenClipboard, чтобы открыть буфер, и функцию EmptyClipboard, чтобы удалить текущее содержимое буфера. Затем следует вызвать функцию ::SetClipboardData, передавая ей код CF\_BITMAP как первый параметр и дескриптор растрового изображения, помещаемого в буфер, как второй. Существует несколько способов создания растрового изображения и получения его дескриптора (см. гл. 20). В следующем примере программы создается пустое растровое изображение, и в него копируются данные. Далее необходимо вызвать функцию ::CloseClipboard, чтобы закрыть буфер. Для команды Cut обработчик сообщений COMMAND в заключение должен удалить из документа графические данные. Например, следующий обработчик сообщений COMMAND для команды Copy создает растровое изображение с текущим содержимым окна представления и копирует его в буфер обмена. Если программа также обеспечивает копирование текста в буфер, то функция OnEditCopy должна определять формат выделенных данных и переключаться на соответствующую программу.

```

void CProgView::OnEditCopy()
{
    // TODO: Здесь добавьте собственный код обработчика

    CBitmap BitmapClip;
    CClientDC ClientDC (this);
    CDC MemDC;
    RECT Rect;

    // создайте пустое растровое изображение
    GetClientRect (&Rect);
    BitmapClip.CreateCompatibleBitmap
        (&ClientDC,
         Rect.right - Rect.left,
         Rect.bottom - Rect.top);

    // создайте объект памяти и перешлите в него
    // растровое изображение:
    MemDC.CreateCompatibleDC (&ClientDC);
    MemDC.SelectObject (&BitmapClip);

    // скопируйте содержимое окна представления в
    // растровое изображение:
    MemDC.BitBlt
        (0,
         0,
         Rect.right - Rect.left,
         Rect.bottom - Rect.top,
         &ClientDC,
         0,
         0,
         SRCCOPY);

    // Откройте буфер
    if (!OpenClipboard ())
        return;

    // Удалите текущее содержимое буфера
    ::EmptyClipboard ();

    // Передайте дескриптор растрового изображения в буфер
    ::SetClipboardData (CF_BITMAP, BitmapClip.m_hObject);

    // Предотвратите удаление растрового изображения
    BitmapClip.Detach ();

    // Закройте буфер
    ::CloseClipboard ();
}

```

Начинается функция OnEditCopy с вызова функции CreateCompatibleBitmap класса CBitmap, позволяющего создать пустое изображение, размеры которого равны размерам окна представления. Затем создается объект контекста устройства памяти и выбирается новое растровое изображение. Вызывается функция BitBlt класса CDC (см. гл. 20) для того, чтобы скопировать все содержимое окна представления в растровое изображение. Функция OnEditCopy вызывает функцию

`::SetClipboardData`, чтобы добавить растровое изображение в открытый буфер. Она присваивает первому параметру значение `CF_BITMAP`, указывающее на формат растрового изображения, а второму параметру – значение `BitmapClip.m_hObject`, содержащее дескриптор растрового изображения. (Переменная `m_hObject` наследуется классом `CBitmap` от класса `CGdiObject`.) После вызова функции `::SetClipboardData` для перемещения растрового изображения в буфер использовать или удалять его *не* нужно. В примере функция `OnEditCopy` вызывает функцию `Detach` класса `CGdiObject` для удаления дескриптора растрового изображения из объекта `BitmapClip`. Если этого не выполнить, то деструктор класса `CBitmap` автоматически удалит растровое изображение, когда `BitmapClip` выйдет из области видимости (т. е. при возврате из функции `OnEditCopy`). В заключение буфер обмена закрывается.

## Извлечение растрового изображения из буфера обмена

Чтобы сделать команду `Paste` доступной или недоступной, обработчик сообщения `UPDATE_COMMAND_UI` для команды `Paste` может передать флажок `CF_BITMAP` в функцию `::IsClipboardFormatAvailable`. Это позволит определить, содержит ли буфер данные в формате растрового изображения. Если в программе поддерживается вставка текста и растрового изображения, то функция `OnUpdateEditPaste` должна проверить также наличие формата `CF_TEXT`, т.е. наличие в буфере обмена текстового содержимого.

```
void CProgView::OnUpdateEditPaste(CCmdUI* pCmdUI)
{
    // TODO: Здесь добавьте собственный код обработчика
    pCmdUI->Enable (::IsClipboardFormatAvailable (CF_BITMAP));
}
```

Для получения растрового изображения из буфера обмена обработчик сообщения `COMMAND` команды `Paste` сначала вызывается функция `OpenClipboard` для открытия буфера. Затем передается значение `CF_BITMAP` в функцию `GetClipboardData` для получения дескриптора растрового изображения. Этот дескриптор используется для копирования или отображения растрового изображения (без изменения содержимого). Далее вызывается функция `::CloseClipboard` для закрытия буфера. Рассмотрим пример обработчика сообщений `COMMAND` для команды `Paste`.

```
void CProgView::OnEditPaste()
{
    // TODO: Здесь добавьте собственный код обработчика

    CClientDC ClientDC (this);
    CBitmap BitmapClip;
    BITMAP BitmapClipInfo;
    HANDLE HBitmapClip;
    CDC MemDC;

    // Откройте буфер
    if (!OpenClipboard ())
        return;

    // Получите дескриптор растрового изображения из буфера
    HBitmapClip = ::GetClipboardData (CF_BITMAP);
    if (HBitmapClip == NULL)
    {
        ::CloseClipboard ();
        return;
    }
}
```

```

// Используйте дескриптор растрового изображения
// для его отображения

// Инициализируйте объект растрового изображения,
// используя дескриптор из буфера
BitmapClip.Attach (HBitmapClip);

// Получите информацию по растровому изображению
BitmapClip.GetObject (sizeof (BITMAP), &BitmapClipInfo);

// Создайте объект контекста устройства и выберите
// в нем растровое изображение
MemDC.CreateCompatibleDC (&ClientDC);
MemDC.SelectObject (&BitmapClip);

// Скопируйте содержимое растрового изображения в рабочую область
ClientDC.BitBlt
(0
,
0,
BitmapClipInfo.bmWidth,
BitmapClipInfo.bmHeight,
&MemDC,
0,
0,
SRCCOPY);

// Удалите дескриптор растрового изображения
// из объекта растрового изображения
BitmapClip.Detach ();

// Закройте буфер
::CloseClipboard ();
}

```

Этот обработчик сообщений COMMAND для команды Paste получает растровое изображение из буфера и отображает его в окне представления (в левый верхний угол окна помещается левый верхний угол растрового изображения). Если программа обеспечивает передачу текста, функция OnEditPaste должна проверять оба формата (CF\_BITMAP и CF\_TEXT), а затем переходить на соответствующую программу в зависимости от обнаруженного формата данных. (Если данные нужного формата в буфере не обнаружены, то функция ::GetClipboardData должна вернуть значение NULL.) Если в буфере содержатся данные обоих форматов, то должен выбираться один из них.

Для управления растровым изображением, полученным из буфера, функция OnEditPaste объявляет объект класса CBitmap. Вызывается функция ::GetClipboardData для получения дескриптора растрового изображения, который передается в функцию Attach класса CGdiObject для инициализации объекта растрового изображения содержимым буфера. Затем вызывается функция GetObject класса CGdiObject для получения информации о растровом изображении и записывается в поле BITMAP структуры BitmapClipInfo. Функция OnEditPaste создает контекст устройства памяти, выбирает растровое изображение и вызывает BitBlt, чтобы скопировать изображение в рабочую область окна. При этом значения ширины и высоты данного изображения получают из структуры BITMAP, воспользовавшись содержимым полей bmWidth и bmHeight. Если размеры растрового изображения превышают размеры окна представления, то Windows автоматически отсекает часть, выходящую за пределы окна. Перед очисткой буфера функция OnPaste вызывает функцию Detach, чтобы удалить дескриптор растрового изображения из объекта BitmapClip. (В этом

случае, как и в рассмотренном ранее, деструктор класса CBitmap не может уничтожить растровое изображение, хранящееся в буфере.)

## Данные зарегистрированных форматов в буфере обмена

В документации на функцию SetClipboardData описаны стандартные форматы буфера обмена. Однако передаваемые данные могут не соответствовать ни одному из них. Например, в рамках текстового процессора можно сохранять форматированный текст, используя собственный пользовательский формат. Символы текста могут быть сохранены с внедренными кодами, задающими шрифт и другие свойства текста. Тем не менее, можно использовать буфер для передачи данных с помощью вызова функции Win32 API ::RegisterClipboardFormat, позволяющей зарегистрировать собственный формат данных. В функцию ::RegisterClipboardFormat можно передать любое нужное имя, и она возвратит идентификатор формата.

```
UINT RegisterClipboardFormat (LPCTSTR lpszFormat);
```

Можно использовать другие функции буфера для переноса форматированного текста теми способами, которые применялись для передачи простого текста. Однако вместо указания формата CF\_TEXT задается его идентификатор, возвращенный функцией ::RegisterClipboardFormat. Например, задав формат текста:

```
UINT TextFormat;
```

```
TextFormat = ::RegisterClipboardFormat ("MyAppText");
```

можно передать текст, соответствующий этому формату, в буфер:

```
::SetClipboardData (TextFormat, HMyText);
```

Здесь имя HMyText задает дескриптор блока памяти с форматированным текстом, выделенным функцией ::GlobalAlloc. Аналогично, в функции ::IsClipboardAvailable и ::GetClipboardData необходимо передать TextFormat, а не CF\_TEXT. В буфер можно поместить *несколько* блоков данных при условии, что они имеют разные форматы.

Несколько программ могут обмениваться данными с буфером, используя зарегистрированный формат, если каждая из программ знает имя формата и может интерпретировать описанные данные. (Эта возможность основана на том, что если специальный формат уже зарегистрирован под указанным именем, то функция ::RegisterClipboardFormat возвращает идентификатор этого же формата и не регистрирует новый.) Учитывая это свойство, необходимо выбирать уникальное имя, чтобы избежать *случайного* совместного использования формата, зарегистрированного другой программой!

При перемещении в буфер данных в зарегистрированном формате, необходимо *также* поместить туда эквивалентный блок данных в стандартном формате. Это позволит отображать или получать данные тем программам, которые не знакомы с вашим пользовательским форматом. Например, при перемещении в буфер специально отформатированного текста добавьте простой текст. В приведенном ниже примере HFormattedText – дескриптор блока текста в пользовательском формате, HPlainText – в стандартном текстовом формате буфера, а TextFormat – индекс пользовательского формата, возвращенный функцией ::RegisterClipboardFormat.

```
::SetClipboardData (TextFormat, HFormattedText);  
::SetClipboardData (CF_TEXT, HPlainText);
```

Очевидно, что одни форматы могут быть более информативными по сравнению с другими. В примере, приведенном выше, текст в пользовательском формате более информативен, чем простой текст. Чтобы к вашим данным получили доступ как можно больше программ Windows, необходимо

добавить достаточно много форматов. Программы Windows обычно вызывают функцию `::EnumClipboardFormats`, перенумеровывающую доступные форматы буфера, и используют первый формат, который они могут воспринять. Поскольку функция `::EnumClipboardFormats` сообщает форматы в том порядке, в котором они добавлялись в буфер, то необходимо добавлять сначала более информативные, а затем менее информативные форматы. Вследствие этого другие программы будут использовать наиболее информативные форматы данных для данного содержимого буфера.

## Резюме

---

Рассмотрены вопросы запуска и управления новыми процессами. Описаны некоторые способы связи и координации действий отдельных процессов: с применением объектов синхронизации, передачей данных по каналам и с помощью общих блоков памяти, а также обменом данными через буфер обмена.

- *Запуск процесса.* Новый процесс создается при выполнении программы. Альтернативой этому является способ, при котором программа, называемая *родительской*, вызывает функцию Win32 API `::CreateProcess` для запуска другой, *дочерней*, программы. Функция `::CreateProcess` предоставляет дескриптор дочернего процесса. Родительский процесс передает его одной из нескольких функций Win32 API: функции `::GetExitCodeProcess` для получения кода возврата процесса, функции `::WaitForSingleObject` для ожидания завершения процесса, функции `::SetPriorityClass` для изменения приоритета процесса или `::TerminateProcess` для его остановки. При передаче идентификатора процесса в функцию `::OpenProcess` программа получает дескриптор другого процесса даже в том случае, если она не является родителем этого процесса.
- *Синхронизация процессов.* Для синхронизации отдельных процессов используются мьютексы, семафоры или события. При этом каждый процесс получает собственный дескриптор объекта синхронизации. Если задать объекту имя при вызове функции `::Create...` (`::CreateMutex`, `::CreateSemaphore` или `::CreateEvent`) или `::Open...` (`::OpenMutex`, `::OpenSemaphore` или `::OpenEvent`), то отдельные процессы получают дескрипторы общих объектов синхронизации.
- *Обмен между процессами.* Анонимным каналом называется механизм обмена информацией между процессами (обычно родительским и дочерним) аналогично записи файлов на диск и чтения с него. При обмене информацией между процессами с использованием *общей памяти* оба процесса вызывают функции `::CreateFileMapping` и `::MapViewOfFile`. Использование *буфера* облегчает обмен простыми данными. При этом они должны соответствовать одному из стандартных форматов (текста или растрового изображения) или воспринимаемому программой зарегистрированному пользовательскому формату.
- *Буфер обмена.* Доступ к буферу обмена предоставляется с помощью команд Cut, Copy и Paste меню Edit. Копируя *блок текста* в буфер обмена (команды Cut или Copy) или вставляя текст из буфера (команда Paste), можно обмениваться текстовой информацией. Чтобы *скопировать* текст в буфер, вызывается функция `::GlobalAlloc`, выделяющая блок памяти. Текст копируется в этот блок, а затем буфер снабжается дескриптором выделенной области памяти. Чтобы *вставить* текст из буфера, запрашивается дескриптор блока памяти, содержащий текст. Этот дескриптор можно использовать для чтения или копирования текста в закрытую область памяти. Копируя *растровое изображение* в буфер, или вставляя растровое изображение из буфера, можно обмениваться графической информацией. При *копировании* растрового изображения в буфер обмена передается дескриптор растрового изображения. При *вставке* растрового изображения из буфера запрашивается дескриптор этого изображения, а затем программа делает копию или отображает его.



- *Нестандартные форматы.* Обмен данными, которые не соответствуют ни одному из стандартных форматов буфера, осуществляется с помощью функции `::RegisterClipboardFormat`, регистрирующей собственный формат. Для обмена текстовыми данными с помощью буфера можно использовать соответствующие процедуры. При этом вместо индекса, указывающего на текстовый формат, передается индекс, возвращаемый функцией `::RegisterClipboardFormat`. Вызовите функцию `::IsClipboardFormatAvailable`, чтобы определить, содержит ли буфер данные, соответствующие указанному формату.

## Глава 24

# OLE-механизм в Visual C++

---

- Внедрение, связывание и автоматизация
- Программа-сервер ExpServer
- Программа-контейнер ExpContainer

При передаче данных из одной программы в другую с помощью буфера обмена программа, принимающая данные, должна быть способна отображать и изменять их. При этом формат данных должен входить в относительно небольшой набор стандартных форматов, воспринимаемых большинством программ. Если принимающая программа не может редактировать и изменять данные (например, текстовый процессор может отображать текст, но не может редактировать растровое изображение), то необходимо вручную переключиться на исходную программу, отредактировать или вновь создать данные, а затем повторить все операции копирования и вставки. Механизм OLE – Object Linking and Embedding (*связывание и внедрение объектов*) – метод обмена данными между программами, позволяющий преодолеть эти ограничения. После добавления данных в документ принимающей программы он поддерживает связь с создавшей его исходной программой.

Принимающей программе не нужно воспринимать формат данных, так как за их отображение и редактирование отвечают исходная программа и механизм OLE. При редактировании данных исходная программа запускается автоматически, вследствие чего команды редактирования становятся доступными. Следовательно, при использовании механизма OLE можно передавать данные в *любом* формате *любой* программе, которая поддерживает этот механизм. Чтобы изменить данные в принимающей программе, не нужно вручную запускать исходную программу и даже помнить ее имя (хотя она должна быть установлена и соответствующим образом зарегистрирована в реестре Windows).

Механизм OLE позволяет построить документ, содержащий блоки данных, созданных в различных программах. Такой документ называют *составным (compound)*. Если данные собраны, то можно сосредоточиться на документе, не отслеживая различные исходные программы. Таким образом, OLE – это механизм, *концентрирующий внимание на документе*, а не на приложениях, в которых он был создан.

Но чем больше удобств механизм OLE предоставляет пользователю, тем больше работы у программиста. В этой главе кратко рассмотрено введение в OLE-программирование (для подробного рассмотрения понадобилось бы несколько больших томов). Основное внимание уделяется использованию кода OLE, предоставляемого библиотекой MFC и мастерами Visual C++. Глава начинается с обзора трех основных механизмов OLE: внедрения, связывания и автоматизации. В этой главе будет создан простой *сервер* OLE, являющийся исходной программой. Затем будет создан простой *контейнер* OLE, являющийся программой, принимающей данные OLE. В этих параграфах описан код, предоставляемый мастерами и библиотекой MFC, а также добавляемый программистом код.

В документации Visual Studio.NET содержится замечание, рекомендуемое по возможности заменять применения технологии OLE на технологию ActiveX, позволяющую представлять не только данные, но и элементы управления (см. гл. 25).

## Внедрение, связывание и автоматизация

---

В рамках OLE-технологии поддерживаются механизмы:

- *внедрения объекта;*
- *связывания объекта;*
- *автоматизации.*

1. Наиболее распространенным из механизмов OLE-технологии является *внедрение объекта*. Под термином *объект* подразумевается блок данных, созданных сервером и отображаемых в контейнере. При внедрении объекта приложение-контейнер сохраняет его как часть документа контейнера, в который вставлен дополнительный компонент данных. *Внедрить объект* можно одним из двух способов:

- *Первый способ внедрения:* можно скопировать или вырезать блок данных из документа программы-сервера, а затем поместить его в документ программы-контейнера. Если данные заданы *не* в собственном формате программы-контейнера, то при выполнении команды Paste меню Edit программа-контейнер автоматически внедряет данные, а не просто статически передает (вставляет) содержимое буфера обмена, как описано в предыдущей главе. Обычно меню Edit контейнера содержит команду Paste Special для явного внедрения или передачи данных с помощью любого другого метода. Разновидностью этого метода передачи является обычная операция *drag-and-drop*, которая позволяет *перетаскивать* данные из окна документа сервера в окно документа контейнера. Данные из документа сервера после их внедрения в контейнер утрачивают связь с оригиналом документа сервера. Таким образом, контейнер использует *отдельную копию* данных (изменение или удаление оригинала документа сервера на данных копии не отражается).
- *Второй способ внедрения:* чтобы внедрить объект, можно выбрать команду Insert New Object... в меню Edit программы-контейнера. (Эта команда может быть помещена в другое меню и иметь отличающееся название.) В результате отобразится диалоговое окно со списком объектов различных типов. (Каждая установленная программа-сервер OLE регистрирует в реестре Windows один или более типов объектов. Эти типы перечисляются в данном диалоговом окне.) При выборе типа объекта OLE автоматически запускает программу-сервер для создания объекта данных. Второй способ полезен именно для *создания нового блока* внедренных данных, а не для внедрения уже существующего.

Для редактирования внедренного объекта можно воспользоваться одним из двух способов:

- *Первый способ редактирования: на месте (in place).* При таком способе объект отображается в окне контейнера. Однако программа-сервер временно объединяет команды меню и кнопки панели инструментов с аналогичными средствами программы-контейнера и делает доступными комбинации клавиш, дополняя тем самым средства редактирования программы-контейнера своими возможностями. Чтобы инициировать редактирование на месте, выполните двойной щелчок кнопкой мыши внутри объекта или выделите объект и выберите команду Edit в подменю Object меню Edit контейнера. Подменю Object озаглавляется в соответствии с типом внедренного объекта. Например, подменю будет озаглавлено "Bitmap Image Object", если объект – растровый рисунок, сгенерированный программой Windows Paint.
- *Второй способ редактирования:* можно редактировать внедренный объект в окне программы-сервера. Такой режим редактирования называют *полностью открытым (fully opened)*, так как сервер полностью открывает свое собственное окно, не используя окно контейнера. Чтобы начать редактирование таким способом, выделите объект и выберите команду Open в подменю Object меню Edit контейнера.

Далее в этой главе будут описаны сервер и контейнер, реализующие оба метода редактирования объекта, и вы сможете поэкспериментировать с внедрением объектов после генерации программы.

2. Использование *связывания* объекта приводит к тому, что данные сохраняются в *программе-сервере*, а не в программе-контейнере. Сервер сохраняет их в одном из своих документов. При этом данные могут составлять часть или весь документ сервера. Контейнер *связан* с данными

исходного документа и рассматривает их как компоненты своего документа, но реально данные объекта не хранит. Чтобы внедрить связанный объект, необходимо скопировать требуемые данные из документа в сервер, а затем, выполнить команду Paste Link меню Edit контейнера. В некоторых программах объект можно внедрить, выполняя команду Paste Special... меню Edit, а затем выбирая опцию Paste Link. Связанный объект отображается в документе контейнера и может редактироваться одним из следующих способов.

- Для редактирования объекта выполните *двойной щелчок кнопкой мыши внутри объекта в окне контейнера*.
- Можно редактировать связанный объект, выделив его, а затем *выбрать команду Open или Edit в подменю Object меню контейнера Edit*.
- Еще один способ редактирования – это *непосредственный запуск программы-сервера и открытие исходного документа*.

Независимо от используемого метода редактирование всегда выполняется в окне сервера, т.е. в полностью открытом режиме. Связывание объекта сложнее внедрения, потому что для редактирования необходимо открыть *два* документа (документ контейнера содержит связь объекта, документ сервера – данные объекта) и объект нельзя отредактировать на месте. Связывание позволяет создавать и поддерживать главный документ в программе-сервере и автоматически изменять один или несколько связанных объектов внутри других программ. Например, можно создать электронную таблицу, поддерживать ее соответствующей программой, а затем использовать связывание объекта для добавления таблицы или ее части в документы других программ. При изменении первичной электронной таблицы *связанные объекты изменяются автоматически*. Если же вся электронная таблица или ее часть была скопирована и внедрена в другую программу, то внедренный фрагмент – отдельная копия исходных данных и *внедренный объект автоматически не обновляется*.

3. Автоматизация – третий механизм, поддерживаемый OLE. При его реализации программа – *сервер автоматизации* – использует некоторые из своих средств совместно с другой программой – *клиентом автоматизации*. Например, программа Web-браузера может предложить свои средства передачи данных другим программам. Какая-либо из этих программ может использовать указанные средства для отображения Web-страниц или загрузки файлов по интерфейсу FTP. В частности, клиент может изменять некоторые данные сервера, называемые *свойствами*, или вызывать некоторые функции сервера, называемые *методами*. Данный механизм назван *автоматизацией*, потому что он дает возможность программе клиента автоматизировать использование методов одной или нескольких других программ, а также работать с программными объектами других программ (а не писать свои собственные) и обеспечивает совместную работу отдельных программ.

Программа клиента автоматизации может содержать макроязык, позволяющий управлять другими программами и автоматизировать выполнение мультипрограммных приложений. (Например, можно написать макрос для управления приложениями Word, Excel и другими средствами Microsoft Office.) Библиотека MFC поддерживает клиента автоматизации и сервер автоматизации, а мастера Visual C++ генерируют текст программ автоматизации. Информация по автоматизации приведена в справочной системе.

## **Программа-сервер ExpServer**

Рассмотрим простую программу ExpServer с OLE-сервером. Программа ExpServer, созданная на основе третьей версии программы ScratchBook (см. гл. 12), создает рисунки, содержащие прямые линии, и сохраняет их в файле на диске. В отличие от ScratchBook программа ExpServer позволяет

программам-контейнерам внедрять в их документы рисунки. В ExpServer реализованы оба метода редактирования внедренных объектов: на месте и в полностью открытом режиме.

Программа ExpServer не поддерживает связывания, и объекты внедряются только командой New Object... меню Edit программы-контейнера (не используя команду копирования, вставки и технологии “drag-and-drop”). Для генерации программной оболочки, поддерживающей основные средства сервера, используется мастер Application Wizard. Затем добавляется текст, реализующий средства рисования ScratchBook, а также дополнительные средства OLE. Позже будет создана простая программа-контейнер, которую можно использовать для проверки работы ExpServer.

В литературе по OLE блок встроенных или связанных данных OLE называют *объектами*. Однако в документации по библиотеке MFC и далее в этой главе используется термин *компонент*, чтобы отличать его от объектов C++ (в частности тех, которые управляют компонентами OLE).

## Генерация программы-сервера

Для генерации текста программы используйте мастер Application Wizard (см. гл. 9). Назовите проект ExpServer, а на вкладках диалогового окна мастера Application Wizard выберите те же опции, которые выбирали в гл. 9, кроме следующих.

- На вкладке Compound Document Support выберите опцию Full-server, а другие опции оставьте неизменными. При выборе опции Full-server генерируется программа, которую можно запустить *либо* как автономное приложение (как во всех предыдущих примерах), *либо* как сервер OLE, поддерживающий внедренные и связанные компоненты (хотя связывание в программе ExpServer не реализовано). Сгенерированная программа выполняется только как сервер OLE, обрабатывающий внедренные элементы (автономно не выполняется, связывание не поддерживает). Обратите внимание: выбор опции Full-server вносит некоторые изменения в классы и текст программы, сгенерированный мастером Application Wizard. Мастер создает два новых класса: класс компонента сервера CExpServerSrvrItem и класс редактирования CInPlaceFrame, а также вносит ряд дополнений и изменений в стандартные MFC-классы. Эти изменения необходимы для создания программы-сервера OLE и описаны ниже. Опция Container генерирует программу-контейнер OLE и применяется при создании программы-контейнера ExpContainer. Опция Both Container And Server создает программу, являющуюся *и* контейнером, *и* сервером OLE.
- На вкладке Document Template Strings в поле File extension введите *srv*. Это задает стандартное расширение для файлов документов (см. гл. 12), сохраняемых в программе.

## Классы

*Класс приложения.* Мастер Application Wizard при генерации сервера OLE вносит дополнения в стандартную функцию InitInstance класса приложения:

1. В начало функции InitInstance добавляется вызов глобальной MFC-функции AfxOleInit. Этот вызов позволяет инициализировать MFC-библиотеку. (Файл Afxole.h включается мастером Application Wizard в предварительно компилируемый файл заголовков StdAfx.h, содержащий определения классов MFC.)

```
// Инициализация библиотек OLE
if (!AfxOleInit())
{
    AfxMessageBox(IDP_OLE_INIT_FAILED);
    return FALSE;
}
```

2. Мастер Application Wizard, создав шаблона документа, добавляет вызов функции `CsingleDocTemplate::SetServerInfo`. При этом определяются идентификаторы меню (они будут рассмотрены далее):

- меню, отображаемого программой ExpServer при редактировании в режиме *полного открытия* (идентификатор `IDR_SRVR_EMBEDDED`);
- меню, отображаемого при *выполнении программы как сервера*, применяемого для редактирования в режиме “in place” (идентификатор `IDR_SRVR_INPLACE`).

```
pDocTemplate->SetServerInfo(
    IDR_SRVR_EMBEDDED, IDR_SRVR_INPLACE,
    RUNTIME_CLASS(CInPlaceFrame));
```

Эти идентификаторы определяют соответствующие ресурсы сочетаний клавиш. При вызове функции `SetServerInfo` определяется класс управления окном (класс `CInPlaceFrame`, описанный ниже), которое обрамляет компонент OLE при редактировании на месте.

3. Далее мастер Application Wizard определяет новую переменную класса приложения `m_server`, являющуюся экземпляром класса `COleTemplateServer` и называемую *объектом шаблона сервера*. Этот объект при запуске программы как сервера OLE создает новый объект документа, используя информацию, хранящуюся в шаблоне документа. Мастер Application Wizard добавляет вызов функции `COleTemplateServer::ConnectTemplate`, которая предоставляет содержащую адрес шаблона документа переменную `m_server`.

```
m_server.ConnectTemplate(clsid, pDocTemplate, TRUE);
```

- Первый параметр метода `ConnectTemplate` задает *идентификатор типа документа*, поддерживаемого сервером. Идентификатор сгенерирован для программы ExpServer мастером Application Wizard:

```
// Этот идентификатор был сгенерирован статистически
// уникальным для вашего приложения. Можете изменить его,
// если сочтете нужным

// {0C60B0A4-27D5-4CE5-AE64-F6AA0F128C0B}
static const CLSID clsid =
{ 0xC60B0A4, 0x27D5, 0x4CE5,
  { 0xAE, 0x64, 0xF6, 0xAA, 0xF, 0x12, 0x8C, 0xB }
};
```

- С помощью второго параметра выполняется *указание шаблона документа*.
  - Присваивание значения `TRUE` третьему параметру означает, что при вызове внедренного объекта контейнером сервера для редактирования каждый раз запускается *новый экземпляр сервера*. В SDI-приложениях необходимо передавать значение `TRUE`, поскольку они одновременно могут управлять только одним документом. Всегда при запуске программы ExpServer (как автономной программы или как сервера OLE) запускается ее новый экземпляр. Таким образом, одновременно могут выполняться несколько копий программы.
4. Мастер Application Wizard при выбранной опции Full-server вызывает функции `EnableShellOpen` и `RegisterShellFileTypes` (см. параграф “Регистрация drw-файлов” гл. 12).

```
// Разрешить открытие файлов DDE
EnableShellOpen();
RegisterShellFileTypes(TRUE);
```

5. Для редактирования внедренного компонента необходимо, чтобы при запуске программы ExpServer как OLE-сервера, в тексте программы, сгенерированном мастером Application Wizard,

вызов статической функции `COleTemplateServer::RegisterAll` зарегистрировал сервер с библиотеками OLE (для этого в системе Windows 2000 необходимо работать с полномочиями для изменения реестра, например, администраторскими).

```
if (cmdInfo.m_bRunEmbedded || cmdInfo.m_bRunAutomated)
{
    // Регистрация приложений OLE-серверов как выполняющихся.
    // Это позволит библиотекам OLE создавать объекты
    // из других приложений
    COleTemplateServer::RegisterAll();

    // Не показывать главное окно приложения
    return TRUE;
}
```

6. При запуске программы `ExpServer` как сервера OLE, шаблон сервера и библиотеки OLE автоматически создают документ для обработки внедренных данных. В этом случае функция `InitInstance` завершается и обычная функция создания или открытия документа `ProcessShellCommand`, предназначенная для выполнения программы как автономного приложения, не вызывается.
7. Далее следует вызов `COleTemplateServer::UpdateRegistry`. Эта функция вызывает регистрацию сервера в системном реестре Windows. При регистрации сервер вводит в системный реестр информацию о программе и типе внедренного компонента, который он поддерживает.

```
// Приложение было запущено автономно или с другими ключами
// (например, /Register или /Regserver). Обновить записи
// в реестре, включая регистрацию типа документа.
else
{
    m_server.UpdateRegistry(OAT_INPLACE_SERVER);
}
```

8. Программа `ExpServer` при автономном выполнении регистрируется каждый раз при удалении системного реестра или изменении пути к папке с исполняемым файлом программы. Сервер (он регистрируется только один раз) должен быть зарегистрирован перед использованием для создания внедренного компонента путем выполнения одной из следующих операций:
  - Можно для вызова функции `UpdateRegistry` запустить программу-сервер как автономную программу.
  - Можно выполнить в программе Windows Explorer двойной щелчок на файле `ExpServer.reg`, размещенном в папке проекта. Этот файл создается мастером `Application Wizard` и содержит информацию для регистрации программы `ExpServer`. Двойной щелчок на имени файла запускает программу `Windows Registry Editor (RegEdit.exe)`, которая вводит в системный реестр эту информацию.
9. Мастер `Application Wizard` выполняет при выборе опции `Full-server` еще одно изменение функции `InitInstance` (не относящееся к OLE): `InitInstance` вызывает функцию `CWnd::DragAcceptFiles`, чтобы сделать доступным открытие файла с помощью метода “перетащить-и-отпустить” (см. параграф “Открытие файлов” гл. 12).

*Класс документа.* Класс `COleServerDoc` предоставляет не только обычные средства для управления документом при автономном выполнении программы, но и средства для управления документом при выполнении программы как сервера OLE при создании или редактировании внедренного компонента OLE (в этом случае документ содержит данные компонента). Класс документа `CexpServerDoc`

порождается от MFC-класса COleServerDoc, а не от CDocument. Кроме того, класс документа переопределяет виртуальную функцию COleServerDoc::OnGetEmbeddedItem, вызываемую при запуске программы как сервера OLE. Функция создает объект класса CExpServerSrvrItem, который будет рассмотрен ниже.

```
// Реализация сервера CExpServerDoc

COleServerItem* CExpServerDoc::OnGetEmbeddedItem()
{
    // OnGetEmbeddedItem вызывается средой для получения
    // метода COleServerItem, связанного с документом.
    // Вызывается только по необходимости.

    CExpServerSrvrItem* pItem = new CExpServerSrvrItem(this);
    ASSERT_VALID(pItem);
    return pItem;
}
```

*Класс компонента сервера.* Мастер Application Wizard при выборе опции Full-server создает новый класс CExpServerSrvrItem, наследуемый от MFC-класса COleServerItem. Новый класс предоставляет *дополнительные* средства обработки документа при выполнении программы как сервера OLE, предназначенные для создания и редактирования внедренного компонента. Как показано выше, в этом случае вызывается функция CExpServerDoc::OnGetEmbeddedItem, создающая объект класса CExpServerSrvrItem, связанный с объектом документа. Объект класса определен и реализован в собственном наборе файлов – SrvrItem.h и SrvrItem.cpp.

Когда программа-контейнер отображает внедренный объект в окне представления, вызывается функция CExpServerSrvrItem::OnDraw. Контейнер используется для работы с компонентом OLE в своем окне представления, если компонент *неактивен*, т. е. не редактируется сервером. Контейнер отображает его в собственном окне при редактировании в окне сервера в режиме полного открытия (при редактировании в режиме полного открытия компонент виден в обоих окнах: и сервера, и контейнера). Вместо отображения в окне контейнера функция CExpServerSrvrItem::OnDraw генерирует метафайл, хранящий команды, необходимые для отображения компонента. Этот файл *проигрывается* в окне представления контейнера. (*Метафайл* хранит текст и графику, записывая команды, которые выполнялись при их создании. При *проигрывании* метафайла эти команды выполняются, и выводимая информация отображается на указанном устройстве.) Мастер Application Wizard реализует функцию CExpServerSrvrItem::OnDraw частично. Специальный текст приложения будет добавлен позже.

Для отображения документа в окне представления при автономном выполнении программы вызывается функция класса *представления*. Функция OnDraw класса представления вызывается, когда *серверу* нужно отобразить внедренный компонент в своем окне представления, т.е. когда компонент *активен* (редактируется в любом режиме). Заметьте: при запуске сервера для редактирования компонента OLE сервер создает объект представления для отображения компонента, а также объекты приложения, документа и обрамляющего окна. Коды для отображения данных этих двух функций OnDraw на первый взгляд достаточно похожи (ниже мы рассмотрим некоторые их отличия). Функция CExpServerSrvrItem::OnGetExtent возвращает размер внедренного компонента OLE, когда программа-контейнер запрашивает размер. Стандартная реализация этой функции, которая просто возвращает закодированный размер (3000 на 3000 единиц HIMETRIC; единица HIMETRIC равна 0,01 миллиметра), генерируется мастером Application Wizard.

*Класс окна редактирования на месте.* Класс CInPlaceFrame, наследуемый от MFC-класса COleIPFrameWnd создается мастером Application Wizard. Когда программа ExpServer выполняется как автономная или как сервер в режиме полного открытия, окно представления совпадает со стандартным главным окном, управляемым, как обычно, классом CFrameWnd главного окна. Если



же программа ExpServer выполняется как сервер в режиме редактирования “на месте”, то окно представления ограничивается специальным обрамляющим окном редактирования, управляемым экземпляром класса CInPlaceFrame, который определен и реализован в файлах IpFrame.h и IpFrame.cpp. Мастер Application Wizard определяет в классе CInPlaceFrame переменную m\_wndResizeBar, являющуюся экземпляром MFC-класса COleResizeBar. Application Wizard реализует функцию CInPlaceFrame::OnCreate, которая, в свою очередь, вызывает функцию Create для объекта m\_wndResizeBar, чтобы задать границу обрамляющего окна редактирования “на месте”. При этом отображаются маркеры изменения размера. Для изменения размера внедренного компонента эти маркеры следует перетащить.

При редактировании внедренного компонента сервером отображается обрамляющее окно редактирования “на месте”. Программа-контейнер обычно отображает собственную границу вокруг выбранного неактивного внедренного компонента. Эта граница имеет маркеры изменения размера, позволяющие пользователю изменять размер компонента. Однако простая программа-контейнер, которую вы создадите позже, не позволяет изменять размер неактивного компонента (она такую границу не отображает).

*Класс представления.* Мастер Application Wizard добавляет в класс представления программы ExpServer функцию OnCanceledEditSrvr, которая получает управление после нажатия клавиши Esc при редактировании внедренного компонента OLE в режиме “на месте”. Она вызывает завершающую сеанс редактирования в этом режиме функцию COleServerDoc::OnDeactivateUI. Клавиша Esc определена с идентификатором ID\_CANCEL\_EDIT\_SRVR в таблице, загружаемой при редактировании “на месте”, т.е. в таблице акселераторов IDR\_SRVR\_INPLACE. Это пример акселератора, который не связан с командой меню. Мастер Application Wizard добавляет функцию OnCanceledEditSrvr в схему сообщений для ее вызова в ответ на использование акселератора.

```
// Поддержка сервера OLE

// Следующий обработчик команды предоставляет стандартный
// интерфейс клавиатуры для пользователя, давая возможность
// отменить обработку объекта на месте. В этом случае сервер
// (а не контейнер) вызывает деактивацию.
void CExpServerView::OnCanceledEditSrvr()
{
    GetDocument()->OnDeactivateUI(FALSE);
}
```

## Ресурсы

Файл Afxolesv.rc включается мастером Application Wizard в файл ресурсов программы. Файл Afxolesv.rc определяет несколько ресурсов, используемых MFC-классами сервера OLE. Мастер Application Wizard определяет отдельное меню и соответствующую таблицу акселераторов для каждого из трех режимов, в которых может выполняться программа ExpServer (ниже каждое из этих меню будет изменено, и будет описано назначение различных его пунктов):

- меню и таблица акселераторов IDR\_MAINFRAME используются при выполнении программы в автономном режиме;
- меню и таблица акселераторов IDR\_SRVR\_EMBEDDED – при выполнении программы как сервера OLE для редактирования в режиме *полного открытия*;
- меню и таблица акселераторов IDR\_SRVR\_INPLACE – при выполнении программы как сервера OLE в режиме *редактирования “на месте”*.

## Модификация кода приложения-сервера

Программа ExpServer предназначена для создания простых рисунков, содержащих сегменты прямых линий. Рисунок может создаваться как обычный документ, редактируемый в автономном режиме, и как внедренный компонент, редактируемый в программе контейнера. Чтобы добавить в программу средства рисования, выполните *те же* модификации и добавьте *тот же* код, который добавлялся в программу ScratchBook в гл. 10, 11 и 12. Для этого выполните действия, перечисленные в этих главах, учитывая следующие особенности программы ExpServer:

- Соответствующие имена классов и файлов следует заменить. Например, вместо модификации класса документа CScratchBookDoc в файле ScratchBookDoc.h измените класс документа CExpServerDoc в файле ExpServerDoc.h.
- Для экономии времени оставьте элементы меню File как есть (несколько элементов этого меню в гл. 10 были удалены, а в гл. 12 были добавлены).
- Можете оставить неизмененную версию меню Edit (с учетом того, что это меню в гл. 10 удалялось, а в гл. 11 была добавлена измененная его версия).
- Стандартное расширение файла (см. гл. 12) уже задано при генерации программы ExpServer в мастере Application Wizard (не нужно изменять строковый ресурс IDR\_MAINFRAME).
- Настройку значка программы (см. гл. 10), при желании, можно пропустить.
- Мастер Application Wizard уже добавил вызовы функций DragAcceptFiles, EnableShellOpen и RegisterShellFileTypes, поэтому в функцию InitInstance не нужно добавлять текст, описанный в гл. 12.

## OLE-поддержка

Добавим в программу несколько компонентов для поддержки OLE. Измените два специальных меню OLE. Сначала в редакторе меню Visual C++ откройте меню IDR\_SRVR\_EMBEDDED. Это меню отображается при выполнении программы ExpServer как сервера OLE в открытом режиме. Обратите внимание, что команды меню File отличаются от присвоенных мастером Application Wizard обычному меню программы, отображаемому при ее автономном выполнении (т. е. меню IDR\_MAINFRAME). Команды меню соответствуют ситуации, когда программа выполняется как сервер OLE. В этом случае она не открывает и не создает новые документы (только редактирует внедренный компонент):

- Обычные команды New и Open... в меню отсутствуют.
- Сервер возвращает измененные данные документа в программу-контейнер, а не сохраняет документ в файле на диске. Поэтому обычная команда Save заменена командой Update, выполняющей соответствующие операции копирования.
- Сервер позволяет создать отдельную копию внедренного компонента в файле на диске; эта задача выполняется командой Save Copy As... меню File (программа-сервер может открыть этот документ при выполнении в автономном режиме).
- При запуске сервера MFC изменяет текст заголовков команд Update и Exit для указания имени документа, содержащего внедренный компонент.

При модификации кода изменять меню File не нужно, однако меню Edit необходимо сделать *таким же*, как в меню IDR\_MAINFRAME программы в режиме автономного выполнения. Для этого удалите всплывающее меню IDR\_SRVR\_EMBEDDED. Затем, удерживая нажатой клавишу Ctrl, перетащите меню Edit из строки меню IDR\_MAINFRAME и опустите его на строку меню IDR\_SRVR\_EMBEDDED. Затем откройте меню IDR\_SRVR\_INPLACE в редакторе меню. Это меню отображается программой ExpServer при редактировании на месте внедренного компонента OLE и *не* имеет всплывающего

меню File, поскольку при редактировании внедренного компонента “на месте” за выполнение команд отвечает программа-контейнер, отображающая собственное меню File.

При редактировании на месте система *объединяет* меню сервера и контейнера, чтобы обе программы отображали соответствующие всплывающие меню. Результирующее объединенное меню содержит все всплывающие меню контейнера для редактирования на месте, расположенные перед двойным разделителем (в контейнере ExpContainer имеется единственное всплывающее меню File). Затем расположены все всплывающие меню сервера для редактирования на месте, расположенные перед двойным разделителем (в программе ExpServer это всплывающее меню Edit). Далее должны размещаться оставшиеся всплывающие меню контейнера, если они есть (в программе ExpContainer такие меню справа от двойного разделителя *отсутствуют*). Завершают объединение оставшиеся всплывающие меню сервера (для программы ExpServer это должно быть всплывающее меню Help).

Измените всплывающее меню Edit в меню IDR\_SRVR\_INPLACE, чтобы оно было таким же, как в меню IDR\_MAINFRAME и IDR\_SRVR\_EMBEDDED. Далее в файле ExpServerDoc.h добавьте определения для функций GetMaxX и GetMaxY в раздел public (определение класса CLine). Эти функции используются в добавляемой в класс документа новой функции GetDocSize.

```
class CLine : public CObject
{
public:
    int GetMaxX ()
    {
        return m_X1 > m_X2 ? m_X1 : m_X2;
    }
    int GetMaxY ()
    {
        return m_Y1 > m_Y2 ? m_Y1 : m_Y2;
    }

protected:
    int m_X1, int m_Y1, int m_X2, int m_Y2;
    CLine ()
    {}
    DECLARE_SERIAL (CLine)

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

Добавьте в файл ExpServerDoc.cpp определение функции GetDocSize в класс документа CExpServerDoc:

```
class CExpServerDoc : public COleServerDoc
{
protected:
    CTypedPtrArray<CObArray, CLine*> m_LineArray;
```

```

public:
    void AddLine (int X1, int Y1, int X2, int Y2);
    CSize GetDocSize ();
    CLine *GetLine (int Index);
    int GetNumLines ();

```

Добавьте определение метода GetDocSize в файл ExpServerDoc.cpp. Метод GetDocSize возвращает текущий размер рисунка, определяемый после проверки всех его линий и нахождения максимальных значений координат x и y конечных точек. Для получения текущего размера рисунка функция GetDocSize вызывается функцией CExpServerSrvrItem::OnDraw.

```

CSize CExpServerDoc::GetDocSize ()
{
    int X, Y, XMax = 1, YMax = 1;

    int Index = m_LineArray.GetSize ();
    while (Index--)
    {
        X = m_LineArray.GetAt (Index)->GetMaxX();
        XMax = X > XMax ? X : XMax;
        Y = m_LineArray.GetAt (Index)->GetMaxY();
        YMax = Y > YMax ? Y : YMax;
    }
    return CSize (XMax, YMax);
}

```

Внесите в класс представления необходимые изменения, чтобы функция ColeServerDoc::UpdateAllItems вызывалась при каждом изменении содержимого документа или размера окна представления. Если программа ExpServer редактирует внедренный компонент, то функция UpdateAllItems сообщает библиотекам OLE об изменении содержимого или размера внедренного компонента и заставляет OLE вызвать функцию OnDraw класса CExpServerSrvrItem, повторно создающую метафайл для отображения неактивного внедренного компонента в окне контейнера. При редактировании внедренного компонента в режиме полного открытия каждое его изменение будет сразу появляться в окне контейнера. Если же программа ExpServer выполняется в автономном режиме, то при вызове UpdateAllItems ничего не происходит. Все изменения размера или содержимого документа сопровождаются вызовом функции OnLButtonUp или OnDraw класса представления; следовательно, в файле ExpServerView.cpp нужно добавить вызов функции UpdateAllItems только в две названные функции. Сначала вызов UpdateAllItems добавьте в функцию OnLButtonUp.

```

void CExpServerView::OnLButtonUp(UINT nFlags, CPoint point)
{
    // TODO: Добавьте сюда собственный код обработчика
    // или вызов стандартного

    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);

        CExpServerDoc* PDoc = GetDocument ();
        PDoc->AddLine (m_PointOrigin.x, m_PointOrigin.y,
            point.x, point.y);
        PDoc->UpdateAllItems (0);
    }

    CView::OnLButtonUp(nFlags, point);
}

```

Далее добавьте вызов функции CExpServerView::UpdateAllItems в обработчик OnDraw.

```

// Отображение класса CExpServerView

void CExpServerView::OnDraw(CDC* pDC)
{
    CExpServerDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    // TODO: добавьте сюда код отображения собственных данных

    int Index = pDoc->GetNumLines ();
    while (Index--) pDoc->GetLine (Index)->Draw (pDC);
    pDoc->UpdateAllItems (0);
}

```

В функцию OnDraw класса CExpServerSrvrItem в файле SrvrItem.cpp необходимо внести следующие дополнения (имеющийся в тексте программы вызов функции SetWindowExt *переопределяет* функцию SetWindowExt, сгенерированную мастером Application Wizard).

```

BOOL CExpServerSrvrItem::OnDraw(CDC* pDC, CSize& rSize)
{
    // Удалите этот вызов, если используется rSize
    UNREFERENCED_PARAMETER(rSize);

    CExpServerDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    // TODO: установите единицы измерения и размер
    // (обычно размер равен возвращаемому функцией OnGetExtent)
    pDC->SetMapMode(MM_ANISOTROPIC);
    pDC->SetWindowOrg(0,0);
    pDC->SetWindowExt(3000, 3000);

    // TODO: Добавьте сюда код рисования. Можно использовать
    // размер HIMETRIC. Все операции помещаются в метафайл контекста
    // устройства (pDC).
    pDC->SetWindowExt (pDoc->GetDocSize ());

    int Index = pDoc->GetNumLines ();
    while (Index--) pDoc->GetLine (Index)->Draw (pDC);

    return TRUE;
}

```

Функция `CExpServerSrvrItem::OnDraw`, как было описано ранее, создает метафайл для отображения неактивного внедренного компонента в окне контейнера. Текст программы, сгенерированный мастером `Application Wizard`, начинается с установки режима отображения равным `MM_ANISOTROPIC`. Эта установка обеспечивает выполнение масштабирования внедренного компонента, чтобы он помещался внутри окна контейнера. (Режимы отображения рассматривались в гл. 19). Затем вызывается функция `CDC::SetWindowExt`. Функция `SetWindowExt` задает общий горизонтальный и вертикальный размер в логических единицах изображения, отображаемого метафайлом. Функция `SetWindowExt`, сгенерированная мастером `Application Wizard`, задает условный размер 3000 на 3000 логических единиц. Однако в эту функцию необходимо передать действительный логический размер внедренного компонента (получаемый при вызове новой функции `CexpServerDoc::GetDocSize`). В результате рисунок полностью помещается внутри границ внедренного компонента в окне контейнера. Оставшийся добавленный текст программы такой же, как и для выполнения рисования в функции `OnDraw` класса представления.

Внеся описанные модификации в код программы `ExpServer`, можно выполнить ее в автономном режиме, и она будет работать как третья версия программы `ScratchBook` (см. гл. 12). Если у вас есть программа-контейнер `OLE`, то можно выполнить программу `ExpServer` как сервер в режиме “на месте” или в режиме полного открытия, используя команды, описанные ранее (в параграфе “Внедрение, связывание и автоматизация”). Программа-контейнер, которую можно использовать для проверки `ExpServer` будет описана в параграфе “Создание программы-контейнера”.

## Текст программы *ExpServer*

Тексты программы `ExpServer` приведены в листингах 24.1—24.12.

---

### Листинг 24.1.

```
// ExpServer.h : главный файл заголовков приложения ExpServer
//

#pragma once

#ifndef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"          // основные символы

// CExpServerApp:
// Реализацию этого класса смотрите в файле ExpServer.cpp
//

class CExpServerApp : public CWinApp
{
public:
    CExpServerApp();

    // Переопределения
public:
    virtual BOOL InitInstance();

    // Реализация
    COleTemplateServer m_server;
    // Объект сервера для создания документов
```

```

        afx_msg void OnAppAbout();
        DECLARE_MESSAGE_MAP()
    };

extern CExpServerApp theApp;

```

---

## Листинг 24.2.

```

// ExpServer.cpp : Определяет поведение классов приложения
//

#include "stdafx.h"
#include "ExpServer.h"
#include "MainFrm.h"

#include "IpFrame.h"
#include "ExpServerDoc.h"
#include "ExpServerView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CExpServerApp

BEGIN_MESSAGE_MAP(CExpServerApp, CWinApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
    // Стандартные команды работы с файлами документов
    ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
    // Стандартная команда печати
    ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()

// Конструктор класса CExpServerApp

CExpServerApp::CExpServerApp()
{
    // TODO: поместите сюда собственный код конструктора
    // Поместите все существенные процедуры инициализации
    // в функцию InitInstance
}

// Единственный объект класса CExpServerApp

CExpServerApp theApp;
// Этот идентификатор был сгенерирован статистически
// уникальным для вашего приложения. Можете изменить его,
// если сочтете нужным

// {0C60B0A4-27D5-4CE5-AE64-F6AA0F128C0B}
static const CLSID clsid =
{ 0xC60B0A4, 0x27D5, 0x4CE5,
  { 0xAE, 0x64, 0xF6, 0xAA, 0xF, 0x12, 0x8C, 0xB }
};

// Инициализация CExpServerApp

```

```

BOOL CExpServerApp::InitInstance()
{
    CWinApp::InitInstance();

    // Инициализация библиотек OLE
    if (!AfxOleInit())
    {
        AfxMessageBox(IDP_OLE_INIT_FAILED);
        return FALSE;
    }

    // Стандартная инициализация.
    // Если вы не используете какие-то из возможностей и хотите
    // уменьшить размер окончного исполняемого модуля, удалите
    // отдельные процедуры инициализации из последующего кода.
    // Измените строку-аргумент функции (ключ в реестре,
    // под которым хранятся ваши установки).
    // TODO: измените эту строку на что-нибудь подходящее,
    // например, название вашей компании или организации
    SetRegistryKey(_T("Local AppWizard-Generated Applications"));
    LoadStdProfileSettings(4); // Загрузка установок из INI-файла
                               // (включая MRU)

    // Регистрация шаблонов документов приложения. Шаблоны
    // документов служат связью между документами, окнами документов
    // и окнами представлений
    CSingleDocTemplate* pDocTemplate;
    pDocTemplate = new CSingleDocTemplate(
        IDR_MAINFRAME,
        RUNTIME_CLASS(CExpServerDoc),
        RUNTIME_CLASS(CMainFrame), // главное окно
                                   // SDI-приложения
        RUNTIME_CLASS(CExpServerView));
    pDocTemplate->SetServerInfo(
        IDR_SRVR_EMBEDDED, IDR_SRVR_INPLACE,
        RUNTIME_CLASS(CInPlaceFrame));
    AddDocTemplate(pDocTemplate);
    // Связь COleTemplateServer с шаблоном документа
    // COleTemplateServer создает новые документы от имени
    // запрашивающих приложений, используя информацию,
    // указанную в шаблоне документа
    m_server.ConnectTemplate(clsid, pDocTemplate, TRUE);
        // SDI-приложения регистрируют объекты, только если
        // ключи /Embedding или /Automation присутствуют
        // в командной строке
    // Разрешить открытие файлов DDE
    EnableShellOpen();
    RegisterShellFileTypes(TRUE);
    // Поиск в командной строке команд управления, DDE,
    // открытия файлов
    CCommandLineInfo cmdInfo;
    ParseCommandLine(cmdInfo);
    // Приложение запущено с ключом /Embedding или /Automation.
    // Run app as automation server.
    if (cmdInfo.m_bRunEmbedded || cmdInfo.m_bRunAutomated)
    {
        // Регистрация приложений OLE-серверов как выполняющихся.
    }
}

```



```

        // Это позволит библиотекам OLE создавать объекты из
        // других приложений
        COleTemplateServer::RegisterAll();

        // Не показывать главное окно приложения
        return TRUE;

    }

    // Приложение запущено с ключом /Unregserver или /Unregister.
    // Отменить регистрацию типа документа. Другие части процесса
    // отмены регистрации происходят в ProcessShellCommand().
    else if (cmdInfo.m_nShellCommand ==
             CCommandLineInfo::AppUnregister)
    {
        UnregisterShellFileTypes();
        m_server.UpdateRegistry(OAT_INPLACE_SERVER, NULL,
                               NULL, FALSE);
    }

    // Приложение было запущено автономно или с другими ключами
    // (например, /Register или /Regserver). Обновить записи
    // в реестре, включая регистрацию типа документа.
    else
    {
        m_server.UpdateRegistry(OAT_INPLACE_SERVER);
    }

    // Выполнение команд, указанных в командной строке. Вернет
    // FALSE, если приложение запускалось с /RegServer, /Register,
    // /Unregserver или /Unregister.
    if (!ProcessShellCommand(cmdInfo))
        return FALSE;

    // Прорисовка и обновление единственного
    // проинициализированного окна
    m_pMainWnd->ShowWindow(SW_SHOW);
    m_pMainWnd->UpdateWindow();
    m_pMainWnd->DragAcceptFiles();
    return TRUE;
}

// CAboutDlg диалог, используемый в App About

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

    // Данные для диалога
    enum { IDD = IDD_ABOUTBOX };

protected:
    virtual void DoDataExchange(CDataExchange* pDX);
    // поддержка DDX/DDV

    // Реализация
protected:
    DECLARE_MESSAGE_MAP()
};

```

```

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
    !
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
END_MESSAGE_MAP()

// Команда приложения на запуск диалога
void CExpServerApp::OnAppAbout()
{
    CAboutDlg aboutDlg;
    aboutDlg.DoModal();
}

// Обработчики сообщений класса CExpServerApp

```

---

### Листинг 24.3.

```

// ExpServerDoc.h : интерфейс класса CExpServerDoc
//

#pragma once

class CLine : public CObject
{
public:
    int GetMaxX ()
    {
        return m_X1 > m_X2 ? m_X1 : m_X2;
    }
    int GetMaxY ()
    {
        return m_Y1 > m_Y2 ? m_Y1 : m_Y2;
    }
protected:
    int m_X1, int m_Y1, int m_X2, int m_Y2;
    CLine ()
    {}
    DECLARE_SERIAL (CLine)

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

```

```

class CExpServerSrvrItem;

class CExpServerDoc : public COleServerDoc
{
protected:
    CTypedPtrArray<COleArray, CLine*> m_LineArray;

public:
    void AddLine (int X1, int Y1, int X2, int Y2);
    CSize GetDocSize ();
    CLine *GetLine (int Index);
    int GetNumLines ();

protected: // используется только для сериализации
    CExpServerDoc();
    DECLARE_DYNCREATE(CExpServerDoc)

// Атрибуты
public:
    CExpServerSrvrItem* GetEmbeddedItem()
    { return reinterpret_cast<CExpServerSrvrItem*>
      (COleServerDoc::GetEmbeddedItem()); }

// Операции
public:

// Переопределения
protected:
    virtual COleServerItem* OnGetEmbeddedItem();
public:
    virtual BOOL OnNewDocument();
    virtual void Serialize(CArchive& ar);

// Реализация
public:
    virtual ~CExpServerDoc();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Сгенерированные функции карты сообщений
protected:
    DECLARE_MESSAGE_MAP()
public:
    afx_msg void OnEditUndo();
    afx_msg void OnUpdateEditUndo(CCmdUI *pCmdUI);
    afx_msg void OnEditClearAll();
    afx_msg void OnUpdateEditClearAll(CCmdUI *pCmdUI);
    virtual void DeleteContents();
};

```

---

#### Листинг 24.4.

```
// ExpServerDoc.cpp : реализация класса CExpServerDoc
//

#include "stdafx.h"
#include "ExpServer.h"

#include "ExpServerDoc.h"
#include "SrvrItem.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CExpServerDoc

IMPLEMENT_DYNCREATE(CExpServerDoc, COleServerDoc)

BEGIN_MESSAGE_MAP(CExpServerDoc, COleServerDoc)
    ON_COMMAND(ID_EDIT_UNDO, OnEditUndo)
    ON_UPDATE_COMMAND_UI(ID_EDIT_UNDO, OnUpdateEditUndo)
    ON_COMMAND(ID_EDIT_CLEAR_ALL, OnEditClearAll)
    ON_UPDATE_COMMAND_UI(ID_EDIT_CLEAR_ALL, OnUpdateEditClearAll)
END_MESSAGE_MAP()

// Конструктор/деструктор класса CExpServerDoc

CExpServerDoc::CExpServerDoc()
{
    // Использование составных файлов OLE
    EnableCompoundFile();

    // TODO: добавьте сюда одноразовый код конструктора
}

CExpServerDoc::~CExpServerDoc()
{
}

BOOL CExpServerDoc::OnNewDocument()
{
    if (!COleServerDoc::OnNewDocument())
        return FALSE;

    // TODO: добавьте сюда код реинициализации
    // (SDI-документы будут повторно использовать
    // этот документ)

    return TRUE;
}

// Реализация сервера CExpServerDoc

COleServerItem* CExpServerDoc::OnGetEmbeddedItem()
```

```

{
    // OnGetEmbeddedItem вызывается средой для получения
    // метода COleServerItem, связанного с документом.
    // Вызывается только по необходимости.

    CExpServerSrvrItem* pItem = new CExpServerSrvrItem(this);
    ASSERT_VALID(pItem);
    return pItem;
}

// Диагностика CExpServerDoc

#ifdef _DEBUG
void CExpServerDoc::AssertValid() const
{
    COleServerDoc::AssertValid();
}

void CExpServerDoc::Dump(CDumpContext& dc) const
{
    COleServerDoc::Dump(dc);
}
#endif //_DEBUG

// Команды CExpServerDoc

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

void CLine::Draw (CDC *PDC)
{
    PDC->MoveTo (m_X1, m_Y1);
    PDC->LineTo (m_X2, m_Y2);
}

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

CLine *CExpServerDoc::GetLine (int Index)
{
    if (Index < 0 || Index >

```

```

        m_LineArray.GetUpperBound ()) return 0;
    return m_LineArray.GetAt (Index);
}

int CExpServerDoc::GetNumLines ()
{
    return m_LineArray.GetSize ();
}

void CExpServerDoc::DeleteContents()
{
    // TODO: Добавьте сюда собственный код или вызов
    // базового класса

    int Index = m_LineArray.GetSize ();
    while (Index-->0) delete m_LineArray.GetAt (Index);
    m_LineArray.RemoveAll ();

    CDocument::DeleteContents();
}

void CExpServerDoc::OnEditClearAll()
{
    // TODO: Добавьте сюда собственный код обработчика

    DeleteContents();
    UpdateAllViews (0);
    SetModifiedFlag ();
}

void CExpServerDoc::OnUpdateEditClearAll(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика

    pCmdUI->Enable (m_LineArray.GetSize ());
}

void CExpServerDoc::OnEditUndo()
{
    // TODO: Добавьте сюда собственный код обработчика

    int Index = m_LineArray.GetUpperBound ();
    if (Index > -1)
    {
        delete m_LineArray.GetAt (Index);
        m_LineArray.RemoveAt (Index);
    }
    UpdateAllViews (0);
    SetModifiedFlag ();
}

void CExpServerDoc::OnUpdateEditUndo(CCmdUI *pCmdUI)
{
    // TODO: Добавьте сюда собственный код обработчика

    pCmdUI->Enable (m_LineArray.GetSize ());
}

```

```

IMPLEMENT_SERIAL (CLine, CObject, 1)

void CLine::Serialize (CArchive &ar)
{
    if (ar.IsStoring()) ar << m_X1 << m_Y1 << m_X2 << m_Y2;
    else ar >> m_X1 >> m_Y1 >> m_X2 >> m_Y2;
}

CSize CExpServerDoc::GetDocSize ()
{
    int X, Y, XMax = 1, YMax = 1;

    int Index = m_LineArray.GetSize ();
    while (Index--)
    {
        X = m_LineArray.GetAt (Index)->GetMaxX();
        XMax = X > XMax ? X : XMax;
        Y = m_LineArray.GetAt (Index)->GetMaxY();
        YMax = Y > YMax ? Y : YMax;
    }
    return CSize (XMax, YMax);
}

```

---

#### Листинг 24.5.

```

// ExpServerView.h : интерфейс класса CExpServerView
//

#pragma once

class CExpServerView : public CView
{
protected:
    CString m_ClassName;
    int m_Dragging;
    HCURSOR m_HCross;
    CPoint m_PointOld;
    CPoint m_PointOrigin;

protected: // используется только для сериализации
    CExpServerView();
    DECLARE_DYNCREATE(CExpServerView)

// Атрибуты
public:
    CExpServerDoc* GetDocument() const;

// Операции
public:

// Переопределения
public:
    virtual void OnDraw(CDC* pDC);
    // переопределена для прорисовки этого вида

```

```

virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
protected:
    virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
    virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
    virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);

// Реализация
public:
    virtual ~CExpServerView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Сгенерированные функции карты сообщений
protected:
    afx_msg void OnCancelEditSrvr();
    DECLARE_MESSAGE_MAP()
public:
    afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
    afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
    afx_msg void OnMouseMove(UINT nFlags, CPoint point);

};

#ifdef _DEBUG // отладочная версия в файле ExpServerView.cpp
inline CExpServerDoc* CExpServerView::GetDocument() const
    { return reinterpret_cast<CExpServerDoc*>(m_pDocument); }
#endif

```

---

#### Листинг 24.6.

```

// ExpServerView.cpp : реализация класса CExpServerView
//

#include "stdafx.h"
#include "ExpServer.h"

#include "ExpServerDoc.h"
#include "ExpServerView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CExpServerView

IMPLEMENT_DYNCREATE(CExpServerView, CView)

BEGIN_MESSAGE_MAP(CExpServerView, CView)
    // Стандартные команды печати
    ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
    ON_COMMAND(ID_CANCEL_EDIT_SRVR, OnCancelEditSrvr)
    ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)

```



```

        ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
        ON_WM_LBUTTONDOWN()
        ON_WM_LBUTTONUP()
        ON_WM_MOUSEMOVE()
    END_MESSAGE_MAP()

    // Конструктор/деструктор класса CExpServerView

    CExpServerView::CExpServerView()
    {
        // TODO: добавьте сюда собственный код конструктора

        m_Dragging = 0;
        m_HCross = AfxGetApp()->LoadStandardCursor (IDC_CROSS);
    }

    CExpServerView::~CExpServerView()
    {
    }

    // Отображение класса CExpServerView

    void CExpServerView::OnDraw(CDC* pDC)
    {
        CExpServerDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);

        // TODO: добавьте сюда код отображения собственных данных

        int Index = pDoc->GetNumLines ();
        while (Index-->0) pDoc->GetLine (Index)->Draw (pDC);
        pDoc->UpdateAllItems (0);
    }

    // Печать CExpServerView

    BOOL CExpServerView::OnPreparePrinting(CPrintInfo* pInfo)
    {
        // подготовка по умолчанию
        return DoPreparePrinting(pInfo);
    }

    void CExpServerView::OnBeginPrinting(CDC* /*pDC*/,
        CPrintInfo* /*pInfo*/)
    {
        // TODO: добавьте сюда дополнительную
        // инициализацию перед печатью
    }

    void CExpServerView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
    {
        // TODO: добавьте очистку после печати
    }

```

```

// Поддержка сервера OLE

// Следующий обработчик команды предоставляет стандартный интерфейс
// клавиатуры для пользователя, давая возможность отменить обработку
// объекта на месте. В этом случае сервер (а не контейнер) вызывает
// деактивацию.
void CExpServerView::OnCancelEditSrvr()
{
    GetDocument()->OnDeactivateUI(FALSE);
}

// Диагностика CExpServerView

#ifdef _DEBUG
void CExpServerView::AssertValid() const
{
    CView::AssertValid();
}

void CExpServerView::Dump(CDumpContext& dc) const
{
    CView::Dump(dc);
}

CExpServerDoc* CExpServerView::GetDocument() const
// не отладочная версия встроена
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CExpServerDoc)));
    return (CExpServerDoc*)m_pDocument;
}
#endif // _DEBUG

// Обработчики сообщений класса CExpServerView

void CExpServerView::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: Добавьте сюда собственный код обработчика
    // или вызов базового класса

    m_PointOrigin = point;
    m_PointOld = point;
    SetCapture ();
    m_Dragging = 1;
    RECT Rect;
    GetClientRect (&Rect);
    ClientToScreen (&Rect);
    ::ClipCursor (&Rect);

    CView::OnLButtonDown(nFlags, point);
}

void CExpServerView::OnLButtonUp(UINT nFlags, CPoint point)
{
    // TODO: Добавьте сюда собственный код обработчика
    // или вызов стандартного

```

```

        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);

            CExpServerDoc* PDoc = GetDocument ();
            PDoc->AddLine (m_PointOrigin.x, m_PointOrigin.y,
                          point.x, point.y);
            PDoc->UpdateAllItems (0);
        }

        CView::OnLButtonUp(nFlags, point);
    }

void CExpServerView::OnMouseMove(UINT nFlags, CPoint point)
{
    // TODO: Добавьте сюда собственный код обработчика
    // или вызов стандартного

    ::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);
}

BOOL CExpServerView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: добавьте сюда собственный код или вызов
    // базового класса

    m_ClassName = AfxRegisterWndClass
        (CS_HREDRAW | CS_VREDRAW, 0,
        (HBRUSH)::GetStockObject (WHITE_BRUSH), 0);
    cs.lpszClass = m_ClassName;

    return CView::PreCreateWindow(cs);
}

```

#### Листинг 24.7.

```
// MainFrm.h : интерфейс класса CMainFrame
//

#pragma once
class CMainFrame : public CFrameWnd
{

protected: // используется только для сериализации
    CMainFrame();
    DECLARE_DYNCREATE(CMainFrame)

// Атрибуты
public:

// Операции
public:

// Переопределения
public:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);

// Реализация
public:
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

// Сгенерированные функции карты сообщений
protected:
    DECLARE_MESSAGE_MAP()
};
```

---

#### Листинг 24.8.

```
// MainFrm.cpp : реализация класса CMainFrame
//

#include "stdafx.h"
#include "ExpServer.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
END_MESSAGE_MAP()
```

```

// Конструктор/деструктор класса CMainFrame

CMainFrame::CMainFrame()
{
    // TODO: добавьте сюда код инициализации
    // элементов класса
}

CMainFrame::~CMainFrame()
{
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    if( !CFrameWnd::PreCreateWindow(cs) )
        return FALSE;
    // TODO: Измените класс или стили окна здесь,
    // изменяя и добавляя поля структуры cs

    return TRUE;
}

// Диагностика класса CMainFrame

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
    CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
    CFrameWnd::Dump(dc);
}

#endif //_DEBUG

// Обработчики сообщений класса CMainFrame

```

---

## Листинг 24.9.

```

// IpFrame.h : интерфейс класса CInPlaceFrame
//

#pragma once

class CInPlaceFrame : public COleIPFrameWnd
{
    DECLARE_DYNCREATE(CInPlaceFrame)
public:
    CInPlaceFrame();

    // Атрибуты
public:

    // Операции

```

```

public:
    // Переопределения
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);

    // Реализация
public:
    virtual ~CInPlaceFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:
    CObjectDropTarget      m_dropTarget;
    CObjectResizeBar      m_wndResizeBar;

    // Сгенерированные функции карты сообщений
protected:
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    DECLARE_MESSAGE_MAP()
};

```

---

#### Листинг 24.10.

```

// IpFrame.cpp : реализация класса CInPlaceFrame
//

#include "stdafx.h"
#include "ExpServer.h"

#include "IpFrame.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CInPlaceFrame

IMPLEMENT_DYNCREATE(CInPlaceFrame, COleIPFrameWnd)

BEGIN_MESSAGE_MAP(CInPlaceFrame, COleIPFrameWnd)
    ON_WM_CREATE()
END_MESSAGE_MAP()

// Конструктор/деструктор класса CInPlaceFrame

CInPlaceFrame::CInPlaceFrame()
{
}

CInPlaceFrame::~CInPlaceFrame()
{
}

int CInPlaceFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)

```

```

{
    if (ColeIPFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    // CResizeBar реализует быстрое изменение размеров.
    if (!m_wndResizeBar.Create(this))
    {
        TRACE0("Failed to create resize bar\n");
        return -1;        // не удалось создать элемент
    }

    // Зарегистрируйте цель операции "drag-and-drop",
    // при которой обрамляющее окно неизменно.
    m_dropTarget.Register(this);

    return 0;
}

BOOL CInPlaceFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Измените класс или стили окна здесь,
    // изменяя и добавляя поля структуры cs

    return ColeIPFrameWnd::PreCreateWindow(cs);
}

// Диагностика класса CInPlaceFrame

#ifdef _DEBUG
void CInPlaceFrame::AssertValid() const
{
    ColeIPFrameWnd::AssertValid();
}

void CInPlaceFrame::Dump(CDumpContext& dc) const
{
    ColeIPFrameWnd::Dump(dc);
}
#endif // _DEBUG

// Команды класса CInPlaceFrame

```

---

#### Листинг 24.11.

```

// SrvrItem.h : интерфейс класса CExpServerSrvrItem
//

#pragma once

class CExpServerSrvrItem : public COleServerItem
{
    DECLARE_DYNAMIC(CExpServerSrvrItem)

    // Конструкторы
public:
    CExpServerSrvrItem(CExpServerDoc* pContainerDoc);

```

```

// Атрибуты
    CExpServerDoc* GetDocument() const
    { return reinterpret_cast<CExpServerDoc*>
      (CServerItem::GetDocument()); }

// Переопределения
public:
    virtual BOOL OnDraw(CDC* pDC, CSize& rSize);
    virtual BOOL OnGetExtent(DVASPECT dwDrawAspect, CSize& rSize);

// Реализация
public:
    ~CExpServerSrvrItem();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:
    virtual void Serialize(CArchive& ar);
    // переопределено для организации ввода/вывода
};

```

---

#### Листинг 24.12.

```

// SrvrItem.cpp : реализация класса CExpServerSrvrItem
//

#include "stdafx.h"
#include "ExpServer.h"

#include "ExpServerDoc.h"
#include "SrvrItem.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// Реализация CExpServerSrvrItem

IMPLEMENT_DYNAMIC(CExpServerSrvrItem, CServerItem)

CExpServerSrvrItem::CExpServerSrvrItem(CExpServerDoc* pContainerDoc)
    : CServerItem(pContainerDoc, TRUE)
{
    // TODO: добавьте сюда одноразовый код конструктора
    // (укажите дополнительные форматы для буфера обмена)
}

CExpServerSrvrItem::~CExpServerSrvrItem()
{
    // TODO: добавьте сюда код очистки
}

void CExpServerSrvrItem::Serialize(CArchive& ar)
{

```



```

    // CExpServerSrvrItem::Serialize вызывается средой для
    // копирования объекта в буфер. Это происходит автоматически
    // через вызов OLE OnGetClipboardData. Хорошо, если
    // встроенный объект поддается функции Serialize.

    if (!IsLinkedItem())
    {
        CExpServerDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);
        pDoc->Serialize(ar);
    }
}

BOOL CExpServerSrvrItem::OnGetExtent(DVASPECT dwDrawAspect, CSize& rSize)
{
    if (dwDrawAspect != DVASPECT_CONTENT)
        return COleServerItem::OnGetExtent(dwDrawAspect, rSize);

    // CExpServerSrvrItem::OnGetExtent вызывается для получения
    // размеров объекта. Реализация по умолчанию возвращает
    // запомненное значение.

    CExpServerDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    // TODO: используем размер по умолчанию

    rSize = CSize(3000, 3000);    // 3000 x 3000 единиц HIMETRIC

    return TRUE;
}

BOOL CExpServerSrvrItem::OnDraw(CDC* pDC, CSize& rSize)
{
    // Удалите этот вызов, если используется rSize
    UNREFERENCED_PARAMETER(rSize);

    CExpServerDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    // TODO: установите единицы измерения и размер
    // (обычно размер равен возвращаемому функцией OnGetExtent)
    pDC->SetMapMode(MM_ANISOTROPIC);
    pDC->SetWindowOrg(0,0);
    pDC->SetWindowExt(3000, 3000);

    // TODO: Добавьте сюда код рисования. Можно использовать
    // размер HIMETRIC.
    // Все операции помещаются в метафайл контекста
    // устройства (pDC).
    pDC->SetWindowExt (pDoc->GetDocSize ());

    int Index = pDoc->GetNumLines ();
    while (Index-->0) pDoc->GetLine (Index)->Draw (pDC);

    return TRUE;
}

```

```
// Диагностика класса CExpServerSrvrItem

#ifdef _DEBUG
void CExpServerSrvrItem::AssertValid() const
{
    COleServerItem::AssertValid();
}

void CExpServerSrvrItem::Dump(CDumpContext& dc) const
{
    COleServerItem::Dump(dc);
}
#endif
```

## Программа-контейнер ExpContainer

Создадим простую программу-контейнер OLE с помощью мастера Application Wizard. В приложение не нужно добавлять специальный код, поскольку программа, сгенерированная мастером, подходит для проверки созданного сервера OLE. Присвоим новой программе имя ExpContainer. На вкладках окна Application Wizard выберите такие же опции, как и при создании программы WinHello (см. гл. 9). Одно *исключение* – на вкладке Compound Document Support выберите опцию Container, а остальные опции оставьте неизменными. Рассмотрим основные изменения, внесенные мастером Application Wizard в обычный текст программы, что, собственно, и делает программу ExpContainer контейнером OLE. Опишем некоторые средства, добавляемые в полнофункциональное приложение-контейнер, и работу программы.

### Классы

*Класс приложения.* Мастер Application Wizard добавляет вызов функции AfxOleInit в функцию InitInstance класса приложения ExpContainer, чтобы инициализировать библиотеки OLE (как и в приложении-сервере). Мастер Application Wizard добавляет в функцию InitInstance после создания шаблона документа вызов функции CSingleDocTemplate::SetContainerInfo. Вызов функции SetContainerInfo задает идентификатор меню, который вместе с соответствующим ресурсом акселератора отображается программой ExpContainer при редактировании внедренного компонента “на месте”.

```
pDocTemplate->SetContainerInfo(IDR_CNTR_INPLACE);
```

Это меню *объединено* с меню, отображаемым программой-сервером при редактировании на месте. Как отмечалось ранее, функция SetContainerInfo похожа на вызываемую для сервера функцию SetServerInfo.

*Класс документа.* Класс COleDocument содержит основные средства обработки для управления документами в контейнере OLE или программах-серверах. Класс документа ExpContainer порождается от MFC-класса COleDocument, а не от обычного класса CDocument. Вспомните: класс документа сервера порождается от класса COleServerDoc, порожденного от COleDocument. Объект документа в контейнере OLE управляет сохранением внедренного компонента или компонентов, а также собственно данными документа (например, текстом в программе текстового процессора). Однако программа ExpContainer сохраняет только внедренные компоненты, а не сами данные. Как показано ниже, данные для каждого внедренного компонента сохраняются в объекте класса CExpContainerCntrItem, наследуемого от COleClientItem и управляемого классом документа. При использовании метода Serialize класса документа мастер Application Wizard в конце добавляет

вызов функции `COleDocument::Serialize` для сериализации внедренных компонентов, сохраненных в документе. Для каждого объекта класса `CExpContainerCntrItem`, сохраняющего внедренный компонент, вызывается функция `Serialize`.

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

        // Вызов базового класса COleDocument для сериализации
        // объектов класса COleClientItem документа контейнера
        COleDocument::Serialize(ar);
    }
}
```

*Класс компонента контейнера.* Новый класс `CExpContainerCntrItem`, порождаемый от MFC-класса `COleClientItem` и реализованный в файлах `CntrlItem.h` и `CntrlItem.cpp`, создается мастером `Application Wizard`. Когда новый компонент OLE внедряется в программу, класс представления, как описано ниже, создает объект этого класса для хранения внедренного компонента и управления им. Каждый объект класса `CExpContainerCntrItem` связан с объектом документа, поддерживающего список таких объектов. Мастер `Application Wizard` переопределяет виртуальные функции класса `COleClientItem`. Далее рассмотрены виртуальные функции, сгенерированные мастером `Application Wizard`, и выполняющие отличные от функций базового класса действия:

- Когда сервер изменяет компонент в режиме редактирования “на месте” или в режиме полного открытия, управление получает виртуальная функция `CExpContainerCntrItem::OnChange`. Версия этой функции, реализованная мастером `Application Wizard`, вызывает версию этой же функции в базовом классе, а затем – функцию `CDocument::UpdateAllViews` (см. гл. 13), после чего компонент перерисовывается функцией `OnDraw` класса представления.

```
void CExpContainerCntrItem::OnChange
(OLE_NOTIFICATION nCode, DWORD dwParam)
{
    ASSERT_VALID(this);

    COleClientItem::OnChange(nCode, dwParam);

    // Когда компонент редактируется (на месте или в режиме
    // полного открытия), он передает сообщение функции
    // OnChange для изменения состояния компонента или
    // внешнего вида его содержимого

    // TODO: сделайте компонент недействительным,
    // вызывая метод UpdateAllViews
    // (с указаниями, требуемыми для вашего приложения)

    GetDocument()->UpdateAllViews(NULL);
    // сейчас просто обновите все представления/без указаний
}
```

- Чтобы получить размер и позицию компонента (когда внедренный компонент редактируется на месте) OLE вызывает виртуальную функцию `CExpContainerCntrItem::OnGetItemPosition`. Функция возвращает эти значения в единицах устройства относительно области окна представления контейнера. Реализация этой функции, сгенерированная мастером Application Wizard, возвращает константу. Поскольку версия функции `OnGetItemPosition` в базовом классе ничего не выполняет, программа-контейнер *должна* реализовать необходимые операции в своей версии этой функции.

```
void CExpContainerCntrItem::OnGetItemPosition(Crect& rPosition)
{
    ASSERT_VALID(this);

    // При активизации на месте вызывается функция
    // CExpContainerCntrItem::OnGetItemPosition, чтобы определить
    // размещение компонента. Стандартная реализация, созданная
    // мастером Application Wizard, просто возвращает
    // закодированный прямоугольник, отражающий текущее
    // положение компонента относительно окна представления.
    // Также для этого можно
    // вызвать функцию CExpContainerCntrItem::GetActiveView

    // TODO: возвратите правильный прямоугольник
    // (в пикселях) в rPosition

    rPosition.SetRect(10, 10, 210, 210);
}
```

- Когда пользователь программы ExpContainer, нажимая клавишу Esc, выводит внедренный компонент из активного состояния, вызывается функция `OnDeactivateUI`. Реализация функции, сгенерированная мастером Application Wizard, вызывает версию этой функции из базового класса, а затем для деактивации внедренного компонента и освобождения используемых им ресурсов вызывает функцию `COleClientItem::Deactivate`.

```
void CExpContainerCntrItem::OnDeactivateUI(BOOL bUndoable)
{
    COleClientItem::OnDeactivateUI(bUndoable);

    // Скрыть объект, если он не является внедренным внешним объектом
    DWORD dwMisc = 0;
    m_lpObject->GetMiscStatus(GetDrawAspect(), &dwMisc);
    if (dwMisc & OLEMISC_INSIDEOUT)
        DoVerb(OLEIVERB_HIDE, NULL);
}
```

*Класс представления.* Класс представления контейнера OLE изменяется мастером Application Wizard. Сначала объявляется новая переменная класса представления `m_pSelection`, которая инициализируется значением NULL в функции `CExpContainerView::OnInitialUpdate`.

```
CExpContainerCntrItem* m_pSelection;
```

Когда пользователь внедряет компонент OLE, этой переменной передается адрес объекта класса `CExpContainerCntrItem`, управляющего новым компонентом описанной ниже функции `CExpContainerView::OnInsertObject`. Так, в программе ExpContainer переменная `m_pSelection` всегда содержит указатель на последний внедренный компонент или значение NULL, если указатель отсутствует. В полной программе-контейнере переменная `m_pSelection` обычно используется как указатель на текущий *выбранный* компонент; в противном случае (если указатель отсутствует) ей присваивается

значение NULL. Функция `CExpContainerView::OnDraw`, созданная мастером Application Wizard, отображает последний внедренный компонент (если он есть), используя адрес, содержащийся в переменной `m_pSelection` для вызова функции `COleClientItem::Draw`. Эта функция отображает компонент, проигрывая созданный программой-сервером метафайл. В полной программе-контейнере функция `OnDraw` обычно рисует собственно данные контейнера, а также *все* видимые внедренные компоненты.

Обработчик команды `New Object...` из меню `Edit` программы-контейнера добавляется мастером Application Wizard в класс представления. Этот обработчик называется `OnInsertObject` и отображает обычное диалоговое окно `InsertObject`, позволяющее выбрать тип внедренного компонента. После закрытия диалогового окна обработчик создает объект `CExpContainerCntrItem` для управления новым внедренным компонентом. Затем передает адрес объекта документа программы конструктору класса `CExpContainerCntrItem` и инициализирует объект класса `CExpContainerCntrItem`, используя информацию объекта диалогового окна, основанную на установках, выбранных в диалоговом окне. Далее активируется компонент для его редактирования на месте. В заключение адрес объекта `CExpContainerCntrItem` для нового внедренного компонента присваивается переменной `m_pSelection`.

Кроме того, в класс представления мастером Application Wizard добавляется обработчик для клавиши `Esc`, являющейся одной из клавиш-акселераторов `IDR_CNTR_INPLACE`, действующих при редактировании на месте. Обработчик сообщения называется `OnCancelEditCntr`. Он работает в сочетании с методом `OnCancelEditSrvr` класса представления сервера (см. выше) и вызывает функцию `COleClientItem::Close` для закрытия компонента, редактируемого на месте, если таковой имеется.

## Ресурсы

Файл `Afxolecl.rc`, включаемый мастером Application Wizard в файл ресурсов программы, определяет несколько ресурсов, используемых классами контейнеров OLE из библиотеки MFC. Мастер Application Wizard определяет отдельное меню и соответствующую таблицу акселераторов для каждого из двух режимов, в которых может выполняться программа `ExpContainer`:

- Меню `IDR_MAINFRAME` и таблица акселераторов используется, когда нет активного внедренного компонента. Это меню содержит обычные команды, определяемые мастером Application Wizard для программы, которая не относится к OLE, и несколько команд меню `Edit`, указанных в контейнере OLE. К особенностям контейнера относятся команды `Paste Special...`, `Insert New Object...` и `Links...` и место для подменю `Object` (первоначально помеченное "<<OLE VERBS GO HERE>>"). Из всех специфических команд контейнера в генерируемом мастером Application Wizard тексте программы реализуется только команды подменю `Object` и команда `Insert New Object...`
- Меню `IDR_CNTR_INPLACE` и таблица акселераторов используются для редактирования внедренного компонента "на месте". При редактировании на месте меню сервера объединяется с меню контейнера (см. выше). По некоторым причинам меню `IDR_CNTR_INPLACE` первоначально представляется редактором в режиме `View As PopUp` (всплывающее меню `File` и два разделителя в единственном всплывающем меню). Чтобы представить меню `IDR_CNTR_INPLACE` в обычном режиме, в котором пункты отображаются в строке меню, щелкните правой кнопкой мыши на любом месте окна редактора меню или на опции `View As PopUp` контекстного меню, чтобы сбросить соответствующий флажок.

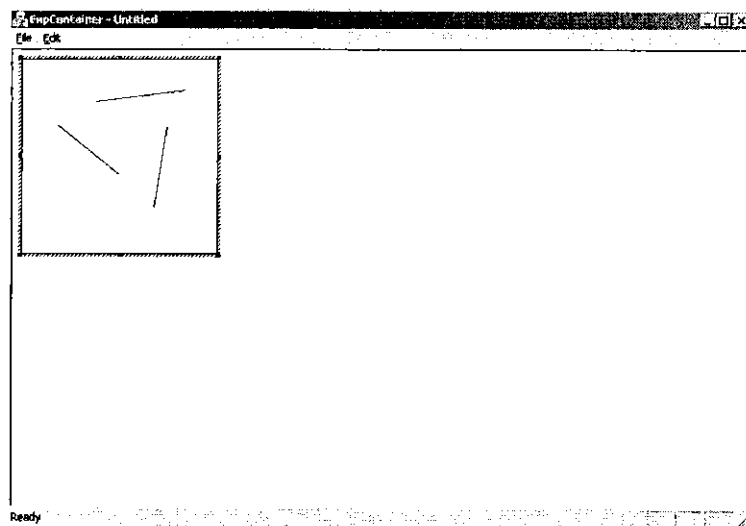
## Отладка программы ExpContainer

Построим и выполним программу `ExpContainer`. Если программа `ExpServer` еще не выполнялась в автономном режиме, сделайте это сейчас, чтобы зарегистрировать OLE-сервер `ExpServer`. Из программы `ExpServer` можно выйти, она не обязана выполняться при внедрении компонента в

ExpContainer. После запуска ExpContainer внедрите компонент OLE, выбрав в меню Edit команду Insert New Object.... Теперь программа будет отображать диалоговое окно Insert Object, позволяющее выбрать тип внедряемого компонента. Список Object Type показывает все типы компонентов, зарегистрированные программами-серверами OLE. Можно внедрить компонент *любого* из этих типов. (Преимущество OLE заключается в том, что контейнер может внедрить компонент любого типа, даже не воспринимая сами данные.) Выберите тип компонента ExpServer Document, зарегистрированный программой ExpServer, для проверки программ ExpServer и ExpContainer. При щелчке на кнопке OK в диалоговом окне Insert Object внедряется новый пустой компонент ExpServer, отображаемый в левом верхнем углу окна контейнера. Компонент автоматически активизируется для редактирования на месте, т. е. программа ExpServer выполняется как сервер в режиме редактирования "на месте".

Всплывающие меню программ ExpServer (Edit и Help) и ExpContainer (File) объединены для предоставления возможности выбора команд рисования ExpServer. Заметьте также, что компонент окружен границей с маркерами управления, позволяющими изменить размер рисунка (сейчас оставьте его неизменным). Эта граница, как и окно представления внутри нее, созданы программой ExpServer. Если в программе-сервере определена панель инструментов для редактирования в режиме на месте, то она будет отображена вместо любых инструментов, отображаемых контейнером. Программы ExpServer и ExpContainer не поддерживают методы внедрения, реализуемые путем копирования блока данных из документа сервера и выполнения команды Paste или Paste Special... в меню контейнера Edit или путем перетаскивания данных из окна сервера в окно контейнера. Полнофункциональные сервер и контейнер позволяют внедрять компонент таким способом.

Создайте внутри компонента рисунок с помощью мыши. Обратите внимание: программа ExpServer отображает крестообразный указатель, когда курсор мыши находится внутри окна представления компонента. Когда фигура нарисована, нажмите клавишу Esc, чтобы деактивировать компонент. Обычно полнофункциональный контейнер позволяет также деактивировать компонент щелчком мыши в окне представления за пределами компонента. Программа ExpServer прекратит выполнение, обычное меню ExpContainer восстановится и неактивный компонент отобразится программой ExpContainer внутри ее окна. Программа ExpContainer не рисует границу вокруг неактивного компонента и поэтому его размер фиксированный. Полнофункциональная программа-контейнер позволяет выделять неактивный внедренный компонент щелчком мыши внутри него. Затем окружает его границей, содержащей маркеры размеров, позволяющие изменить размеры компонента. Разумеется, в полнофункциональном контейнере внедренный компонент или компоненты будут отображаться вместе с собственными данными документа. Кроме того, можно перетащить весь компонент на новое место внутри документа контейнера.



Открыв подменю ServDe Object в меню Edit и выбрав команду Edit, активируйте компонент *в режиме редактирования на месте*. Обычно полнофункциональная программа контейнера позволяет активизировать внедренный компонент в режиме “на месте” двойным щелчком мыши внутри компонента. Используйте маркеры размеров, чтобы увеличить размер рисунка. Обратите внимание: при нажатии клавиши Esc для завершения сеанса редактирования рисунок уменьшается в размерах. Это происходит потому, что компонент в контейнере принимает свой начальный размер и, чтобы полностью разместиться, увеличенный рисунок сжимается.

Открыв подменю ServDe Object в меню Edit и выбрав команду Open, отредактируйте компонент *в режиме полного открытия*. При этом запускается программа ExpServer, отображающая компонент в отдельном окне. Если окно программы ExpContainer будет видимым при редактировании рисунка в ExpServer, то вы заметите, что любые изменения, сделанные в ExpServer, сразу же появляются в представлении рисунка в ExpContainer. После редактирования компонента выберите команду Exit & Return to Untitled в меню File программы ExpServer. При этом закроется окно ExpServer, и отредактированный неактивный компонент будет отображен в ExpContainer. Обратите внимание: при выполнении программы ExpServer в режиме полного открытия в ее меню File можно выбрать команду SaveCopy As..., чтобы сохранить на диске копию внедренного компонента как документа ExpServer. Запустите ExpServer как автономную программу, чтобы открыть этот документ.

## Текст программы ExpContainer

Исходные тексты программы ExpContainer приведены в листингах 24.13—24.22.

---

### Листинг 24.13.

```
// CntrItem.cpp : реализация класса CExpContainerCntrItem
//

#include "stdafx.h"
#include "ExpContainer.h"

#include "ExpContainerDoc.h"
#include "ExpContainerView.h"
#include "CntrItem.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// Реализация CExpContainerCntrItem

IMPLEMENT_SERIAL(CExpContainerCntrItem, COleClientItem, 0)

CExpContainerCntrItem::CExpContainerCntrItem(
    (CExpContainerDoc* pContainer)
    : COleClientItem(pContainer)
{
    // TODO: добавьте сюда однократный код конструктора
}

CExpContainerCntrItem::~CExpContainerCntrItem()
{
    // TODO: добавьте сюда код очистки
}
```

```

void CExpContainerCntrItem::OnChange(OLE_NOTIFICATION nCode,
    DWORD dwParam)
{
    ASSERT_VALID(this);

    COleClientItem::OnChange(nCode, dwParam);

    // Когда объект редактируется (на месте или сервером)
    // он посылает сообщения OnChange

    // TODO: обозначить объект как подлежащий перерисовке
    // функцией UpdateAllViews (соответственно требованиям
    // приложения)

    GetDocument()->UpdateAllViews(NULL);
        // пока что обновляются ВСЕ представления
}

BOOL CExpContainerCntrItem::OnChangeItemPosition(const CRect& rectPos)
{
    ASSERT_VALID(this);

    // Во время активации функция CExpContainerCntrItem::OnChangeItemPosition
    // вызывается сервером для изменения положения окна объекта.
    // Обычно это результат изменения данных в документе, например,
    // изменения размера объекта.
    // По умолчанию вызывается базовый класс, который вызовет
    // COleClientItem::SetItemRects для перемещения объекта в
    // новое положение.

    if (!COleClientItem::OnChangeItemPosition(rectPos))
        return FALSE;

    // TODO: обновить содержимое буфера объекта

    return TRUE;
}

void CExpContainerCntrItem::OnGetItemPosition(CRect& rPosition)
{
    ASSERT_VALID(this);

    // Во время активации, CExpContainerCntrItem::OnGetItemPosition
    // вызывается для получения положения объекта. Реализация по
    // умолчанию просто возвращает четыре координаты прямоугольника.
    // Обычно этот прямоугольник показывает текущее положение
    // объекта в окне просмотра.
    // Можно получить текущее представление, вызвав
    // CExpContainerCntrItem::GetActiveView.

    // TODO: верните правильный прямоугольник
    // (в пикселах) в rPosition

    rPosition.SetRect(10, 10, 210, 210);
}

```



```

void CExpContainerCntrItem::OnActivate()
{
    // Разрешается только один активный объект в рамке.
    CExpContainerView* pView = GetActiveView();
    ASSERT_VALID(pView);
    COleClientItem* pItem = GetDocument()->GetInPlaceActiveItem(pView);
    if (pItem != NULL && pItem != this)
        pItem->Close();

    COleClientItem::OnActivate();
}

void CExpContainerCntrItem::OnDeactivateUI(BOOL bUndoable)
{
    COleClientItem::OnDeactivateUI(bUndoable);

    DWORD dwMisc = 0;
    m_lpObject->GetMiscStatus(GetDrawAspect(), &dwMisc);
    if (dwMisc & OLEMISC_INSIDEOUT)
        DoVerb(OLEIVERB_HIDE, NULL);
}

void CExpContainerCntrItem::Serialize(CArchive& ar)
{
    ASSERT_VALID(this);
    // Сначала вызовите базовый класс для чтения данных в COleClientItem.
    // поскольку производится установка указателя m_pDocument,
    // полученного из CExpContainerCntrItem::GetDocument,
    // рекомендуется сначала использовать функцию Serialize базового
    // класса.
    COleClientItem::Serialize(ar);

    // теперь сохраним/загрузим данные, специфичные для
    // CExpContainerCntrItem
    if (ar.IsStoring())
    {
        // TODO: добавьте сюда код сохранения
    }
    else
    {
        // TODO: добавьте сюда код загрузки
    }
}

// Диагностика класса CExpContainerCntrItem

#ifdef _DEBUG
void CExpContainerCntrItem::AssertValid() const
{
    COleClientItem::AssertValid();
}

void CExpContainerCntrItem::Dump(CDumpContext& dc) const
{
    COleClientItem::Dump(dc);
}
#endif

```

---

**Листинг 24.14.**

```
// CntrItem.h : интерфейс класса CExpContainerCntrItem
//

#pragma once

class CExpContainerDoc;
class CExpContainerView;

class CExpContainerCntrItem : public COleClientItem
{
    DECLARE_SERIAL(CExpContainerCntrItem)

// Конструкторы
public:
    CExpContainerCntrItem(CExpContainerDoc* pContainer = NULL);
    // pContainer может быть равен NULL для использования
    // IMPLEMENT_SERIALIZE. IMPLEMENT_SERIALIZE требует наличия в
    // классе конструктора без аргументов. Обычно объекты OLE
    // создаются с не-NULL указателем документа

// Атрибуты
public:
    CExpContainerDoc* GetDocument()
        { return reinterpret_cast<CExpContainerDoc*>(COleClientItem::GetDocument()); }
    CExpContainerView* GetActiveView()
        { return reinterpret_cast<CExpContainerView*>(COleClientItem::GetActiveView()); }

    public:
        virtual void OnChange(OLE_NOTIFICATION wNotification,
                               DWORD dwParam);
        virtual void OnActivate();
    protected:
        virtual void OnGetItemPosition(CRect& rPosition);
        virtual void OnDeactivateUI(BOOL bUndoable);
        virtual BOOL OnChangeItemPosition(const CRect& rectPos);

// Реализация
public:
    ~CExpContainerCntrItem();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif
    virtual void Serialize(CArchive& ar);
};
```

---

**Листинг 24.15.**

```
// ExpContainer.cpp : определяет поведение классов приложения.
//

#include "stdafx.h"
```

```

#include "ExpContainer.h"
#include "MainFrm.h"

#include "ExpContainerDoc.h"
#include "ExpContainerView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CExpContainerApp

BEGIN_MESSAGE_MAP(CExpContainerApp, CWinApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
    // Стандартные команды работы с файлами документов
    ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
    // Стандартная команда подготовки печати
    ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()

// Конструктор класса CExpContainerApp

CExpContainerApp::CExpContainerApp()
{
    // TODO: добавьте сюда собственный код конструктора
    // Поместите весь существенный код инициализации в
    // функцию InitInstance
}

// Единственный объект класса CExpContainerApp

CExpContainerApp theApp;

// Инициализация CExpContainerApp

BOOL CExpContainerApp::InitInstance()
{
    CWinApp::InitInstance();

    // Инициализация библиотек OLE
    if (!AfxOleInit())
    {
        AfxMessageBox(IDP_OLE_INIT_FAILED);
        return FALSE;
    }
    // Стандартная инициализация
    // Если вы не используете какие-то из возможностей и хотите
    // уменьшить размер оконечного исполняемого модуля,
    // удалите отдельные процедуры инициализации из последующего
    // кода
    // Измените строку-аргумент функции (ключ в реестре, под
    // которым хранятся в реестре ваши установки)
    // TODO: измените эту строку на что-нибудь подходящее,
    // например, название вашей компании или организации
    SetRegistryKey(_T("Local AppWizard-Generated Applications"));
}

```

```

LoadStdProfileSettings(4); // Загрузка установок из INI-файла
                             // (включая MRU)
// Регистрация шаблонов документов приложения. Шаблоны
// документов служат связью между документами, окнами документов
// и окнами представлений
CSingleDocTemplate* pDocTemplate;
pDocTemplate = new CSingleDocTemplate(
    IDR_MAINFRAME,
    RUNTIME_CLASS(CExpContainerDoc),
    RUNTIME_CLASS(CMainFrame),
    // основное окно SDI-приложения
    RUNTIME_CLASS(CExpContainerView));
pDocTemplate->SetContainerInfo(IDR_CNTR_INPLACE);
AddDocTemplate(pDocTemplate);
// Поиск в командной строке команд управления,
// DDE, открытия файлов
CCommandLineInfo cmdInfo;
ParseCommandLine(cmdInfo);
// Выполнение команд, указанных в командной строке. Вернет
// FALSE, если приложение запускалось с /RegServer, /Register,
// /Unregserver или /Unregister.
if (!ProcessShellCommand(cmdInfo))
    return FALSE;
// Прорисовка и обновление единственного
// проинициализированного окна
m_pMainWnd->ShowWindow(SW_SHOW);
m_pMainWnd->UpdateWindow();
return TRUE;
}

// CAboutDlg диалог, используемый в App About

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

// Данные для диалога
    enum { IDD = IDD_ABOUTBOX };

protected:
    virtual void DoDataExchange(CDataExchange* pDX);
// Поддержка DDX/DDV

// Реализация
protected:
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
}

```

```

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
END_MESSAGE_MAP()

// Команда приложения на запуск диалога
void CExpContainerApp::OnAppAbout()
{
    CAboutDlg aboutDlg;
    aboutDlg.DoModal();
}

// Обработчики сообщений класса CExpContainerApp

```

---

#### Листинг 24.16.

```

// ExpContainer.h : главный файл заголовков приложения ExpContainer
//

#pragma once

#ifdef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"          // основные символы

// CExpContainerApp:
// Смотрите реализацию этого класса в файле ExpContainer.cpp
//

class CExpContainerApp : public CWinApp
{
public:
    CExpContainerApp();

// Переопределения
public:
    virtual BOOL InitInstance();

// Реализация
    afx_msg void OnAppAbout();
    DECLARE_MESSAGE_MAP()
};

extern CExpContainerApp theApp;

```

---

#### Листинг 24.17.

```

// ExpContainerDoc.cpp : реализация класса CExpContainerDoc
//

#include "stdafx.h"
#include "ExpContainer.h"

#include "ExpContainerDoc.h"

```

```

#include "CntrItem.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CExpContainerDoc

IMPLEMENT_DYNCREATE(CExpContainerDoc, COleDocument)

BEGIN_MESSAGE_MAP(CExpContainerDoc, COleDocument)
    // Разрешить реализацию контейнера OLE по умолчанию
    ON_UPDATE_COMMAND_UI(ID_EDIT_PASTE,
        COleDocument::OnUpdatePasteMenu)
    ON_UPDATE_COMMAND_UI(ID_EDIT_PASTE_LINK,
        COleDocument::OnUpdatePasteLinkMenu)
    ON_UPDATE_COMMAND_UI(ID_OLE_EDIT_CONVERT,
        COleDocument::OnUpdateObjectVerbMenu)
    ON_COMMAND(ID_OLE_EDIT_CONVERT, COleDocument::OnEditConvert)
    ON_UPDATE_COMMAND_UI(ID_OLE_EDIT_LINKS,
        COleDocument::OnUpdateEditLinksMenu)
    ON_COMMAND(ID_OLE_EDIT_LINKS, COleDocument::OnEditLinks)
    ON_UPDATE_COMMAND_UI_RANGE(ID_OLE_VERB_FIRST,
        ID_OLE_VERB_LAST, COleDocument::OnUpdateObjectVerbMenu)
END_MESSAGE_MAP()

// Конструктор/деструктор класса CExpContainerDoc

CExpContainerDoc::CExpContainerDoc()
{
    // Использование составных файлов OLE
    EnableCompoundFile();

    // TODO: добавьте сюда одноразовый код конструктора
}

CExpContainerDoc::~CExpContainerDoc()
{
}

BOOL CExpContainerDoc::OnNewDocument()
{
    if (!COleDocument::OnNewDocument())
        return FALSE;

    // TODO: добавьте сюда код реинициализации
    // (SDI-документы будут повторно использовать этот документ)

    return TRUE;
}

// Сериализация класса CExpContainerDoc

void CExpContainerDoc::Serialize(CArchive& ar)
{

```

```

        if (ar.IsStoring())
        {
            // TODO: добавьте сюда код сохранения
        }
        else
        {
            // TODO: добавьте сюда код загрузки
        }

        // Вызов базового класса COleDocument разрешает сериализацию
        // объектов документа контейнера COleClientItem
        COleDocument::Serialize(ar);
    }

    // Диагностика CExpContainerDoc

#ifdef _DEBUG
    void CExpContainerDoc::AssertValid() const
    {
        COleDocument::AssertValid();
    }

    void CExpContainerDoc::Dump(CDumpContext& dc) const
    {
        COleDocument::Dump(dc);
    }
#endif // _DEBUG

    // Команды CExpContainerDoc

```

---

#### Листинг 24.18.

```

// ExpContainerDoc.h : интерфейс класса CExpContainerDoc
//

#pragma once

class CExpContainerDoc : public COleDocument
{
protected: // используются только для сериализации
    CExpContainerDoc();
    DECLARE_DYNCREATE(CExpContainerDoc)

    // Атрибуты
public:

    // Операции
public:

    // Переопределения
public:
    virtual BOOL OnNewDocument();
    virtual void Serialize(CArchive& ar);

    // Реализация
public:
    virtual ~CExpContainerDoc();

```

```

#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Сгенерированные функции карты сообщений
protected:
    DECLARE_MESSAGE_MAP()
};

```

---

## Листинг 24.19.

```

// ExpContainerView.cpp : реализация класса CExpContainerView
//

#include "stdafx.h"
#include "ExpContainer.h"

#include "ExpContainerDoc.h"
#include "CntrItem.h"
#include "ExpContainerView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CExpContainerView

IMPLEMENT_DYNCREATE(CExpContainerView, CView)

BEGIN_MESSAGE_MAP(CExpContainerView, CView)
    ON_WM_DESTROY()
    ON_WM_SETFOCUS()
    ON_WM_SIZE()
    ON_COMMAND(ID_OLE_INSERT_NEW, OnInsertObject)
    ON_COMMAND(ID_CANCEL_EDIT_CNTR, OnCancelEditCntr)
    ON_COMMAND(ID_FILE_PRINT, OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
END_MESSAGE_MAP()

// Конструктор/деструктор класса CExpContainerView

CExpContainerView::CExpContainerView()
{
    m_pSelection = NULL;
    // TODO: добавьте сюда код конструктора
}

CExpContainerView::~CExpContainerView()
{
}

```



```

BOOL CExpContainerView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Измените класс или стили окна здесь, изменяя
    // или добавляя поля в структуре cs

    return CView::PreCreateWindow(cs);
}

// Прорисовка CExpContainerView

void CExpContainerView::OnDraw(CDC* pDC)
{
    ;
    CExpContainerDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    // TODO: добавьте сюда код отображения собственных данных
    // TODO: также добавьте код отображения OLE-объектов.

    // Прорисовывает объекты в произвольной позиции. Этот код
    // удаляется после добавления реального.

    // TODO: удалите этот код после добавления собственного
    // кода отображения.

    if (m_pSelection == NULL)
    {
        POSITION pos = pDoc->GetStartPosition();
        m_pSelection = DYNAMIC_DOWNCAST(CExpContainerCntrItem,
            pDoc->GetNextClientItem(pos));
    }
    if (m_pSelection != NULL)
    {
        CSize size;
        CRect rct;
        GetClientRect(&rct);

        if (SUCCEEDED(m_pSelection->GetExtent(&size,
            m_pSelection->m_nDrawAspect)))
        {
            pDC->HIMETRICToLP(&size);
            rct.right = size.cx;
            rct.bottom = size.cy;
        }
        m_pSelection->Draw(pDC, rct);
    }
}

void CExpContainerView::OnInitialUpdate()
{
    CView::OnInitialUpdate();

    // TODO: удалите этот код после написания собственного
    // кода выборки
    m_pSelection = NULL;    // инициализация выборки
}

```

```

// Печать CExpContainerView

BOOL CExpContainerView::OnPreparePrinting(CPrintInfo* pInfo)
{
    // подготовка по умолчанию
    return DoPreparePrinting(pInfo);
}

void CExpContainerView::OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo)
{
    // TODO: добавьте сюда код инициализации
}

void CExpContainerView::OnEndPrinting(CDC* pDC, CPrintInfo* pInfo)
{
    // TODO: добавьте сюда код очистки
}

void CExpContainerView::OnDestroy()
{
    // Деактивируйте объект после удаления; это важно
    // если используется разделенное окно отображения
    COleClientItem* pActiveItem =
        GetDocument()->GetInPlaceActiveItem(this);
    if (pActiveItem != NULL && pActiveItem->GetActiveView() == this)
    {
        pActiveItem->Deactivate();
        ASSERT(GetDocument()->GetInPlaceActiveItem(this) == NULL);
    }
    CView::OnDestroy();
}

// Поддержка и команды OLE Client

BOOL CExpContainerView::IsSelected(const COleObject* pDocItem) const
{
    // Реализация, адекватная вашему выбору, состоит только из
    // объектов CExpContainerCntrItem. Для поддержки различных
    // механизмов обработки реализация должна быть изменена

    // TODO: реализуйте здесь функцию проверки
    // выбранного объекта OLE

    return pDocItem == m_pSelection;
}

void CExpContainerView::OnInsertObject()
{
    // Вызов стандартного окна Insert Object для получения
    // информации о новом объекте CExpContainerCntrItem
    COleInsertDialog dlg;
    if (dlg.DoModal() != IDOK)
        return;

    BeginWaitCursor();
}

```

```

CExpContainerCntrItem* pItem = NULL;
TRY
{
    // Создание нового объекта связанного с данным документом
    CExpContainerDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    pItem = new CExpContainerCntrItem(pDoc);
    ASSERT_VALID(pItem);

    // Инициализация объекта по данным диалога
    if (!dlg.CreateItem(pItem))
        AfxThrowMemoryException(); // вызов исключений.
    ASSERT_VALID(pItem);

    if (dlg.GetSelectionType() == COleInsertDialog::createNewItem)
        pItem->DoVerb(OLEIVERB_SHOW, this);

    ASSERT_VALID(pItem);
    // Это произвольный интерфейс пользователя,
    // устанавливающий последний вставленный объект

    // TODO: переделайте выборку согласно требованиям
    // вашего приложения
    m_pSelection = pItem;
    // установить выборку на последний объект
    pDoc->UpdateAllViews(NULL);
}
CATCH(CException, e)
{
    if (pItem != NULL)
    {
        ASSERT_VALID(pItem);
        pItem->Delete();
    }
    AfxMessageBox(IDP_FAILED_TO_CREATE);
}
END_CATCH

EndWaitCursor();
}

// Следующий обработчик предоставляет стандартный клавиатурный
// интерфейс пользователя для отмены редактирования на месте.
// В этом случае деактивацию вызывает контейнер, а не сервер

void CExpContainerView::OnCancelEditCntr()
{
    // Close any in-place active item on this view.
    COleClientItem* pActiveItem =
        GetDocument()->GetInPlaceActiveItem(this);
    if (pActiveItem != NULL)
    {
        pActiveItem->Close();
    }
    ASSERT(GetDocument()->GetInPlaceActiveItem(this) == NULL);
}

```

```

// Специальная обработка OnSetFocus и OnSize требуется, если
// объект редактируется на месте
void CExpContainerView::OnSetFocus(CWnd* pOldWnd)
{
    ColeClientItem* pActiveItem =
        GetDocument()->GetInPlaceActiveItem(this);
    if (pActiveItem != NULL &&
        pActiveItem->GetItemState() ==
            ColeClientItem::activeUIState)
    {
        // требуется установить фокус на этот элемент,
        // если он в данном окне представления
        CWnd* pWnd = pActiveItem->GetInPlaceWindow();
        if (pWnd != NULL)
        {
            pWnd->SetFocus(); // не вызывать базовый класс
            return;
        }
    }

    CView::OnSetFocus(pOldWnd);
}

void CExpContainerView::OnSize(UINT nType, int cx, int cy)
{
    CView::OnSize(nType, cx, cy);
    ColeClientItem* pActiveItem =
        GetDocument()->GetInPlaceActiveItem(this);
    if (pActiveItem != NULL)
        pActiveItem->SetItemRects();
}

void CExpContainerView::OnFilePrint()
{
    // По умолчанию предоставить документу обработку печати с помощью
    // IOleCommandTarget. Если вам не нужно это действие, удалите
    // вызов ColeDocObjectItem::DoDefaultPrinting.
    // Если вызов не удастся, мы попробуем печать по умолчанию
    // с использованием интерфейса IPrint.
    CPrintInfo printInfo;
    ASSERT(printInfo.m_ppd != NULL);
    if (S_OK == ColeDocObjectItem::DoDefaultPrinting
        (this, &printInfo))
        return;

    CView::OnFilePrint();
}

// Диагностика класса CExpContainerView

#ifdef _DEBUG
void CExpContainerView::AssertValid() const
{
    CView::AssertValid();
}

```

```

void CExpContainerView::Dump(CDumpContext& dc) const
{
    CView::Dump(dc);
}

CExpContainerDoc* CExpContainerView::GetDocument() const
// не отладочная версия встроена
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CExpContainerDoc)));
    return (CExpContainerDoc*)m_pDocument;
}
#endif // _DEBUG

// Обработчики сообщений класса CExpContainerView

```

---

#### Листинг 24.20.

```

// ExpContainerView.h : интерфейс класса CExpContainerView
//

#pragma once

class CExpContainerCntrItem;

class CExpContainerView : public CView
{
protected: // используется только для сериализации
    CExpContainerView();
    DECLARE_DYNCREATE(CExpContainerView)

// Атрибуты
public:
    CExpContainerDoc* GetDocument() const;
    // m_pSelection содержит указание на CExpContainerCntrItem.
    // Для многих приложений, такая переменная не подходит для
    // указания на выбранный объект, например, для указания на
    // множество объектов или на объекты не класса
    // CExpContainerCntrItem.

    // TODO: замените механизм выборки на подходящий для
    // вашего приложения
    CExpContainerCntrItem* m_pSelection;

// Операции
public:

// Переопределения
    public:
        virtual void OnDraw(CDC* pDC);
        // переопределена для прорисовки этого вида
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
protected:
    virtual void OnInitialUpdate();
    // впервые вызывается после конструктора
    virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);

```

```

        virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
        virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
        virtual BOOL IsSelected(const CObject* pDocItem) const;
        // Поддержка контейнера

// Реализация
public:
    virtual ~CExpContainerView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Сгенерированные функции карты сообщений
protected:
    afx_msg void OnDestroy();
    afx_msg void OnSetFocus(CWnd* pOldWnd);
    afx_msg void OnSize(UINT nType, int cx, int cy);
    afx_msg void OnInsertObject();
    afx_msg void OnCancelEditCntr();
    afx_msg void OnFilePrint();
    DECLARE_MESSAGE_MAP()
};

#ifdef _DEBUG // Отладочная версия в файле ExpContainerView.cpp
inline CExpContainerDoc* CExpContainerView::GetDocument() const
    { return reinterpret_cast<CExpContainerDoc*>(m_pDocument); }
#endif

```

---

#### Листинг 24.21.

```

// MainFrm.cpp : реализация класса CMainFrame
//

#include "stdafx.h"
#include "ExpContainer.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
    ON_WM_CREATE()
END_MESSAGE_MAP()

static UINT indicators[] =
{
    ID_SEPARATOR,          // индикатор строки состояния

```

```

        ID_INDICATOR_CAPS,
        ID_INDICATOR_NUM,
        ID_INDICATOR_SCROLL,
    };

// Конструктор/деструктор CMainFrame

CMainFrame::CMainFrame()
{
    // TODO: добавьте сюда код инициализации элемента
}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT,
        WS_CHILD | WS_VISIBLE | CBRS_TOP | CBRS_GRIPPER |
        CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC) ||
        !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
    {
        TRACE0("Failed to create toolbar\n");
        return -1;      // не удалось создать
    }

    if (!m_wndStatusBar.Create(this) ||
        !m_wndStatusBar.SetIndicators(indicators,
        sizeof(indicators)/sizeof(UINT)))
    {
        TRACE0("Failed to create status bar\n");
        return -1;      // не удалось создать
    }
    // TODO: Удалите три следующие строки, если вы не хотите,
    // чтобы панель инструментов была парящей
    m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
    EnableDocking(CBRS_ALIGN_ANY);
    DockControlBar(&m_wndToolBar);

    return 0;
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    if( !CFrameWnd::PreCreateWindow(cs) )
        return FALSE;
    // TODO: Измените стиль или класс окна, изменяя
    // и добавляя поля структуры cs

    return TRUE;
}

```

```

// Диагностика CMainFrame

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
    CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
    CFrameWnd::Dump(dc);
}

#endif // _DEBUG

// Обработчики сообщений CMainFrame

```

---

#### Листинг 24.22.

```

// MainFrm.h : интерфейс класса CMainFrame
//

#pragma once
class CMainFrame : public CFrameWnd
{
protected: // используется только для сериализации
    CMainFrame();
    DECLARE_DYNCREATE(CMainFrame)

// Атрибуты
public:

// Операции
public:

// Переопределения
public:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);

// Реализация
public:
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected: // элементы строки состояния
    CStatusBar m_wndStatusBar;
    CToolBar m_wndToolBar;

// Сгенерированные функции карты сообщений
protected:
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    DECLARE_MESSAGE_MAP()
};

```



## Резюме

---

Рассмотрен механизм OLE 2 и использование его основных средств.

- *Сервер OLE* – это программа, создающая, обслуживающая и редактирующая блоки данных OLE, называемые *объектами* или *компонентами*. *Контейнер OLE* – это программа, которая получает компонент OLE и встраивает его в свои документы. При *внедрении (embedding)* контейнер OLE хранит данные компонента как составную часть документа контейнера. При *связывании (linking)* OLE-сервер хранит данные OLE-компонента как часть документа сервера. Как при внедрении, так и при связывании, сервер и библиотеки OLE используются для отображения и редактирования данных. При внедрении документ может редактироваться в окне контейнера (редактирование *на месте* – *in place*) или в окне сервера (редактирование в режиме *полного открытия* – *fully-open*). При связывании документ редактируется только в окне сервера. *Автоматизация (Automation)* является механизмом OLE, который дает возможность программе *клиента автоматизации* использовать средства программы *сервера автоматизации*.
- *Опции мастера Application Wizard*. Библиотека MFC и мастера Visual C++ позволяют создавать программы, используя средства внедрения, связывания и автоматизации OLE. Чтобы создать исходный код для сервера OLE, сгенерируйте программу с помощью мастера Application Wizard и на вкладке Compound Document Support выберите одну из опций: Full-server, Mini-server или Both container and server. При выборе опции *Full-server* генерируется программа, выполняемая как автономное приложение или как сервер OLE для внедрения или связывания компонентов. При выборе опции Full-server мастер Application Wizard генерирует два дополнительных класса программы для управления компонентом OLE и границей, окружающей его при редактировании в окне контейнера. Он также добавляет в классы программы код, необходимый для сервера OLE. При выборе опции *Mini-server* генерируется программа, выполняемая только как сервер OLE для внедрения или связывания компонентов, а при выборе опции *Both container and server* – программа, которая может выполняться и как полный сервер, и как контейнер OLE.
- *Программа-сервер*. При создании сервера обычно добавляется специальный код приложения, расширяющий возможности OLE-программы, а также код для реализации собственных средств программы (например, средств рисования, редактирования текста и т.п.).
- *Программа-контейнер*. Для создания исходного кода контейнера OLE сгенерируйте программу мастером Application Wizard и на вкладке Compound Document Support окна Application Wizard выберите опцию Container (Контейнер) или Both Container And Server (Контейнер и сервер). При генерации программы контейнера мастер Application Wizard создает новый класс для управления внедрением и связыванием компонентов, а также вносит изменения в стандартные классы программы. Как и при работе с сервером, необходимо добавить код для расширения возможностей программы контейнера OLE, в частности, для управления документами (выполнения операций создания, редактирования и сохранения).
- *OLE и ActiveX*. В документации Visual Studio.NET содержится замечание, рекомендуемое по возможности заменять применения технологии OLE на технологию ActiveX, позволяющую представлять не только данные, но и элементы управления.

## Глава 25

# ActiveX и Visual C++

---

- **Собственный ActiveX-элемент**
- **Программа-контейнер ActiveX-элемента**

Как правило, под элементом ActiveX понимают мобильный программный модуль, решающий определенную задачу или набор задач. Он может, например, отображать календарь, осуществлять презентацию с использованием средств мультимедиа, создавать диаграмму, поддерживать разговор в сети Internet или служить формой для чтения и записи информации в базу данных. Элементы ActiveX похожи на такие стандартные элементы управления Windows, как кнопки или списки, которые можно размещать в диалоговых и других окнах. Однако эти элементы можно создать самостоятельно, загрузить из сети Internet или получить из других источников. Они могут реализовать почти неограниченный набор функций. Ранее ActiveX называли *элементами управления OLE* или *элементами ОСХ*, так как в файлах для хранения элементов ActiveX используется расширение .osx. Многие разработчики по-прежнему называют их элементами ОСХ (коды и ресурсы ActiveX хранятся в отдельном файле с расширением .osx).

Если вы уже разработали или получили элемент ActiveX, то его можно включить в программы Visual C++, Visual Basic, Visual J++, поместить на Web-страничке в сети Internet или intranet (локальная сеть с протоколом Internet), использовать в приложении баз данных Access или в любой программе, созданной как контейнер элемента ActiveX (называемой контейнерным приложением или контейнером). Элементы ActiveX можно устанавливать в любое приложение, независимо от языка программирования. При этом данное контейнерное приложение почти так же тесно взаимодействует с элементом, как и с частью собственного кода. Элементы ActiveX предоставляют три основных способа взаимодействия с контейнерными приложениями:

- *Свойства* – это атрибуты элемента, например, цвет фона, который программа-контейнер может читать или изменять.
- *Метод* – это функция элемента ActiveX, которую может вызывать программа-контейнер. Например, метод может отобразить диалоговое окно About с информацией об элементе.
- *Событие* – это то, что происходит внутри элемента, например, щелчок на элементе, передающий сообщение контейнеру. Элемент ActiveX выполняет это действие, вызывая для обработки соответствующего события функцию программы-контейнера.

Рассмотреть целиком технологию ActiveX в одной главе невозможно. Поэтому в этой главе содержится только краткое введение в мир элементов ActiveX. В первой части главы описан простой элемент ActiveX, демонстрирующий основные технические приемы для определения свойств, методов и событий. Во второй части главы приведено контейнерное приложение для пользовательского элемента ActiveX, разработанное специально для отображения и управления созданным в первой части главы элементом ActiveX.

## Собственный ActiveX-элемент

---

Для изучения технологии проектирования ActiveX-элементов создадим простой элемент ActiveX, называемый ActiveXControl, предназначенный для отображения рисунка. При щелчке на элементе он переключается между двумя версиями рисунка (подобно стандартному элементу управления

“флажок”). Свойства элемента позволяют контейнеру изменять цвет фона элемента (цвет рисунка матовый), а также добавлять и удалять рамку вокруг рисунка. Для отображения диалогового окна About, содержащего информацию об элементе, он предоставляет метод, вызываемый контейнером.

## Генерация и модификация программных файлов

Процедура генерации файлов, содержащих текст программы компонента ActiveX ActiveXControl, мало отличается от уже знакомой вам процедуры генерации приложения.

1. Откройте окно New Project.
2. Выберите в списке типов проектов MFC ActiveX Control.
3. Введите в текстовое поле Project Name имя ActiveXControl, а в поле Location – путь к папке файла проекта.
4. Нажмите кнопку ОК. Мастер отобразит диалоговое окно для выбора параметров, а затем сгенерирует файлы с исходным кодом для элемента ActiveX. Используемый мастер напоминает генерирующий приложения мастер Application Wizard.
5. Изменять установки по умолчанию в данном случае не требуется. Щелкните в диалоговом окне New Project Information на кнопке ОК.

Теперь настроим ресурсы программы, создадим код для отображения рисунков и добавим обработчик сообщений. Приведенные инструкции будут краткими, поскольку соответствующие методы уже рассмотрены в предыдущих главах.

*Отображаемые иллюстрации.* Два рисунка, отображаемых элементом ActiveX, создаются с использованием растровых изображений. Добавьте два файла изображений (Bitmaps) к ресурсам создаваемой программы. Убедитесь, что им присвоены идентификаторы IDB\_BITMAP1 и IDB\_BITMAP2. В графическом редакторе нарисуйте в этих файлах произвольные изображения или воспользуйтесь готовыми изображениями, вставив их через буфер обмена. В процессе работы эти изображения будут поочередно сменять друг друга.

*Изображение и значок программы.* При желании можно настроить растровое изображение значка программы с идентификатором ресурса IDB\_ACTIVEXCONTROL. Это изображение появится на панели элементов для помещения на создаваемое окно (вместе с изображениями стандартных элементов управления Windows) при разработке диалогового окна в программе контейнера, содержащего элемент ActiveXControl. Версия растрового изображения по умолчанию содержит буквы OCX. Кроме того, можно настроить значок программы IDB\_ABOUTDLL, отображаемый в диалоговом окне About.

*Отображение иллюстраций.* Далее необходимо добавить фрагмент программы для создания и отображения растрового изображения внутри элемента ActiveX (см. гл. 20). Добавим три переменные в начало определения класса CActiveXControlCtrl в файле ActiveXControlCtrl.h. Класс CActiveXControlCtrl – это класс, управляющий элементом ActiveX.

```
class CActiveXControlCtrl : public COleControl
{
    DECLARE_DYNCREATE(CActiveXControlCtrl)

public:
    CBitmap m_SpaceImage1, m_SpaceImage2, *m_CurrentImage;

    // Конструктор
public:
    CActiveXControlCtrl();
```

Переменные m\_SpaceImage1 и m\_SpaceImage2 используются для хранения двух растровых изображений, отображаемых в элементе ActiveX, а m\_CurrentImage является указателем на объект

отображаемого в данный момент растрового изображения. Для загрузки растровых изображений из соответствующих ресурсов и инициализации переменной `m_CurrentImage` добавьте в конструктор класса `CActiveXControlCtrl` в файле `ActiveXControlCtrl.cpp` следующий текст.

```
CActiveXControlCtrl::CActiveXControlCtrl()
{
    InitializeIIDs(&IID_DActiveXControl, &IID_DActiveXControlEvents);
    // TODO: Добавьте сюда собственный код.

    m_SpaceImage1.LoadBitmap (IDB_BITMAP1);
    m_SpaceImage2.LoadBitmap (IDB_BITMAP2);
    m_CurrentImage = &m_SpaceImage1; // первоначальное изображение
}
```

Когда элемент необходимо нарисовать или перерисовать, вызывается функция `OnDraw` класса `CActiveXControlCtrl`. Измените определение функции `OnDraw` в файле `ActiveXControlCtrl.cpp` так, чтобы внутри элемента `ActiveX` отображалось текущее растровое изображение. *Удалите* вызов функции `Ellipse` (фрагмент, сгенерированный мастером, рисует белый фон и вычерчивает эллипс). Затем добавьте текст, приведенный ниже. Обратите внимание: параметр `rcBounds`, передаваемый в функцию `OnDraw`, содержит текущие размеры элемента `ActiveX`, параметр `rcInvalid` – координаты текущей недействительной области внутри элемента (т.е. области, помеченной для перерисовки). Рисование с помощью функции `OnDraw` класса элемента `ActiveX` производится так же, как и рисование с помощью описанной в предыдущих главах функции `OnDraw` класса окна представления.

```
void CActiveXControlCtrl::OnDraw(
    CDC* pdc, const CRect& rcBounds, const CRect& rcInvalid)
{
    // TODO: Замените следующий код собственным кодом прорисовки.

    CBrush Brush (TranslateColor (GetBackColor ()));
    pdc->FillRect (rcBounds, &Brush);

    BITMAP MyBitmap;
    CDC MemDC;

    MemDC.CreateCompatibleDC (NULL);
    MemDC.SelectObject (*m_CurrentImage);
    m_CurrentImage->GetObject (sizeof (MyBitmap), &MyBitmap);
    pdc->BitBlt ((rcBounds.right - MyBitmap.bmWidth) / 2,
        (rcBounds.bottom - MyBitmap.bmHeight) / 2,
        MyBitmap.bmWidth, MyBitmap.bmHeight,
        &MemDC, 0, 0, SRCCOPY);
}
```

*Обработчик сообщения о щелчке мыши.* Добавьте в класс `CActiveXControlCtrl`, обработчик сообщения `WM_LBUTTONDOWN`. Функция `OnLButtonDown` вызывается при щелчке на элементе после отпускания кнопки мыши. Добавьте в обработчик следующий код:

```
// Обработчики сообщений класса CActiveXControlCtrl

void CActiveXControlCtrl::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: Добавьте сюда собственный код обработчика
    // или вызов стандартного
```

```

        if (m_CurrentImage == &m_SpaceImage1)
            m_CurrentImage = &m_SpaceImage2;
        else
            m_CurrentImage = &m_SpaceImage1;

        InvalidateControl ();

        ColeControl::OnLButtonUp(nFlags, point);
    }

```

Добавленный текст изменяет значение переменной `m_CurrentImage` таким образом, чтобы она указывала на другое растровое изображение. Затем для перерисовки элемента (используя новое растровое изображение) обработчик вызывает функцию `InvalidateControl`, которая вызовет функцию `OnDraw`.

## Свойства ActiveX

Определим два свойства ActiveX:

- *стандартное свойство* `BackColor`, дающее контейнеру возможность изменять цвет фона;
- *пользовательское свойство* `DisplayColor`, позволяющее контейнеру добавлять или убирать цветную рамку вокруг рисунка.

Свойство `BackColor` является *стандартным (stock)*. Это значит, что оно входит в набор общих свойств, включающих свойства `Caption`, `Font` и др., инициализируемых и сохраняемых MFC. MFC выполняет соответствующее действие в ответ на изменение значения стандартного свойства. Необходимо только сделать это свойство доступным и написать фрагмент программы, использующий его значение. Чтобы сделать свойство `BackColor` доступным, откройте вкладку `Resource View`, разверните узел на дереве, соответствующий классу страниц свойств, и на втором сверху подузле щелкните правой кнопкой, чтобы открыть контекстное меню. В меню выберите пункт `Add...` и в открывшемся меню выберите пункт `Property...`

В открывшемся окне выберите пункт `BackColor` в списке `Name`, убедитесь, что выбрана опция `Stock`. Завершите создание свойства нажатием кнопки `Add...` Код MFC хранит значение свойства `BackColor` и инициализирует его, задавая цвет окна или диалогового окна контейнера, в котором отображается компонент. Код MFC также объявляет элемент недействительным, тем самым вынуждая функцию `OnDraw` перерисовывать элемент при каждом изменении значения его свойства, производимого либо контейнером, либо самим элементом. Единственное, чего не делает MFC – не использует цвет, заданный в свойстве, для рисования фона элемента. Позже в метод `OnDraw` будет введен соответствующий текст для выполнения этой операции.

Свойство `DisplayColor` является *пользовательским*, а это означает, что его нужно создать самостоятельно, присвоить ему имя и написать большую часть кода для поддержки. Чтобы определить свойство `DisplayColor`, вновь откройте окно добавления свойств, выберите опцию `Member Variable`, в поле `Name` введите `DisplayColor`, в списке `Type` введите `BOOL`. Оставьте в текстовом поле `Variable Name` имя `m_DisplayColor`, а в текстовом поле `Notification Function` – имя `OnDisplayColorChanged`.

Генерируется пользовательское свойство с именем `DisplayColor` и типом данных `BOOL` (т.е. оно может принимать значения `TRUE` или `FALSE`). Значение свойства будет храниться в переменной типа `BOOL` с именем `m_DisplayColor`, которую мастер определяет как переменную класса `CActiveXControlCtrl`. При каждом изменении значения свойства программой-контейнером MFC будет присваивать переменной `m_DisplayColor` новое значение и вызывать уведомляющую функцию `OnDisplayColorChanged` класса `CActiveXControlCtrl`. Мастер определит базовую оболочку для этой функции, а затем в нее следует самостоятельно добавить код. Так как свойство `DisplayColor` определяется пользователем, то необходимо явно инициализировать его значение.

Сделайте это, добавив вызов глобальной MFC-функции `PX_Bool` в функцию `CActiveXControlCtrl::DoPropExchange` в файле `CActiveXControlCtrl.cpp`. Этот вызов инициализирует свойство `DisplayColor` и присваивает значение `FALSE` переменной класса `m_DisplayColor`.

```
void CActiveXControlCtrl::DoPropExchange(CPropExchange* pPX)
{
    ExchangeVersion(pPX, MAKELONG(_wVerMinor, _wVerMajor));
    COleControl::DoPropExchange(pPX);

    // TODO: Вызывайте PX_ функции для каждого
    // пользовательского свойства.

    PX_Bool (pPX, _T("DisplayColor"), m_DisplayColor, FALSE);
}
```

Добавьте вызов функции `CActiveXControlCtrl::InvalidateControl` в уведомляющую функцию `OnDisplayColorChanged` класса `CActiveXControlCtrl` в файле `CActiveXControlCtrl.cpp`. При каждом изменении свойства `DisplayColor` программой-контейнером управление передается функции `OnDisplayColorChanged`. Вызов функции `InvalidateControl` приводит к вызову функции `CActiveXControlCtrl::OnDraw`, перерисовывающей элемент ActiveX с рамкой или без нее, согласно значению свойства `DisplayColor`.

```
void CActiveXControlCtrl::OnDisplayColorChanged(void)
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState());

    // TODO: Добавьте сюда собственный код обработчика

    InvalidateControl ();

    SetModifiedFlag();
}
```

Необходимо изменить в файле `ActiveXControlCtrl.cpp` функцию `CActiveXControlCtrl::OnDraw` так, чтобы при рисовании элемента она использовала текущие значения двух описанных свойств. Пояснения по методике рисования даны в гл. 19. Сначала *удалите* вызов функции `FillRect`. Затем добавьте в программу пару фрагментов. Два первых добавленных оператора рисуют фон элемента ActiveX, используя цвет, заданный свойством `BackColor` при вызове функции `COleControl::GetBackColor`. Функция `GetBackColor` возвращает цветовой код `OLE_COLOR`, соответствующий формату представления цвета в элементах ActiveX. При вызове функции `TranslateColor` полученный цветовой код преобразуется в значение `COLORREF`, соответствующее формату представления цвета в Windows. Передавая значение `OLE_COLOR` в функцию `COleControl::GetBackColor`, можно *установить* значение свойства `BackColor`.

```
void CActiveXControlCtrl::OnDraw(
    CDC* pdc, const CRect& rcBounds, const CRect& rcInvalid)
{
    // TODO: Замените следующий код собственным кодом прорисовки.

    CBrush Brush (TranslateColor (GetBackColor ()));
    pdc->FillRect (rcBounds, &Brush);

    BITMAP myBitmap;
    CDC MemDC;
```

```

MemDC.CreateCompatibleDC (NULL);
MemDC.SelectObject (*m_CurrentImage);
m_CurrentImage->GetObject (sizeof (MyBitmap), &MyBitmap);
pdc->BitBlt ((rcBounds.right - MyBitmap.bmWidth) / 2,
            (rcBounds.bottom - MyBitmap.bmHeight) / 2,
            MyBitmap.bmWidth, MyBitmap.bmHeight,
            &MemDC, 0, 0, SRCCOPY);

if (m_DisplayColor)
{
    CBrush *pOldBrush = (CBrush *) pdc->SelectStockObject
        (NULL_BRUSH);
    CPen Pen (PS_SOLID | PS_INSIDEFRAME, 10, RGB(0, 0, 0));
    CPen *pOldPen = pdc->SelectObject (&Pen);

    pdc->Rectangle (rcBounds);

    pdc->SelectObject (pOldPen);
    pdc->SelectObject (pOldBrush);
}
}

```

Если свойство DisplayColor, значение которого хранится в m\_DisplayColor, в текущий момент имеет значение TRUE, то второй добавленный фрагмент программы рисует рамку по границам элемента.

*Модификация страницы свойств.* Программа элемента ActiveX может содержать одну или несколько *страниц свойств*. Каждая страница определяется как диалоговое окно и содержит набор элементов управления для определения значений свойств элемента ActiveX. Как показано далее, проектируя программу-контейнер элемента ActiveX с использованием Visual C++, для задания значений свойств элемента можно использовать страницы свойств. Каждая такая страница отображается как вкладка диалогового окна Properties, отображаемого редактором диалоговых окон. Вначале проект элемента ActiveX содержит только одну заданную по умолчанию страницу свойств, определенную как ресурс диалогового окна IDD\_PROPPAGE\_ACTIVEXCONTROL.

Первоначальная версия страницы содержит только надпись с сообщением “TODO”. Надпись заменяется флажком, который можно использовать для изменения значения свойства DisplayColor. Для этого откройте ресурс диалогового окна IDD\_PROPPAGE\_ACTIVEXCONTROL в редакторе диалоговых окон Visual C++. Сначала удалите существующую надпись с сообщением TODO. Затем добавьте флажок, задав для него идентификатор IDC\_DISPLAYCOLOR и заголовок “Display a frame around the picture”. Свяжите новый флажок со свойством DisplayColor, выполнив следующие действия:

1. Щелкните правой кнопкой мыши на флажке, помещенном в окно.
2. Из контекстного меню выберите Add Variable.
3. В открытом окне установите флажок Control Variable, в списке Category выберите Value, в поле типа – Bool, в поле Name – m\_DisplayColor.

Созданное ранее свойство DisplayColor инициализируется значением FALSE. Поэтому флажок, соответствующий свойству DisplayColor, изначально не будет отмечен при отображении страницы свойств в процессе разработки программы-контейнера. На заключительном этапе необходимо добавить в программу вторую страницу свойств. Добавьте стандартную страницу свойств Color, которую можно использовать для выбора значения свойства BackColor при разработке программы-контейнера. Чтобы добавить эту страницу, откройте файл ActiveXControlCtrl.cpp и измените таблицу страниц свойств, определяющую все страницы свойств, предоставляемых элементом, как показано ниже:

```
// TODO: Добавляйте новые страницы свойств по мере нужды.
// Одновременно увеличивайте значение счетчика строкой ниже.
BEGIN_PROPPAGEIDS(CActiveXControlCtrl, 2)
    PROPPAGEID(CActiveXControlPropPage::guid)
    PROPPAGEID(CLSID_CColorPropPage)
END_PROPPAGEIDS(CActiveXControlCtrl)
```

Передаваемый в макрос BEGIN\_PROPPAGEIDS номер необходимо изменить с 1 на 2, а затем добавить новый вызов макроса PROPPAGEID. Первый вызов макроса PROPPAGEID в этой таблице добавляет измененную, заданную по умолчанию, страницу свойств. Стандартная страница свойств Color связана со свойством BackColor автоматически. Поэтому не нужно этого делать вручную.

## Методы ActiveX

В проект ActiveXControl включен метод, называемый AboutBox, определенный мастером при создании. Когда программа-контейнер вызывает этот метод, элемент ActiveX отображает диалоговое окно About, определенное в диалоговом ресурсе IDD\_ABOUTBOX\_ACTIVEXCONTROL. В этом упражнении не нужно определять дополнительные методы. Если хотите добавить метод в какой-либо другой разрабатываемый элемент ActiveX, это можно сделать с помощью контекстного меню, как и определение свойств.

## События ActiveX

Определив событие, программа элемента ActiveX может вызывать связанную с событием функцию Fire... (например, FireClick или FireModified), чтобы уведомить программу-контейнер о наступлении события. Вызов функции Fire... называют *активизацией события*. При активизации вызывается обработчик в программе-контейнере, если он был определен для этого события. Подобно свойству или методу, событие может быть *пользовательским* или *стандартным*:

- Для стандартных событий библиотека MFC предоставляет функцию Fire... и код, вызывающий ее в определенные моменты времени.
- Для пользовательских событий Visual Studio генерирует новую функцию Fire... Если требуется активизировать событие, то необходимо самому написать код для вызова этой функции.

Внутри класса COleControl определена функция FireClick. MFC автоматически вызывает ее для активизации события Click каждый раз, когда пользователь щелкает на элементе. Следовательно, в текст программы не нужно добавлять обращения к функции FireClick. Но если требуется активизировать это событие в другие моменты времени, то можно добавлять подобные обращения.

## Построение ActiveX-элемента

Для построения элемента ActiveX, называемого ActiveXControl, щелкните на кнопке Build. Генерируется файл ActiveXControl.ocx элемента ActiveX, который автоматически регистрируется в системе, что сделает возможным доступ к нему из программы-контейнера. В следующей части главы будет написана программа-контейнер, отображающая созданный элемент. Для передачи элемента ActiveX другим пользователям (возможно, вместе с демонстрационной программой-контейнером) необходимо предоставить инсталляционную программу, регистрирующую компонент в системе пользователя. Информация по этому вопросу находится в справочной системе.

Разрабатывая компонент ActiveX, периодически нужно проверять его функции, отображая его в программе Test Container. Если эта программа включена в установку Visual C++, то ее можно запустить, выбрав команду ActiveX Control Test Container в меню Tools. Дополнительная информация содержится в справочной системе.



## Текст элемента ActiveXControl

Текст программы элемента ActiveX, называющегося ActiveXControl, приведен в листингах 25.1—25.6.

---

### Листинг 25.1.

```
#pragma once

// ActiveXControl.h : главный заголовочный файл для ActiveXControl.DLL

#ifdef !defined( __AFXCTL_H__ )
#error include 'afxctl.h' before including this file
#endif

#include "resource.h"          // основные символы

// CActiveXControlApp : Смотрите реализацию данного класса в файле
// ActiveXControl.cpp.

class CActiveXControlApp : public COleControlModule
{
public:
    BOOL InitInstance();
    int ExitInstance();
};

extern const GUID CDECL _tlid;
extern const WORD _wVerMajor;
extern const WORD _wVerMinor;
```

---

### Листинг 25.2.

```
// ActiveXControl.cpp : реализация класса CActiveXControlApp и регистрация DLL.

#include "stdafx.h"
#include "ActiveXControl.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

CActiveXControlApp NEAR theApp;

const GUID CDECL BASED_CODE _tlid =
    { 0xF8B38A52, 0xECFE, 0x446A, { 0xA0, 0x26, 0x2A, 0x81, 0x85,
    0xCB, 0x5A, 0xFB } };
const WORD _wVerMajor = 1;
const WORD _wVerMinor = 0;

// CActiveXControlApp::InitInstance - инициализация DLL

BOOL CActiveXControlApp::InitInstance()
{
    BOOL bInit = COleControlModule::InitInstance();
```

```

        if (bInit)
        {
            // TODO: Добавьте сюда инициализацию собственных модулей.
        }

        return bInit;
    }

// CActiveXControlApp::ExitInstance - выгрузка DLL
int CActiveXControlApp::ExitInstance()
{
    // TODO: Добавьте сюда выгрузку собственных модулей.

    return COleControlModule::ExitInstance();
}

// DllRegisterServer - Добавка записей в системный реестр.
STDAPI DllRegisterServer(void)
{
    AFX_MANAGE_STATE(_afxModuleAddrThis);

    if (!AfxOleRegisterTypeLib(AfxGetInstanceHandle(), _tlid))
        return ResultFromScode(SELFREG_E_TYPELIB);

    if (!COleObjectFactoryEx::UpdateRegistryAll(TRUE))
        return ResultFromScode(SELFREG_E_CLASS);

    return NOERROR;
}

// DllUnregisterServer - Удаление записей из системного реестра
STDAPI DllUnregisterServer(void)
{
    AFX_MANAGE_STATE(_afxModuleAddrThis);

    if (!AfxOleUnregisterTypeLib(_tlid, _wVerMajor, _wVerMinor))
        return ResultFromScode(SELFREG_E_TYPELIB);

    if (!COleObjectFactoryEx::UpdateRegistryAll(FALSE))
        return ResultFromScode(SELFREG_E_CLASS);

    return NOERROR;
}

```

---

### Листинг 25.3.

```

#pragma once

// ActiveXControlCtrl.h : объявление класса ActiveX Control
// CActiveXControlCtrl.
// Смотрите реализацию этого класса в файле ActiveXControlCtrl.cpp.

```

```

class CActiveXControlCtrl : public COleControl
{
    DECLARE_DYNCREATE(CActiveXControlCtrl)

public:
    CBitmap m_SpaceImage1, m_SpaceImage2, *m_CurrentImage;

    // Конструктор
public:
    CActiveXControlCtrl();

    // Переопределения
public:
    virtual void OnDraw(CDC* pdc, const CRect& rcBounds,
        const CRect& rcInvalid);
    virtual void DoPropExchange(CPropExchange* pPX);
    virtual void OnResetState();

    // Реализация
protected:
    ~CActiveXControlCtrl();

    DECLARE_OLECREATE_EX(CActiveXControlCtrl) // Фабрика классов
    DECLARE_OLETYPELIB(CActiveXControlCtrl) // GetTypeInfo
    DECLARE_PROPPAGEIDS(CActiveXControlCtrl) // ID страниц свойств
    DECLARE_OLECTLTYPE(CActiveXControlCtrl) // Имя типа и статус

    // Карты сообщений
    DECLARE_MESSAGE_MAP()

    // Карты рассылки
    DECLARE_DISPATCH_MAP()

    afx_msg void AboutBox();

    // Карты событий
    DECLARE_EVENT_MAP()

    // ID рассылок и событий
public:
    enum {
        dispidDisplayColor = 1
    };
    afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
protected:
    void OnDisplayColorChanged(void);
    BOOL m_DisplayColor;
};

```

---

#### Листинг 25.4.

```

// ActiveXControlCtrl.cpp : Реализация класса ActiveX Control
// CActiveXControlCtrl.

#include "stdafx.h"
#include "ActiveXControl.h"

```

```

#include "ActiveXControlCtrl.h"
#include "ActiveXControlPropPage.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

IMPLEMENT_DYNCREATE(CActiveXControlCtrl, COleControl)

// Карта сообщений

BEGIN_MESSAGE_MAP(CActiveXControlCtrl, COleControl)
    ON_OLEVERB(AFX_IDS_VERB_PROPERTIES, OnProperties)
    ON_WM_LBUTTONDOWN()
END_MESSAGE_MAP()

// Карта рассылки

BEGIN_DISPATCH_MAP(CActiveXControlCtrl, COleControl)
    DISP_FUNCTION_ID(CActiveXControlCtrl, "AboutBox",
        DISPID_ABOUTBOX, AboutBox, VT_EMPTY, VTS_NONE)
    DISP_STOCKPROP_BACKCOLOR()
    DISP_PROPERTY_NOTIFY(CActiveXControlCtrl,
        DisplayColor, m_DisplayColor, OnDisplayColorChanged,
        VT_BOOL)
END_DISPATCH_MAP()

// Карта событий

BEGIN_EVENT_MAP(CActiveXControlCtrl, COleControl)
END_EVENT_MAP()

// Страницы свойств

// TODO: Добавляйте новые страницы свойств по мере необходимости.
// Одновременно увеличивайте значение счетчика строкой ниже.
BEGIN_PROPPAGEIDS(CActiveXControlCtrl, 2)
    PROPPAGEID(CActiveXControlPropPage::guid)
    PROPPAGEID(CLSID_CColorPropPage)
END_PROPPAGEIDS(CActiveXControlCtrl)

// Инициализация фабрики классов

IMPLEMENT_OLECREATE_EX(CActiveXControlCtrl,
    "ACTIVEXCONTROL.ActiveXControlCtrl.1",
    0xf6e1e7cc, 0x3075, 0x46d9, 0xba, 0x33, 0x67,
    0xa7, 0xc8, 0x1d, 0x7e, 0xb4)

// ID библиотеки типов и версия

IMPLEMENT_OLETYPELIB(CActiveXControlCtrl, _tlid, _wVerMajor,
    _wVerMinor)

// Интерфейс

const IID BASED_CODE IID_DActiveXControl =
    { 0x7DBC13C8, 0xBDF9, 0x440A, { 0x9C, 0xF, 0x21, 0xC,
        0xF4, 0x2A, 0xE1, 0x50 } };

```

```

const IID BASED_CODE IID_DActiveXControlEvents =
    { 0xD0720DAD, 0x4EF, 0x44DA, { 0xB8, 0x6C, 0x24, 0x65,
      0x4B, 0x5F, 0x22, 0x92 } };

// Информация об элементах управления

static const DWORD BASED_CODE _dwActiveXControlOleMisc =
    OLEMISC_ACTIVATEWHENVISIBLE |
    OLEMISC_SETCLIENTSITEFIRST |
    OLEMISC_INSIDEOUT |
    OLEMISC_CANTLINKINSIDE |
    OLEMISC_RECOMPOSEONRESIZE;

IMPLEMENT_OLECTLTYPE(CActiveXControlCtrl, IDS_ACTIVEXCONTROL,
    _dwActiveXControlOleMisc)

// CActiveXControlCtrl::CActiveXControlCtrlFactory::UpdateRegistry -
// Добавление/удаление записей в реестре о CActiveXControlCtrl

BOOL CActiveXControlCtrl::CActiveXControlCtrlFactory::UpdateRegistry
    (BOOL bRegister)
{
    // TODO: Убедитесь, что ваш элемент управления соответствует
    // модели разделенных потоков.
    // За дополнительной информацией обратитесь к MFC TechNote 64.
    // Если элемент не соответствует данной модели, измените
    // нижеследующий код, заменив 6-й параметр с
    // afxRegApartmentThreading на 0.

    if (bRegister)
        return AfxOleRegisterControlClass(
            AfxGetInstanceHandle(),
            m_clsid,
            m_lpszProgID,
            IDS_ACTIVEXCONTROL,
            IDB_ACTIVEXCONTROL,
            afxRegApartmentThreading,
            _dwActiveXControlOleMisc,
            _tlid,
            _wVerMajor,
            _wVerMinor);
    else
        return AfxOleUnregisterClass(m_clsid, m_lpszProgID);
}

CActiveXControlCtrl::CActiveXControlCtrl()
{
    InitializeIIDs(&IID_DActiveXControl, &IID_DActiveXControlEvents);
    // TODO: Добавьте сюда собственный код.

    m_SpaceImage1.LoadBitmap (IDB_BITMAP1);
    m_SpaceImage2.LoadBitmap (IDB_BITMAP2);
    m_CurrentImage = &m_SpaceImage1; // первоначальное изображение
}

```

```

CActiveXControlCtrl::~CActiveXControlCtrl()
{
    // TODO: Добавьте сюда код очистки.
}

void CActiveXControlCtrl::OnDraw(
    CDC* pdc, const CRect& rcBounds, const CRect& rcInvalid)
{
    // TODO: Замените следующий код собственным кодом прорисовки.

    CBrush Brush (TranslateColor (GetBackColor ()));
    pdc->FillRect (rcBounds, &Brush);

    BITMAP MyBitmap;
    CDC MemDC;

    MemDC.CreateCompatibleDC (NULL);
    MemDC.SelectObject (*m_CurrentImage);
    m_CurrentImage->GetObject (sizeof (MyBitmap), &MyBitmap);
    pdc->BitBlt ((rcBounds.right - MyBitmap.bmWidth) / 2,
        (rcBounds.bottom - MyBitmap.bmHeight) / 2,
        MyBitmap.bmWidth, MyBitmap.bmHeight,
        &MemDC, 0, 0, SRCCOPY);

    if (m_DisplayColor)
    {
        CBrush *pOldBrush = (CBrush *)
            pdc->SelectStockObject (NULL_BRUSH);
        CPen Pen (PS_SOLID | PS_INSIDEFRAME, 10, RGB(0, 0, 0));
        CPen *pOldPen = pdc->SelectObject (&Pen);

        pdc->Rectangle (rcBounds);

        pdc->SelectObject (pOldPen);
        pdc->SelectObject (pOldBrush);
    }
}

// CActiveXControlCtrl::DoPropExchange - поддержка

void CActiveXControlCtrl::DoPropExchange(CPropExchange* pPX)
{
    ExchangeVersion(pPX, MAKELONG(_wVerMinor, _wVerMajor));
    COleControl::DoPropExchange(pPX);

    // TODO: Вызывайте PX_ функции
    // для каждого пользовательского свойства.

    PX_Bool (pPX, _T("DisplayColor"), m_DisplayColor, FALSE);
}

// Сброс элемента в состояние по умолчанию

void CActiveXControlCtrl::OnResetState()
{
    // Сброс стандартных элементов из DoPropExchange

```

```

        COleControl::OnResetState();

        // TODO: Сбросьте оставшиеся элементы.
    }

void CActiveXControlCtrl::AboutBox()
{
    CDialog dlgAbout(IDD_ABOUTBOX_ACTIVEXCONTROL);
    dlgAbout.DoModal();
}

// Обработчики сообщений класса CActiveXControlCtrl

void CActiveXControlCtrl::OnLButtonUp(UINT nFlags, CPoint point)
{
    // TODO: Добавьте сюда собственный код обработчика
    // или вызов стандартного

    if (m_CurrentImage == &m_SpaceImage1)
        m_CurrentImage = &m_SpaceImage2;
    else
        m_CurrentImage = &m_SpaceImage1;

    InvalidateControl ();

    COleControl::OnLButtonUp(nFlags, point);
}

void CActiveXControlCtrl::OnDisplayColorChanged(void)
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState());

    // TODO: Добавьте сюда собственный код обработчика

    InvalidateControl ();

    SetModifiedFlag();
}

```

---

### Листинг 25.5.

```

#pragma once

// ActiveXControlPropPage.h : Объявление класса страниц свойств
// CActiveXControlPropPage.
// реализацию этого класса смотрите в файле ActiveXControlPropPage.cpp.

class CActiveXControlPropPage : public COlePropertyPage
{
    DECLARE_DYNCREATE(CActiveXControlPropPage)
    DECLARE_OLECREATE_EX(CActiveXControlPropPage)

    // Конструктор
public:
    CActiveXControlPropPage();

```

```

// Данные для диалога
enum { IDD = IDD_PROPPAGE_ACTIVEXCONTROL };

// Реализация
protected:
    virtual void DoDataExchange(CDataExchange* pDX);
    // поддержка DDX/DDV

// Карты сообщений
protected:
    DECLARE_MESSAGE_MAP()
public:
    BOOL m_DisplayColor;
};

```

---

## Листинг 25.6.

```

// ActiveXControlPropPage.cpp:
// Реализация CActiveXControlPropPage - класса страниц свойств.

#include "stdafx.h"
#include "ActiveXControl.h"
#include "ActiveXControlPropPage.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

IMPLEMENT_DYNCREATE(CActiveXControlPropPage, COlePropertyPage)

// Карта сообщений

BEGIN_MESSAGE_MAP(CActiveXControlPropPage, COlePropertyPage)
END_MESSAGE_MAP()

// Инициализация фабрики классов

IMPLEMENT_OLECREATE_EX(CActiveXControlPropPage,
    "ACTIVEXCONTROL.ActiveXControlPropPage.1",
    0xe7e521a9, 0x53c4, 0x40a2, 0xa2, 0x26, 0x4,
    0x8f, 0x6d, 0xc5, 0x55, 0x15)

// CActiveXControlPropPage::CActiveXControlPropPageFactory::
// UpdateRegistry -
// Добавка/удаление в реестре записей о CActiveXControlPropPage

BOOL
CActiveXControlPropPage::CActiveXControlPropPageFactory::UpdateRegistry
(BOOL bRegister)
{
    if (bRegister)
        return AfxOleRegisterPropertyPageClass(AfxGetInstanceHandle(),
            m_clsid, IDS_ACTIVEXCONTROL_PPG);
    else
        return AfxOleUnregisterClass(m_clsid, NULL);
}

```



```

CActiveXControlPropPage::CActiveXControlPropPage() :
    COlePropertyPage(IDD, IDS_ACTIVEXCONTROL_PPG_CAPTION)
    , m_DisplayColor(FALSE)
    {
    }

// CActiveXControlPropPage::DoDataExchange - Перемещение данных
// между свойствами и страницами

void CActiveXControlPropPage::DoDataExchange(CDataExchange* pDX)
{
    DDP_PostProcessing(pDX);
    DDX_Check(pDX, IDC_DISPLAYCOLOR, m_DisplayColor);
}

// Обработчики сообщений класса CActiveXControlPropPage

```

## Программа-контейнер ActiveX-элемента

Создадим с помощью мастера Application Wizard и других инструментов Visual C++ программу-контейнер элемента ActiveX, называющуюся ActiveXContainer, предназначенную специально для отображения созданного ранее элемента ActiveX и взаимодействия с ним. Приложение ActiveXContainer построено на основе диалогового окна, в котором отображается элемент ActiveX. Программа считывает и устанавливает свойство BackColor и позволяет поочередно показывать оба изображения, использованных в создании элемента ActiveX.

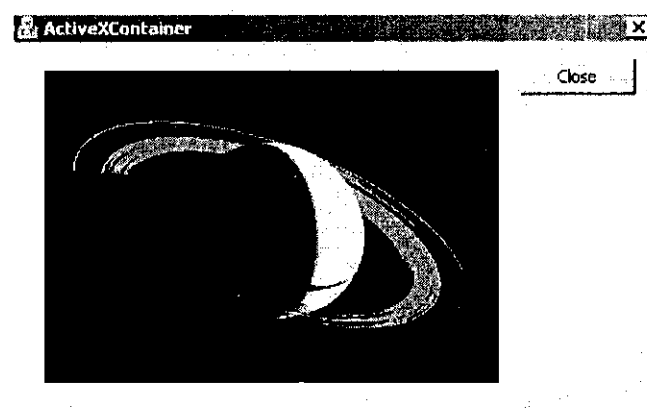
Процедура создания файлов с исходным текстом программы ActiveXControl состоит из следующих шагов.

1. Выберите в окне New Project создание нового приложения MFC Application.
2. В окне Application Wizard на вкладке Application Type выберите опцию Dialog Based. Остальные установки можно оставить без изменений и нажать кнопку Finish.

Завершив создание исходных файлов программы ActiveXContainer, мастер Application Wizard открывает главное диалоговое окно программы IDD\_ACTIVEXCONTROL\_DIALOG. Если этот ресурс еще не открыт, откройте его. Для редактирования окна программы выполните следующие действия.

1. Элемент управления с надписью "TODO" и кнопку OK необходимо удалить.
2. Свойство Caption кнопки Cancel следует заменить на Close.
3. Щелкните правой кнопкой мыши на пустом месте в панели инструментов, в контекстном меню выберите пункт Customize ToolBox..., в открывшемся окне поставьте флажок напротив пункта ActiveXControl Control, нажмите кнопку OK. В панели инструментов появится элемент OCX.
4. В диалоговое окно добавьте элемент ActiveX, щелкнув на кнопке OCX в нижней части панели инструментов, а затем – на нужном месте в диалоговом окне.
5. Растяните элемент в окне до размера, достаточного для отображения использованного при создании элемента рисунка и тонкой каймы вокруг него.
6. Щелкнув на элементе ActiveX правой кнопкой мыши и выбрав команду Properties, измените его свойства. При желании можете изменить цвет фона элемента, используя его свойство BackColor. Оставьте неизменными стандартные значения всех остальных свойств, включая идентификатор диалогового окна IDC\_ACTIVEXCONTAINERCTRL1.

Можно приступить к построению, выполнению и проверке работы программы ActiveXContainer. Каждый раз при щелчке на элементе ActiveX программа элемента выполняет переключение между версиями изображения. Окно программы показано на рисунке.



## Текст программы ActiveXContainer

Исходные тексты программы ActiveXContainer приведены в листингах 25.7—25.10.

---

### Листинг 25.7.

```
// ActiveXContainer.h : главный заголовочный файл приложения
//

#pragma once

#ifndef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"           // основные символы

// CActiveXContainerApp:
// Смотрите реализацию этого класса в файле ActiveXContainer.cpp
//

class CActiveXContainerApp : public CWinApp
{
public:
    CActiveXContainerApp();

// Переопределения
public:
    virtual BOOL InitInstance();

// Реализация

    DECLARE_MESSAGE_MAP()
    afx_msg void OnBnClickedFrame();
};

extern CActiveXContainerApp theApp;
```

---

### Листинг 25.8.

```
// ActiveXContainer.cpp : Определяет поведение классов приложения.
//

#include "stdafx.h"
#include "ActiveXContainer.h"
#include "ActiveXContainerDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CActiveXContainerApp

BEGIN_MESSAGE_MAP(CActiveXContainerApp, CWinApp)
    ON_COMMAND(ID_HELP, CWinApp::OnHelp)
    ON_BN_CLICKED(IDFRAME, OnBnClickedFrame)
END_MESSAGE_MAP()

// Конструктор класса CActiveXContainerApp

CActiveXContainerApp::CActiveXContainerApp()
{
    // TODO: добавьте сюда собственный код конструктора.
    // Поместите весь существенный код инициализации
    // в функцию InitInstance.
}

// Единственный объект класса CActiveXContainerApp

CActiveXContainerApp theApp;

// Инициализация CActiveXContainerApp

BOOL CActiveXContainerApp::InitInstance()
{
    CWinApp::InitInstance();

    AfxEnableControlContainer();

    CActiveXContainerDlg dlg;
    m_pMainWnd = &dlg;
    INT_PTR nResponse = dlg.DoModal();
    if (nResponse == IDOK)
    {
        // TODO: Поместите сюда собственный код обработчика
    }
    else if (nResponse == IDCANCEL)
    {
        // TODO: Поместите сюда собственный код обработчика
    }

    return FALSE;
}
```

```

void CActiveXContainerApp::OnBnClickedFrame()
{
    // TODO: Поместите сюда собственный код обработчика
}

```

---

#### Листинг 25.9.

```

// ActiveXContainerDlg.h : файл заголовков
//

#pragma once

// диалог CActiveXContainerDlg
class CActiveXContainerDlg : public CDialog
{
// Конструктор
public:
    CActiveXContainerDlg(CWnd* pParent = NULL);
    // стандартный документ

// Данные для диалога
    enum { IDD = IDD_ACTIVEXCONTAINER_DIALOG };

protected:
    virtual void DoDataExchange(CDataExchange* pDX);
// Поддержка DDX/DDV

// Реализация
protected:
    HICON m_hIcon;

    // Сгенерированные функции карты сообщений
    virtual BOOL OnInitDialog();
    afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
    afx_msg void OnPaint();
    afx_msg HCURSOR OnQueryDragIcon();
    DECLARE_MESSAGE_MAP()
};

```

---

#### Листинг 25.10.

```

// ActiveXContainerDlg.cpp : файл реализации
//

#include "stdafx.h"
#include "ActiveXContainer.h"
#include "ActiveXContainerDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CAboutDlg диалог, используемый в App About

```

```

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

    // Данные для диалога
    enum { IDD = IDD_ABOUTBOX };

protected:
    virtual void DoDataExchange(CDataExchange* pDX);
    // поддержка DDX/DDV

    // Реализация
protected:
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
END_MESSAGE_MAP()

// Диалог CActiveXContainerDlg

CActiveXContainerDlg::CActiveXContainerDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CActiveXContainerDlg::IDD, pParent)
{
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CActiveXContainerDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CActiveXContainerDlg, CDialog)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    //})AFX_MSG_MAP
END_MESSAGE_MAP()

// Обработчики сообщений класса CActiveXContainerDlg

BOOL CActiveXContainerDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);
}

```

```

CMenu* pSysMenu = GetSystemMenu(FALSE);
if (pSysMenu != NULL)
{
    CString strAboutMenu;
    strAboutMenu.LoadString(IDS_ABOUTBOX);
    if (!strAboutMenu.IsEmpty())
    {
        pSysMenu->AppendMenu(MF_SEPARATOR);
        pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
                               strAboutMenu);
    }
}

// Установка значков. Среда делает это автоматически,
// если главное окно приложения не диалоговое
SetIcon(m_hIcon, TRUE);           // большой значок
SetIcon(m_hIcon, FALSE);          // маленький значок

// TODO: добавьте сюда дополнительный код обработчика

return TRUE;
}

void CActiveXContainerDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
    if ((nID & 0xFFF0) == IDM_ABOUTBOX)
    {
        CAboutDlg dlgAbout;
        dlgAbout.DoModal();
    }
    else
    {
        CDialog::OnSysCommand(nID, lParam);
    }
}

// Если в окне есть кнопка минимизации, нижеследующий код требуется
// для отображения значка. Для MFC-приложений с моделью
// документ/представление это делает среда.

void CActiveXContainerDlg::OnPaint()
{
    if (IsIconic())
    {
        CPaintDC dc(this); // контекст устройства для рисования

        SendMessage(WM_ICONERASEBKGND, reinterpret_cast<WPARAM>
                     (dc.GetSafeHdc()), 0);

        // Центровка значка
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
        int x = (rect.Width() - cxIcon + 1) / 2;
        int y = (rect.Height() - cyIcon + 1) / 2;
    }
}

```

```

        // Отображение
        dc.DrawIcon(x, y, m_hIcon);
    }
    else
    {
        CDialog::OnPaint();
    }
}

// Получение курсора при перетаскивании окна
HCURSOR CActiveXContainerDlg::OnQueryDragIcon()
{
    return static_cast<HCURSOR>(m_hIcon);
}

```

## Резюме

Вы научились создавать элементы ActiveX, а также программы-контейнеры для отображения таких элементов и взаимодействия с ними.

- *Элемент ActiveX.* Элемент ActiveX – это переносимый программный модуль, который можно вставить в любую программу-контейнер ActiveX. *Программа-контейнер* отображает элемент ActiveX как часть средств интерфейса и взаимодействует с ним с помощью свойств, методов и событий. *Свойство* – это атрибут элемента, который программа-контейнер может читать или изменять. *Метод* – это функция, предоставляемая элементом, которая может быть вызвана программой-контейнером. *Событие* происходит внутри элемента и, вызывая обработчик программы-контейнера, уведомляет эту программу о произошедшем событии.
- *Создание ActiveX-элемента.* Можно быстро создать элемент ActiveX на основе средств библиотеки MFC, выбрав тип проекта MFC ActiveX Control в окне New Project. При этом мастер создает исходные файлы программы элемента ActiveX. Для определения свойств, методов и событий программы элемента ActiveX используется вкладка Resource View. Каждый указанный компонент может представляться в двух вариантах: *стандартном* и *пользовательском*. Для реализации стандартных компонентов большая часть кода предоставляется библиотекой MFC, а для пользовательских необходимо писать собственный код. В элемент ActiveX можно ввести одну или несколько *страниц свойств*, содержащих элементы управления для установки начальных значений свойств элемента при разработке программы-контейнера. При построении элемента ActiveX генерируется файл .osx, который содержит код и ресурсы элемента. Этот файл должен быть зарегистрирован для получения доступа к нему из программы-контейнера.
- *Программа-контейнер ActiveX-элемента.* Любая программа, созданная мастером Application Wizard, может быть контейнером элемента ActiveX при условии, что в диалоговом окне мастера Application Wizard выбрана опция ActiveX Controls. Чтобы добавить элемент ActiveX в программу-контейнер, необходимо добавить в панель инструментов Controls редактора диалоговых окон кнопку для данного элемента. Для этого выбирается команда Components And Controls... в меню Project подменю Add to Project, а затем – имя элемента. Элемент ActiveX добавляется в диалоговое окно при щелчке на соответствующей кнопке панели инструментов Controls. Затем используется методика, подобная применяемой для добавления и настройки стандартных элементов управления Windows. После этого класс диалогового окна вызывает методы элемента (для получения или изменения свойств элемента).

# Предметный указатель

<b>A</b>	
ActiveX .....	734
встроенная поддержка программами .....	138
методы элемента .....	734
программа-контейнер .....	749
свойства элемента .....	734
события элемента .....	734
создание программы-контейнера .....	749
создание элемента .....	734
элемент .....	734

<b>G</b>	
GDI .....	477

<b>M</b>	
MFC-классы .....	
ограничения на использование потоками .....	632
MSDN .....	23

<b>O</b>	
OLE .....	678
автоматизация .....	680
внедрение объекта .....	679
клиент автоматизации .....	680
методы .....	680
полностью открытое редактирование .....	679
программа-контейнер .....	678
программа-сервер .....	678
редактирование внедренного объекта .....	680
редактирование на месте .....	679
свойства .....	680
связывание объекта .....	679
сервер автоматизации .....	680

<b>V</b>	
Visual Studio .....	
выбор сочетаний клавиш .....	20
запуск .....	17
инсталляция .....	14
компоненты .....	16
создание проекта .....	17
состав пакета .....	16
сочетания клавиш .....	20
требования к системе .....	14
установка .....	14

<b>W</b>	
Win32 API .....	258
функция CloseClipboard .....	666
функция CloseHandle .....	636
функция CreateEvent .....	636
функция CreateFileMapping .....	660
функция CreateMutex .....	634
функция CreatePipe .....	659
функция CreateProcess .....	655
функция CreateSemaphore .....	636
функция CreateThread .....	628
функция DeleteCriticalSection .....	636
функция DestroyCaret .....	454
функция EmptyClipboard .....	666
функция EnterCriticalSection .....	636
функция EnumClipboardFormats .....	668
функция ExitProcess .....	656

функция GetAsyncKeyState .....	450
функция GetClipboardData .....	668
функция GetCommandLine .....	656
функция GetExitCodeProcess .....	656
функция GetExitCodeThread .....	631
функция GetKeyState .....	450
функция GetPriorityClipboardFormat .....	668
функция GlobalAlloc .....	664
функция GlobalFree .....	665
функция GlobalLock .....	664, 669
функция GlobalUnlock .....	664, 669
функция InitializeCriticalSection .....	636
функция IsClipboardFormatAvailable .....	668
функция LeaveCriticalSection .....	636
функция lstrcpy .....	665
функция MapViewOfFile .....	661
функция OpenMutex .....	658
функция OpenProcess .....	658
функция PulseEvent .....	636
функция ReadFile .....	660
функция ReleaseMutex .....	634
функция ReleaseSemaphore .....	636
функция ResetEvent .....	636
функция SetClipboardData .....	666
функция SetEvent .....	636
функция SetPriorityClass .....	656
функция SetStdHandle .....	660
функция TerminateProcess .....	656
функция TerminateThread .....	630
функция WaitForMultipleObject .....	636
функция WaitForMultipleObjects .....	637
функция WaitForSingleObject .....	634
функция WriteFile .....	660

<b>A</b>	
Автоматизация в OLE .....	680
Анимация с помощью функции BitBlt .....	567
Аппаратные ошибки .....	128
Архитектура Document/View Architecture Support .....	137
Асинхронное выполнение потоков .....	633

<b>Б</b>	
Базы данных .....	
поддержка приложениями .....	137
Битовые операции .....	563
Буфер обмена .....	661
данные зарегистрированных форматов .....	675
извлечение растрового изображения .....	673
извлечение текста .....	668
кнопки панели инструментов .....	662
команды управления .....	662
обмен текстом .....	663
прием растрового изображения .....	671
прием текста .....	663
программа-получатель .....	663
работа с графикой .....	671
формат данных .....	666

<b>В</b>	
Вешка разбивки .....	266
Виртуальная функция .....	83
Виртуальный код клавиши .....	449
Вкладка Class View .....	23, 82
возможности .....	82



отображение/сокрытие.....	19
просмотр иерархии классов.....	82
Вкладка Solution Explorer.....	23
отображение/сокрытие.....	19
Вкладки диалогового окна.....	361
Вложенные исключения.....	126
Внедрение объекта в OLE.....	679
Возврат стека.....	124
Временная приостановка потока.....	630
Встраивание функции, настройка.....	37
Вторичный поток.....	628
Выбор шрифта.....	432, 444
Выделение памяти оператором new.....	50
Выполнение программы.....	141
Высота символа.....	432

## Г

Генерация исходного кода.....	136
Главное окно	
многодокументного приложения.....	413

## Д

Данные	
учет изменений.....	231
флаг изменений.....	231
чтение/запись.....	228
Деление на ноль.....	128
Дескриптор мьютекса.....	635
Дескриптор объекта	
создание дубликата.....	659
Дескрипторы объектов.....	657
Деструктор.....	66
в шаблоне класса.....	117
вызов.....	67
вызов для массива.....	67
Диалоговая панель.....	290, 300
Диалоговое окно	
добавление в программу.....	324
добавление элементов.....	326
значок системного меню.....	325
кнопка закрытия.....	324
немодальное.....	360
общего назначения.....	380
отображение.....	340
порядок обхода элементов.....	328
с вкладками.....	361
свойства.....	324
системное меню.....	325
создание управляющего класса.....	329
управляющий класс.....	329
формирование.....	383
функции управления.....	334
элементы.....	326
Динамическое связывание.....	85
Документ	
обработка разных типов.....	414
печать.....	587
предварительный просмотр.....	587
составной.....	678
шаблон.....	157
Доступ	
к наследуемым переменным.....	79
к полям класса.....	58
к членам класса.....	57
Дочернее окно.....	413
Дружественная (friend) функция.....	80
Дружественный (friend) класс.....	80

## З

Завершение потока.....	630
Задержка при выполнении	
индикация.....	172
Запуск новых потоков.....	628
Значки.....	569
импортирование.....	570
конструирование.....	570
отображение.....	570
создание.....	570
Значок программы, создание.....	173

## И

Иерархия классов.....	81
Изменение исходного кода.....	138
Изменение класса окна.....	266
Изображение растровое.....	557
Изображения	
обмен через буфер.....	671
Инициализация массивов.....	107
Инкапсуляция.....	57
Инсталляция Visual Studio	
автоматический вход в систему.....	14
аппаратура.....	14
восстановление после неудачи.....	15
выбор компонентов.....	15
выбор места для установки.....	15
лицензионное соглашение.....	15
начало.....	14
обновление системы.....	14
операционная система.....	14
регистрация.....	15
справочная система.....	15
требования.....	14
Инструмент	
выбор.....	478
кисть.....	476
перо.....	476
Инструменты рисования.....	476
Интервал между строками.....	432
Интерфейс	
MDI.....	411
мультидокументный.....	411
однооконный.....	142
Информация о приложении, окно.....	138
Исключение (exception).....	119
Win32.....	128
вложенные.....	126
выбор типа обработчика.....	124
модель обработки.....	123
обработчик.....	119
оператор catch.....	119, 123
оператор throw.....	119
оператор try.....	119
размещение обработчиков.....	124
способы обработки.....	123
структурированное.....	128
языка C.....	128
Исходный код	
изменение.....	138
редактирование.....	138
создание.....	136

## К

Канал, создание.....	659
Карта сообщений.....	168
Кисть.....	476

выбор.....	478	Класс CCmdUI.....	209
выбор узора.....	480	функция Enable.....	209, 299
узор.....	480	функция SetCheck.....	209, 299
Клавиатура		функция SetRadio.....	209, 299
ввод символов.....	447	функция SetText.....	209, 299
Клавиша		Класс CDC	
виртуальный код.....	449	функция Arc.....	502
системная.....	447	функция BitBlt.....	562, 563, 565
текущее состояние.....	450	функция DrawText.....	434
Класс		функция Ellipse.....	506
CRectangle.....	55	функция ExtTextOut.....	434
встроенные функции.....	68	функция FillRect.....	564
главного окна.....	143, 413	функция GetClipboard.....	433
деструктор.....	66	функция GetDeviceCaps.....	563, 592, 595
диалогового окна.....	383	функция GetLength.....	446
документа.....	138, 143, 413	функция GetPixel.....	485
доступ к наследуемым переменным.....	79	функция GetTextMetrics.....	432
доступ к членам.....	57	функция GrayString.....	434
дочернего окна.....	413	функция LineTo.....	170, 501
дружественный (friend).....	80	функция LPtoDP.....	265
закрытые (private) члены.....	56	функция MoveTo.....	170, 501
иерархия.....	81	функция PatBlt.....	564
компонента сервера.....	684	функция Pie.....	507
конструктор.....	60	функция PolyBezier.....	503
конструктор по умолчанию.....	61	функция PolyBezierTo.....	504
копия.....	56	функция Polygon.....	507
методы.....	55, 68	функция PolylineTo.....	502
множественное наследование.....	82	функция polyling.....	501
наследование.....	76	функция PolyPolygon.....	507
объект.....	56	функция PolyPolyline.....	502
окна редактирования на месте.....	684	функция Rectangle.....	506
определение.....	54	функция RoundRect.....	506
открытые (public) члены.....	56	функция SelectObject.....	432
перегруженный конструктор.....	62	функция SelectStockObject.....	444, 478
переменные-члены.....	55	функция SetArcDirection.....	502
поля.....	55	функция SetBkColor.....	433
представитель.....	56	функция SetBkMode.....	433
представления.....	140, 143, 162, 188, 414	функция SetMapMode.....	482
приложения.....	143, 382, 413	функция SetPixel.....	485
размещение определений в программе.....	69	функция SetPixelV.....	485
создание объекта.....	56	функция SetROP2.....	170, 504
создание производного.....	76	функция SetStretchBltMode.....	569
статические члены.....	72	функция SetTextAlign.....	433
управляющий диалоговым окном.....	329	функция SetTextCharacterExtra.....	433
функции доступа к полям.....	58	функция SetTextColor.....	442
функции-члены.....	55, 68	функция SetTextColors.....	433
шаблон.....	112	функция StretchBlt.....	568
экземпляр.....	55	функция TabbedTextOut.....	434
элемента управления.....	332	функция TextOut.....	434
Класс CArchive		Класс CDialog	
оператор <<.....	258	функции управления окнами.....	334
оператор >>.....	258	функция DoModal.....	387
функция Read.....	258	функция OnPaint.....	386
функция Serialize.....	258	Класс CDocument	
функция Write.....	258	функция DeleteContents.....	210
Класс CBitmap		функция SetModifiedFlag.....	231
функция CreateBitmap.....	557	функция UpdateAllViews.....	208
функция CreateCompatibleBitmap.....	557	Класс CEditView	
функция LoadBitmap.....	557	печать документов.....	588
функция LoadOEMBitmap.....	557	Класс CFile	
Класс CBrush		258	
функция CreateHatchBrush.....	480	Класс CFont	
функция CreatePatternBrush.....	480	функция CreateFontIndirect.....	443
функция CreateSolidBrush.....	480	функция DeleteObject.....	443
Класс CClientDC		Класс CFontDialog	
функция DPtoLP.....	262	функция GetColor.....	442
Класс CCmdTarget		функция GetFaceName.....	442
функция BeginWaitCursor.....	172	функция GetSize.....	442
функция EndWaitCursor.....	172	Класс CFormView.....	393
		Класс CFrameWnd	

функция DockControlBar .....	294	функция GetThreadPriority .....	631
функция OnCreateClient .....	267	функция SetThreadPriority .....	631
Класс CGdiObject .....		функция SuspendThread .....	630
функция Attach .....	557	Класс CWnd .....	
функция GetObject .....	443	функции управления окнами .....	334
Класс CLine .....	203	функция ClientToScreen .....	168
функция Draw .....	205	функция CreateSolidCaret .....	454
Класс CMenu .....		функция EnableWindow .....	333
функция AppendMenu .....	563	функция GetClientRect .....	168
функция SetMenuItemBitmap .....	563	функция GetDlgItem .....	333
Класс CObArray .....	204	функция GetParentFrame .....	489
функция Serialize .....	228	функция HideCaret .....	455
Класс CObject .....	204	функция InvalidateRect .....	271
Класс COleTemplateServer .....		функция IsIconic .....	489
функция ConnectTemplate .....	682	функция OpenClipboard .....	664
функция OnGetEmbeddedItem .....	684	функция SendMessage .....	451
функция RegisterAll .....	683	функция SetCapture .....	167
функция UpdateRegistry .....	683	функция SetCaretPos .....	454
Класс CPen .....	478	функция ShowCaret .....	454
Класс CPrintInfo .....		функция UpdateData .....	330
функция SetMaxPage .....	587	Класс документа .....	203
функция SetMinPage .....	592	Ключевое слово .....	
Класс CRect .....	264	extern .....	31
функция InflateRect .....	385	Ключевые слова .....	29
функция IntersectRect .....	265	VC7 .....	30
функция PtInRect .....	264	расширений C++ .....	30
Класс CScratchBookDoc .....		Кнопка .....	
функция AddLine .....	205, 231	закрывающего диалогового окна .....	324
функция GetLine .....	205	максимизации .....	138
функция GetNumLines .....	205	минимизации .....	138
Класс CScratchBookView .....		разворачивания .....	138
функция OnLButtonUp .....	206	сворачивания .....	138
Класс CScrollView .....	260	Код .....	
функция GetTotalSize .....	263	возврата .....	629
функция OnInitialUpdate .....	262	исходный .....	136
функция OnPrepareDC .....	261, 476	охраняемый раздел .....	120
функция OnUpdate .....	270	Код возврата процесса .....	656
функция SetScaleToFitSize .....	263	Код растровой операции .....	564
функция SetScrollSizes .....	263	Комментарии в программах .....	33
функция UpdateAllViews .....	268	Консольная программа .....	18
Класс CSingleDocTemplate .....	157	Константа .....	
конструктор .....	157	CBRS_FLYBY .....	294
Класс CSplitterWnd .....		CBRS_GRIPPER .....	294
переменная m_SplitterWnd .....	267	CBRS_SIZE_DYNAMIC .....	294
функция Create .....	267	CBRS_TOOLTIPS .....	294
Класс CToolBar .....		CBRS_TOP .....	294
функция CreateEx .....	294	WS_CHILD .....	294
функция LoadToolBar .....	294	WS_VISIBLE .....	294
Класс CTypedPtrArray .....		объект .....	63
функция GetAt .....	209	Константы .....	
Класс CView .....		инициализация .....	43
функция DoPreparePrinting .....	587	объявление .....	43
функция OnBeginPrinting .....	591	указатель на .....	44
функция OnDraw .....	140, 203	Конструктор .....	60
функция OnEndPrinting .....	591	в шаблоне класса .....	116
функция OnPaint .....	338	вызов .....	67
функция OnPrepareDC .....	453	вызов для массива .....	67
функция OnPrint .....	591	инициализация переменных в .....	64
функция PreCreateView .....	175	класса .....	60
Класс CWinApp .....	157	класса-наследника .....	78
функция AddDocTemplate .....	157	копирования .....	100
функция EnableShellOpen .....	230	перегруженный .....	62
функция LoadIcon .....	570	по умолчанию .....	61
функция LoadStandardCursor .....	164	преобразования .....	100, 102
функция LoadStdProfileSettings .....	158	производного класса .....	78
функция RegisterShellTypes .....	230	Контекст устройства .....	475
функция SetRegistryKey .....	158	Конфигурация проекта .....	24, 141
Класс CWinThread .....		Координаты .....	

логические .....	260, 482
устройства .....	260, 482
фактические .....	260
Критические секции .....	635
Курсор .....	453
Курсоры мыши .....	164

## Л

Линии	
лекальные .....	501
прямые .....	501
регулярные кривые .....	501
рисование отрезков .....	501
Линия разбивки .....	266
Логические координаты .....	260, 482

## М

Макрос	
DECLARE_SERIAL .....	229
IMPLEMENT_SERIAL .....	229
RGB .....	479
Макрокоманда	
DECLARE_SERIAL .....	229
IMPLEMENT_SERIAL .....	229
Максимизация окна, кнопка .....	138
Массив	
инициализация .....	107
создание оператором new .....	51
уничтожение оператором delete .....	51
Мастер	
Application Wizard .....	415
Win32 Console Application .....	18
Мастер Application Wizard .....	136, 585
вкладка Advanced Features .....	138
вкладка Application Type .....	137
вкладка Compound Document Support .....	137
вкладка Database support .....	137
вкладка Document Template Strings .....	137
вкладка Generated Classes .....	138
вкладка Overview .....	137
вкладка User Interface Features .....	137
Мастер Event Handler Wizard .....	295
Меню	
системное .....	138, 325
удаление пунктов .....	173
Меню программы	
добавление пунктов .....	189
редактирование .....	189
Метафайл .....	684
Метод класса .....	55
Механизм OLE .....	См. OLE
Минимизация окна, кнопка .....	138
Многопоточковая программа .....	628
Модальное диалоговое окно .....	322
Мультидокументный интерфейс .....	411
классы программы .....	413
Мышь	
курсоры .....	164
проверка состояния клавиш .....	170
Мьютекс .....	634
атрибуты защиты .....	634
дескриптор .....	635
имя .....	634
начальное состояние .....	634
период ожидания .....	635
состояние .....	635

## Н

Наследование множественное .....	82
Наследование классов .....	76
Настройка ресурсов программы .....	173
Начало представления .....	484
Некорректное обращение к памяти .....	128
Немодальное диалоговое окно .....	360

## О

Область видимости .....	34, 36
расширение .....	36
Область представления .....	143
Область просмотра	
координаты .....	260
начало .....	260
прокрутка .....	260
режим отображения .....	263
Обработка исключений	
оператор catch .....	119, 123
оператор throw .....	119
оператор try .....	119
Обработчик	
OnLButtonDown .....	166
OnLButtonUp .....	171
OnMouseMove .....	169
исключения .....	119
нажатия кнопок мыши .....	170
создание в Visual Studio .....	166
Обработчик сообщений	
WM_CREATE .....	293
Обработчики сообщений .....	164
Объект	
GDI .....	477
внедрение в OLE .....	679
встроенный .....	65
графический .....	477
дескриптор .....	657
класса главного окна .....	157
класса документа .....	157
класса представления .....	157
контекста устройства .....	140, 432, 475, 482, 587
связывание в OLE .....	679
синхронизации потоков .....	633
создание .....	56
создание по шаблону .....	114
удаление .....	56
файла памяти .....	661
шаблона сервера .....	682
Объект контекста устройства .....	261
Объявление переменной .....	34
в операторе switch .....	35
внутри блока .....	34
с инициализацией .....	35
Объявление функции .....	30
в C .....	30
в C++ .....	30
Объект-константа .....	63
Окна, сочетания клавиш .....	22
Окно	
главное .....	413
дочернее .....	413
информации о приложении .....	138
модальное .....	322
разделение .....	266
Окно диалоговое	
New Project .....	18
Print .....	588

Print Preview.....	588
Print Setup.....	588
Text Properties.....	323
добавление в программу.....	324
добавление элементов.....	326
значок системного меню.....	325
немодальное.....	360
общего назначения.....	380
отображение.....	340
порядок обхода элементов.....	328
предварительного просмотра.....	588
редактирование.....	324
с вкладками.....	361
свойства.....	324
системное меню.....	325
создание управляющего класса.....	329
управляющий класс.....	329
формирование.....	383
функции управления.....	334
элементы.....	326
Окно представления.....	143
границы рисунка.....	262
отображение текста.....	430, 431
перерисовка.....	476
рисование.....	476
Окно приложения.....	
кнопка разворачивания.....	138
кнопка сворачивания.....	138
панель инструментов.....	138
системное меню.....	138
создание элементов мастером.....	137
строка состояния.....	138
Окно программы, вешка разбивки.....	266
Оператор.....	
catch.....	119, 123
throw.....	119
try.....	119
перегрузка.....	90
разрешенный для перегрузки.....	95
Оператор <<.....	22, 258
Оператор >>.....	258
Оператор delete.....	50
Оператор new.....	50
проверка выделения памяти.....	50
создание массивов с помощью.....	51
Оператор присваивания, перегрузка.....	96
Операции растровые, код.....	564
Операция расширения области видимости.....	36
Определение класса.....	54
Определение функции.....	30
в C.....	30
в C++.....	30
Опция Document/View Architecture Support.....	137
Отладка программы.....	24
сочетания клавиш.....	25
Отображение.....	
режим.....	482, 589
диалогового окна.....	340, 360
текста в окне представления.....	430
Охраняемый раздел кода.....	120
Ошибки.....	
аппаратные.....	128
программного обеспечения.....	128
<b>П</b>	
Память.....	
выделение оператором new.....	50
некорректное обращение.....	128
Панель.....	
диалоговая.....	300
переключаемая.....	300
Панель инструментов.....	290
кнопки работы с буфером.....	662
промежуток между кнопками.....	292
создание.....	138, 291
удаление кнопок.....	174
Панель инструментов Standard.....	
отображение/сокрытие.....	17
Панель элементов.....	
добавление элемента ActiveX.....	749
Параметр типа.....	109
Параметры функции по умолчанию.....	38
Первичный поток.....	628
Перегрузка.....	
конструктора.....	62
оператора присваивания.....	96
операторов.....	90
функций.....	48
Переключаемая панель.....	290, 300
Переменная.....	
доступ к наследуемой.....	79
инициализация в конструкторе.....	64
область видимости.....	34, 36
объявление.....	34
ссылочная.....	39
член класса.....	55
Переопределение шаблона.....	112
Переопределение функции в Visual Studio.....	210
Перо.....	476
выбор.....	478
выбор цвета.....	479
стиль.....	479
цвет.....	479
Печать, встроенная поддержка.....	138
Печать документа.....	587
диалоговое окно Print.....	588
имитация страницы.....	587
размер.....	589
режим отображения.....	589
установки.....	587
Печать документов.....	585
в классе CEditView.....	588
организация в Application Wizard.....	585
Позиция, текущая.....	501
Поиск сочетания клавиш.....	21, 22
Полиморфизм.....	85
Поля класса.....	55
Помощь, справочная система.....	23
Порядок обхода элементов в окне.....	328
установка.....	329
Последние открытые файлы, список.....	158
Последовательность выполнения программы.....	155
Построение программы.....	24
Потоки.....	628
временная приостановка.....	630
вторичные.....	628
завершение.....	630
запуск новых.....	628
изменение приоритета.....	631
код возврата.....	629
мьютекс.....	634
ограничение доступа к ресурсам.....	634
ограничения на использование MFC.....	632
первичные.....	628
приоритет.....	629, 631

продолжение выполнения .....	630
рабочие .....	632
сигнализация .....	633
синхронизация .....	632
совместное использование ресурсов .....	628
управление .....	630
функции .....	629
функции управления .....	630
Поток-потребитель .....	633
Поток-производитель .....	633
Почтовый ящик .....	660
Предварительный просмотр документа .....	587
диалоговое окно Print Preview .....	588
имитация страницы .....	587
Представление .....	162
начало .....	484
Представление файла .....	661
Принтер .....	
объект контекста устройства .....	587
установки .....	587
Приоритет потока .....	629
Приостановка выполнения потока .....	630
Программа .....	
16-разрядная .....	18
32-х разрядная .....	18
встроенная поддержка ActiveX .....	138
встроенная поддержка печати .....	138
выходная версия .....	27
компиляция .....	24
консольная .....	18
конфигурация Debug .....	24
конфигурация Release .....	24
отладка .....	24
поддержка баз данных .....	137
построение .....	24
просмотр ошибок .....	24
размещение определений классов .....	69
с однооконным интерфейсом (SDI) .....	142
Программа ActiveXContainer .....	
генерация исходного кода .....	749
добавление элемента ActiveX .....	749
изменение ресурсов .....	749
листинг .....	750
модификация ресурсов .....	749
создание исходного кода .....	749
текст .....	750
Программа ActiveXControl .....	
включение в другие программы .....	740
генерация исходного кода .....	735
добавление свойств .....	737
значок программы .....	735
изменение кода .....	735
компиляция .....	740
листинг .....	741
методы .....	740
модификация кода .....	735
модификация страниц свойств .....	739
обработчик сообщения .....	736
отображаемые рисунки .....	735
построение элемента .....	740
ресурсы .....	735
события .....	740
создание исходного кода .....	735
сообщения .....	736
страницы свойств .....	739
текст .....	741
Программа ChessBoard .....	
генерация исходного кода .....	558
изменение кода .....	572

листинг .....	573
редактирование кода .....	572
создание графического файла .....	558
создание изображения .....	558
создание исходного кода .....	558
текст .....	573
Программа DialogView .....	382
генерация кода .....	382
изменение кода .....	384
класс диалогового окна .....	383
класс приложения .....	382
листинги .....	387
обработчики сообщений .....	384
переменные элементов управления .....	384
редактирование кода .....	384
создание .....	382
текст программы .....	387
Программа ExpContainer .....	
выполнение .....	714
генерация исходного кода .....	710
использование .....	714
класс документа .....	710
класс компонента контейнера .....	711
класс представления .....	712
класс приложения .....	710
листинг .....	715
меню .....	713
ресурсы .....	713
создание исходного кода .....	710
текст .....	715
Программа ExpServer .....	
включение возможностей OLE .....	681
выполнение .....	714
генерация исходного кода .....	681
добавление поддержки OLE .....	686
изменение кода .....	686
изменение меню .....	686
использование .....	714
класс документа .....	683
класс компонента сервера .....	684
класс окна редактирования на месте .....	684
класс представления .....	685
класс приложения .....	681
листинг .....	690
модификация кода .....	686
модификация меню .....	686
ресурсы .....	685
создание исходного кода .....	681
текст .....	690
Программа FontInfo .....	
генерация исходного кода .....	431
изменение исходного кода .....	435
изменение меню .....	435
код отображения текста .....	431
листинг .....	455
прокрутка текста .....	445
редактирование исходного кода .....	435
редактирование меню .....	435
создание исходного кода .....	431
текст программы .....	455
функция OnDraw .....	431
Программа FontView .....	
генерация исходного кода .....	323
добавление диалогового окна .....	324
изменение кода .....	335
изменение меню .....	340
изменение ресурсов .....	324
команда Text Properties .....	323
меню Text .....	323, 340
обработчики сообщений .....	336

окно Text Properties .....	323	добавление переменных .....	164
определение обработчиков сообщений .....	331	значок программы .....	173
определение переменных .....	330	изменение класса окна .....	266
отображение диалогового окна .....	340	изменение кода .....	509, 585, 589
редактирование кода .....	335	изменение меню .....	207, 226, 292, 508
редактирование меню .....	340	изменение ресурсов .....	291
редактирование ресурсов .....	324	изменение строкового ресурса .....	226
создание диалогового окна .....	324	изменение текста .....	585, 589
создание исходного кода .....	323	класс представления .....	164
текст .....	344	классы фигур .....	512
функция OnInitDialog .....	335	команда Open .....	226, 227
<b>Программа FormView</b> .....		команда Remove All .....	203, 208, 209
генерация кода .....	394	команда Save .....	226, 227
добавление обработчиков сообщений .....	395	команда Save As .....	226
добавление элементов управления .....	395	команда Undo .....	203
листинг .....	399	листинг .....	526, 596
меню .....	395	меню .....	173
обработчики сообщений .....	395	модификация кода .....	509, 585, 589
редактирование меню .....	395	модификация меню .....	292, 508, 585
редактирование окна .....	395	модификация ресурсов .....	291, 585
создание исходного кода .....	394	модификация текста .....	585, 589
текст программы .....	399	настройка меню .....	226
функция OnDraw .....	397	настройка ресурсов .....	173
функция OnInitialUpdate .....	397	панель инструментов .....	174, 290
<b>Программа FractalView</b> .....		размер рисунка .....	589
генерация исходного кода .....	486	регистрация типа файлов .....	230
изменение кода .....	486	редактирование кода .....	589
листинг .....	490	редактирование текста .....	589
модификация кода .....	486	рисование фигур .....	522
создание исходного кода .....	486	создание кода .....	163
текст .....	490	список последних использованных файлов .....	226
<b>Программа Hello</b> .....		строка состояния .....	301
исходный текст .....	19	текст .....	232, 526
создание проекта .....	17	текст программы .....	176, 272, 302, 596
<b>Программа ImpFractal</b> .....		улучшенная печать .....	589
вычисление размеров окна .....	638	файлы рисунков .....	230
генерация исходного кода .....	637	функция OnDraw .....	267
изменение исходного кода .....	637	функция OnLButtonDown .....	261
листинг .....	641	функция OnLButtonUp .....	262
модификация исходного кода .....	637	функция OnMouseMove .....	262
создание исходного кода .....	637	функция Serialize .....	228
текст .....	641	чтение/запись данных .....	228
<b>Программа MyScribe</b> .....	188	эффективная перерисовка .....	268
MDI-версия .....	412	<b>Программа TabView</b> .....	
буфер для хранения текста .....	247	вкладка Alignment .....	362
генерация исходного кода .....	412	вкладка Pitch and Spacing .....	362
генерация кода .....	187, 188	вкладка Style .....	362
значки .....	416	генерация исходного кода .....	361
изменение значков .....	416	изменение кода .....	364
изменение меню .....	246, 415	изменение меню .....	363
изменение ресурсов .....	415	определение обработчиков сообщений .....	363
изменение строкового ресурса .....	246	редактирование кода .....	364
листинг .....	416	редактирование меню .....	363
меню .....	189, 415	создание исходного кода .....	361
модификация значков .....	416	текст .....	365
модификация меню .....	246, 415	<b>Программа WinHello</b> .....	
модификация ресурсов .....	415	внесение изменений в код .....	138
мультидокументная версия .....	412	выполнение .....	141
переопределение функций .....	246	генерация исходного кода .....	136
ресурсы .....	415	класс главного окна .....	143
создание исходного кода .....	412	класс документа .....	143
создание кода .....	187	класс представления .....	143
сочетания клавиш .....	190	класс приложения .....	143
текст программы .....	191, 247, 416	классы .....	142
функция DeleteContents .....	246	команды меню .....	142
<b>Программа ScratchBook</b> .....	162	листинги .....	145
создание обработчиков сообщений команд .....	295	последовательность работы .....	155
генерация кода .....	163	состав проекта .....	142
добавление панели инструментов .....	290	текст программы .....	145
		функция InitInstance .....	157

функция OnDraw.....	140
функция WinMain.....	156
Программы	
MDI.....	411
комментарии.....	33
мультимедийный интерфейс.....	411
написание при помощи мастеров.....	135
расположение файлов.....	136
с графическим интерфейсом.....	135
с использованием API.....	135
с использованием MFC.....	135
создание новых.....	136
Продолжение выполнения потока.....	630
Проект	
выбор типа.....	18
выходная версия.....	27
выходная конфигурация.....	24
задание имени.....	18
закрытие.....	19
компиляция.....	24
конфигурации.....	141
конфигурация Debug.....	24
конфигурация Release.....	24
конфигурирование.....	24
открытие.....	19
открытие недавно открытого.....	19
отладка.....	24
отладочная конфигурация.....	24
построение.....	24
просмотр ошибок.....	24
рабочая область.....	18
рабочая папка.....	18
расположение файлов.....	136
состав.....	142
тип.....	18
файлы.....	143
Проект Hello, исходный текст.....	19
Прокрутка области отображения.....	260
Процессы.....	654
дескрипторы.....	658
дескрипторы общих объектов.....	657
дочерние.....	655
запуск.....	654
код возврата.....	656
наследование дескрипторов.....	659
ожидание завершения.....	656
получение дескрипторов объектов.....	657
родительские.....	655
<b>Р</b>	
Работа с текстом, сочетания клавиш.....	21
Рабочая область проекта.....	18
Рабочие потоки.....	632
Разбивка	
вешка.....	266
линия.....	266
Разворачивание окна, кнопка.....	138
Разделение окна.....	266
Различия между C и C++.....	29
Растровое изображение.....	557
битовые операции.....	563
загрузка.....	557
значки.....	569
код операции.....	564
отображение.....	559
рисование.....	559
создание.....	558
создание пустого.....	557

Растровые изображения	
извлечение из буфера.....	673
обмен через буфер.....	71
прием растрового в буфер обмена.....	6.
Регистрация типа файлов.....	230
Редактирование исходного кода.....	138
Редактируемое поле.....	246
Режим	
отображения.....	482
рисования.....	504
Режим отображения.....	263
при печати.....	589
Ресурсы программы, настройка.....	173
Рисование.....	485
выбор инструмента.....	478
выбор узора.....	480
дополнительные функции.....	508
закраска фигур.....	480
замкнутых фигур.....	505
инструменты.....	476
кисть.....	476
кривых Безье.....	503
лекальных кривых.....	503
линии.....	485
незакрашенных фигур.....	507
отрезков линий.....	501
перо.....	476
прозрачных (незакрашенных) фигур.....	478
регулярных кривых.....	502
режим.....	504
режим отображения.....	482
сплайна.....	503
точки.....	485
узор.....	480
фигура.....	476
фигуры.....	485
функции.....	477
Рисунок, границы в окне представления.....	262

## С

Свойства	
диалогового окна.....	324
шрифта.....	430
Сворачивание окна, кнопка.....	138
Связывание	
динамическое.....	85
статическое.....	85
Связывание объекта в OLE.....	679
Символ, высота.....	432
Символы, ввод с клавиатуры.....	447
Синхронизация потоков.....	632
объекты.....	633
Системная клавиша.....	447
Системное меню.....	138, 325
Создание	
панели инструментов.....	138
программы.....	136
строки состояния.....	138
Создание новой программы	
с классом CScrollView.....	260
Создание новых потоков.....	628
Создание обработчиков сообщений команд.....	295
Создание экземпляра (instantiating).....	110
Сообщение	
WM_CHAR.....	447, 452
WM_CREATE.....	293, 453
WM_KEYDOWN.....	447



WM_LBUTTONDOWN.....	166
WM_LBUTTONUP.....	171
WM_MOUSEMOVE.....	169
WM_PAINT.....	476
WM_SETFOCUS.....	454
WM_SIZE.....	487
WM_SYSKEYDOWN.....	447
Сообщения.....	164
карта.....	168
Состав проекта WinHello.....	142
Составной документ.....	678
Состояние, строка.....	138
Состояние мьютекса.....	635
Сохранение данных.....	203
Сочетания клавиш.....	
изменение.....	190
отладчика.....	25
поиск.....	21, 22
работа с текстом.....	21
редактирование.....	190
стандартные.....	20
Спецификатор template.....	114
Список инициализации.....	64
Список последних использованных файлов.....	226
Список последних открытых файлов.....	158
Сплайн.....	503
Справочная система.....	23
инсталляция.....	15
установка.....	15
Среда Visual Studio.NET.....	
языковые ресурсы.....	141
Ссылка.....	39
на константный объект.....	45
Статические члены класса.....	72
Статическое связывание.....	85
Стиль пера.....	479
Строка состояния.....	290, 301
создание.....	138
Строки.....	
вывод в окне представления.....	431
интервал между.....	432
отображение в окне представления.....	431
Структура CREATESTRUCT.....	176

## T

Текст.....	
выбор шрифта.....	432
извлечение из буфера обмена.....	668
отображение в окне представления.....	430, 431
передача через буфер обмена.....	663
свойства шрифта.....	430
цвет.....	433
цвет фона.....	433
Текст программы.....	145
ScratchBook.....	232
Текстовый элемент управления.....	246
Текущая позиция.....	501
Технология ActiveX.....	См. ActiveX
Точка, рисование.....	485

## У

Удаление объекта.....	56
Узор, выбор.....	480
Указатель.....	
константный на неконстанту.....	44
на базовый класс.....	83

на константный объект.....	45
на константу.....	44
Указатель this.....	71
Управление потоком.....	630
Установка Visual Studio.....	
автоматический вход в систему.....	14
аппаратура.....	14
восстановление после неудачи.....	15
выбор компонентов.....	15
выбор места для установки.....	15
лицензионное соглашение.....	15
начало.....	14
обновление системы.....	14
операционная система.....	14
регистрация.....	15
справочная система.....	15
требования.....	14
Устройство.....	
координаты.....	482
объект контекста.....	261, 432, 475, 587
режим отображения.....	263
Учет изменений в данных.....	231

## Ф

Файл.....	
Afxtempl.h.....	204
StdAfx.h.....	204
окончание для C++.....	29
памяти.....	660
регистрация типа.....	230
ресурсов.....	175
рисунка (*.dwm).....	230
список последних использованных.....	226
Файлы.....	
открытие drag-and-drop.....	415
проекта.....	143
просмотр имен.....	23
расположение.....	136
расширение.....	415
список открытых.....	158
Фактические координаты.....	260
Фигуры прозрачные (незакрашенные).....	478
Флаг изменений в данных.....	231
Фокус ввода.....	447
Формат данных для буфера обмена.....	666, 675
Формирование диалогового окна.....	383
Функции.....	
доступа к полям класса.....	58
рисования.....	477
рисования замкнутых фигур.....	505
управления потоком.....	630
установки атрибутов рисования.....	482
Функции управления окнами.....	334
Функция.....	
AddDocTemplate.....	157
AddLine.....	205, 231
AfxBeginThread.....	628
AfxRegisterWndClass.....	176
AppendMenu.....	563
Arc.....	502
Attach.....	557
BeginWaitCursor.....	172
BitBlt.....	562, 565
ClientToScreen.....	168
CloseClipboard.....	666
CloseHandle.....	636
ConnectTemplate.....	682
Create.....	267

CreateBitmap	557	HideCaret	455
CreateCompatibleBitmap	557	InflateRect	385
CreateEvent	636	InitializeCriticalSection	636
CreateEx	294	IntersectRect	265
CreateFileMapping	660	InvalidateRect	271
CreateFontIndirect	443	IsClipboardFormatAvailable	668
CreateHatchBrush	480	IsIconic	489
CreateMutex	634	LeaveCriticalSection	636
CreatePatternBrush	480	LineTo	170, 501
CreatePipe	659	LoadBitmap	557
CreateProcess	655	LoadIcon	570
CreateSemaphore	636	LoadOEMBitmap	557
CreateSolidBrush	480	LoadStandardCursor	164
CreateSolidCaret	454	LoadStdProfileSettings	158
CreateThread	628	LoadToolBar	294
создание экземпляра (instantiating)	110	LPToDP	265
DeleteContents	210	Istrcpy	665
DeleteCriticalSection	636	MapViewOfFile	661
DeleteObject	443	MoveTo	170, 501
DestroyCaret	454	MsgWaitForMultipleObjects	637
DockControlBar	294	OnBeginPrinting	591
DoModal	387	OnCreateClient	267
DoPreparePrinting	587	OnDraw	140, 203, 206, 267, 397
DPToLP	262	OnEndPrinting	591
Draw	205	OnGetEmbeddedItem	684
DrawText	434	OnIdle	486
Ellipse	506	OnInitDialog	335
EmptyClipboard	666	OnInitialUpdate	262, 397
Enable	209, 299	OnLButtonDown	261, 264
EnableShellOpen	230	OnLButtonUp	206, 262
EnableWindow	333	OnMouseMove	262, 266
EndWaitCursor	172	OnPaint	338, 386
EnterCriticalSection	636	OnPrepareDC	261, 453, 476
EnumClipboardFormats	668	OnPreparePrinting	586
ExitProcess	656	OnPrint	591
ExtTextOut	434	OnUpdate	270
FillRect	564	OpenClipboard	664
get	23	OpenMutex	658
GetAsyncKeyState	450	OpenProcess	658
GetAt	209	ParseCommandLine	157
GetClientRect	168	PatBlt	564
GetClipboardData	668	Pie	507
GetClipBox	433	PolyBezier	503
GetColor	442	PolyBezierTo	504
GetCommandLine	656	Polygon	507
GetDeviceCaps	563, 592, 595	Polyline	501
GetDimRect	268	PolylineTo	502
GetDlgItem	333	PolyPolygon	507
GetExitCodeProcess	656	PolyPolyline	502
GetExitCodeThread	631	PreCreateView	175
GetFaceName	442	ProcessShellCommand	158
GetKeyState	450	PtInRect	264
GetLength	446	PulseEvent	636
getline	22	Read	258
GetLine	205	ReadFile	660
GetMenu	563	Rectangle	506
GetNumLines	205	RegisterAll	683
GetObject	443	RegisterShellTypes	230
GetParentFrame	489	ReleaseCapture	171
GetPixel	485	ReleaseMutex	634
GetPriorityClipboardFormat	668	ReleaseSemaphore	636
GetSize	442	ResetEvent	636
GetTextMetrics	432	RoundRect	506
GetThreadPriority	631	SelectObject	432, 481
GetTotalSize	263	SelectStockObject	444, 478
GlobalAlloc	664	SendMessage	451
GlobalFree	665	Serialize	228, 229, 258
GlobalLock	664, 669	SetArcDirection	502
GlobalUnlock	664, 669	SetBkColor	433
GrayString	434	SetBkMode	433