13

Программирование для Windows с использованием MFC

Настоящая глава является началом пути к серьезной разработке Windows-приложений с использованием библиотеки Microsoft Foundation Classes (MFC). Вы получите представление о коде, который генерирует мастер создания приложений (Application Wizard) для программ MFC, и о том, какие опции можно включать в ваш код.

В этой главе вы изучите следующие вопросы.

- Базовые элементы программы, основанной на MFC.
- Отличие между приложениями однодокументного интерфейса (Single Document Interface — SDI) и многодокументного интерфейса (Multiple Document Interface — MDI).
- Как использовать мастер MFC Application Wizard для генерации программ SDI и MDI.
- Какие файлы генерирует MFC Application Wizard, и каково их содержимое.
- Как структурированы программы, генерируемые MFC Application Wizard.
- Ключевые классы программ, сгенерированных MFC Application Wizard, и как они взаимосвязаны.
- Общий подход к настройке программ, сгенерированных MFC Application Wizard.

В последующих главах вы будете постепенно расширять программы, сгенерированные в этой главе, путем инкрементного добавления кода. В конечном итоге вы получите масштабную работающую Windows-программу, которая будет включать почти все базовые приемы программирования пользовательского интерфейса, изученные на этом пути.

Концепция “документ-представление” в MFC

Когда вы пишете приложения с применением MFC, это подразумевает следование определенной структуре построения вашей программы с размещением и обработкой данных приложения определенным образом. Это может показаться ограничивающим фактором, но на самом деле большей частью это вовсе не так, и выигрыш в скорости и простоте реализации, который получается, перевешивает любые вероятные недостатки. Структура программы MFC включает две ориентированные на приложение сущности: **документ** (document) и **представление** (view), поэтому давайте рассмотрим, в чем состоит их суть и как они используются.

Что такое документ?

Документ — это имя, присвоенное коллекции данных вашего приложения, с которыми взаимодействует пользователь. Хотя слово *документ* выглядит как нечто, подразумевающее текстовую природу, на самом деле документ не ограничивается текстом. Это могут быть данные игры, геометрическая модель, текстовый файл, коллекция данных о распределении баобабов на планете маленького принца — словом, все, что хотите. Термин *документ* — это просто принятое обозначение данных приложения, рассматриваемых как единое целое.

Не должно быть сюрпризом, что документ в вашей программе определяется как объект класса документа. Ваш класс документа наследуется от класса библиотеки MFC по имени `CDocument`, и вы добавляете свои собственные данные-члены для хранения элементов, необходимых программе, а также функции-члены, поддерживающие их обработку. Ваше приложение не ограничено единственным типом документа; можно определить множество классов документов, если приложение имеет дело с несколькими разными представлениями документов.

Обработка данных приложения подобным образом позволяет использовать стандартные механизмы, предусмотренные в MFC для управления коллекцией данных приложения как единым элементом, а также сохранять и извлекать данные, содержащиеся в объектах документов на диске. Эти механизмы наследуются кодом класса вашего документа от базового класса, определенного в библиотеке MFC, так что вы автоматически получаете широкий набор функциональности, встроенной в приложение, не написав при этом ни строчки собственного кода.

Документные интерфейсы

У вас есть выбор, со сколькими документами одновременно будет иметь дело ваша программа — только с одним либо с несколькими. **Однодокументный интерфейс**, обозначаемый сокращенно как **SDI** (Single Document Interface), поддерживается библиотекой MFC для программ, которым нужно открывать по одному документу за раз. Программа, использующая этот интерфейс, называется **SDI-приложением**.

Для программ, которые должны открывать по нескольку документов одновременно, вы можете применить **многодокументный интерфейс**, сокращенно называемый **MDI** (Multiple Document Interface). В MDI, наряду с возможностью открывать множество документов одного типа, существует также возможность организовать одновременную обработку документов разного типа, причем каждый документ отображается в своем собственном окне. Разумеется, вы должны предоставить код соответствующей обработки для всех различных представлений документов, которые собираетесь поддерживать. В приложении MDI каждый документ отображается в дочернем окне приложения. У вас есть также дополнительный вариант построения интерфейса, ко-

торый называется **архитектурой множества документов верхнего уровня** (multiple top-level document architecture), когда окно каждого документа является дочерним по отношению к рабочему столу.

Что такое представление?

Представление всегда относится к определенному объекту документа. Как вы уже видели, документ в вашей программе содержит набор данных приложения, а представление — это объект, предлагающий механизм отображения некоторых или всех данных, которые хранятся в документе. Он определяет, как данные отображаются в окне, и как пользователь может взаимодействовать с ними. Подобно тому, как вы определяете документ, вы также будете определять собственный класс представления, наследуя его от класса MFC по имени `CView`. Обратите внимание, что объект представления и окно, в котором он отображается — не одно и то же. Окно, в котором появляется представление, называется **обрамляющим окном** (frame window). В действительности представление отображается в своем собственном окне, которое в точности заполняет клиентскую часть обрамляющего окна. На рис. 13.1 показан документ с двумя представлениями.

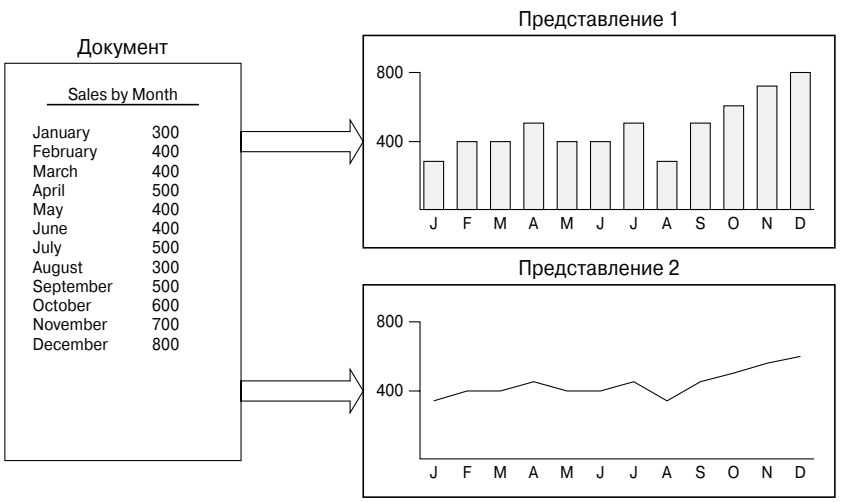

Рис. 13.1. Документ с двумя представлениями

В примере на рис. 13.1 каждое представление отображает все данные, которые содержит документ, в разной форме, хотя при необходимости представление также может отображать только часть данных документа.

Объект документа может иметь столько объектов, ассоциированных с ним, сколько вам нужно. Каждый объект представления может обеспечивать свое отличающееся представление данных документа или подмножества его данных. Например, если вы имеете дело с текстом, то разные представления могут отображать независимые блоки текста из одного и того же документа. Если программа имеет дело с графическими данными, вы можете отобразить все данные документа в отдельных окнах в разном масштабе или в разных форматах. На рис. 13.1 показан документ, содержащий числовые данные — месячные объемы продаж продукта, причем одно представление отображает гистограмму, отражающую производительность продаж, а второе — те же данные в форме графика.

Связь документа с его представлениями

Библиотека MFC включает механизм интеграции документа с его представлениями и каждого обрамляющего окна с текущим активным представлением. Объект документа автоматически поддерживает список указателей на ассоциированные с ним представления, а объект представления имеет член данных, содержащий указатель на документ, с которым он связан. Каждое обрамляющее окно сохраняет указатель на текущий активный объект представления. Координация между документом, представлением и обрамляющим окном устанавливается объектами другого класса MFC, называемыми **шаблонами документа**.

Шаблоны документа

Шаблон документа (document template) управляет объектами документов в вашей программе, а также тем, как окна и представления ассоциированы с каждым из них. Существует один шаблон документа для каждого из типов документов, которые определены в вашем приложении. Если есть два или более документов одного и того же типа, для управления ими понадобится только один шаблон документа. Чтобы точнее описать роль шаблона документа, можно сказать, что шаблон документа создает объекты документов, а представления документа создаются объектом обрамляющего окна. Объект приложения, являющийся фундаментальным для каждого приложения MFC, создает сам объект шаблона документа. На рис. 13.2 показано графическое представление этих отношений.

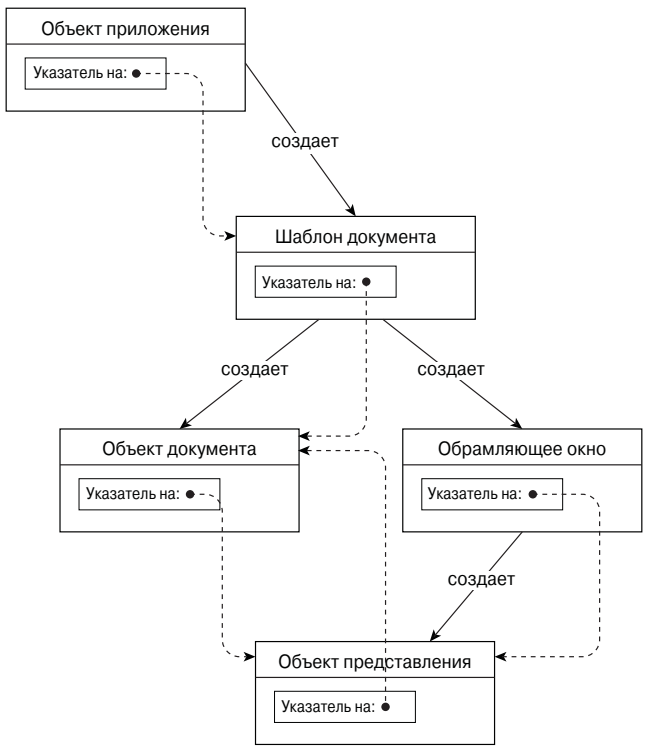

Рис. 13.2. Отношения между шаблоном документа, документом, обрамляющим окном и представлением

На этой диаграмме с помощью пунктирных линий показано использование указателей для связи объектов. Эти указатели позволяют функциям-членам объекта одного класса обращаться к общедоступным (public) данным и функциям-членам в интерфейсе другого объекта.

Классы шаблонов документа

Библиотека MFC включает два класса для определения шаблонов документа. Для SDI-приложений используется библиотечный класс CSingleDocTemplate. Он относительно прост, потому что приложение SDI имеет только один документ и обычно только одно представление. MDI-приложения несколько более сложны. Они могут иметь множество одновременно активных документов, поэтому для определения их шаблона документа требуется другой класс — CMultiDocTemplate. С большей частью этих классов вы ознакомитесь по мере продвижения в разработке кода приложения.

Ваше приложение и MFC

На рис. 13.3 показаны четыре базовых класса, которые появляются почти в каждом Windows-приложении на основе MFC.

- Класс приложения CMyAPp.
- Класс обрамляющего окна CMyWnd.
- Класс представления CMyView, который определяет, как должны отображаться данные, содержащиеся в CMyDoc, в клиентской области окна, которое создано объектом CMyWnd.
- Класс документа CMyDoc, который определяет документ, содержащий данные приложения.

Действительные имена этих классов специфичны для каждого конкретного приложения, однако наследование от MFC в основном одно и то же, хотя могут быть и альтернативные базовые классы, в частности, класс представления. Как вы увидите чуть позже, в MFC предусмотрены разные вариации класса представления, включающие широкий набор предварительно разработанной функциональности, что позволяет сэкономить немало усилий по кодированию. Обычно вам не придется расширять класс, определяющий шаблон документа для вашего приложения, так что стандартного MFC-класса CSingleDocTemplate для SDI-программы обычно достаточно. Когда вы создаете программу MDI, вашим классом шаблона документа является CMultiDocTemplate, который также унаследован от CDocTemplate.

Стрелки на диаграмме направлены от базового класса к производному. Показанные здесь библиотечные классы MFC формируют довольно сложную структуру наследования, но фактически представляют лишь малую часть полной структуры MFC. Обычно знать все детали иерархии MFC не обязательно, но важно иметь общее представление об унаследованных членах ваших классов. Вы не увидите никаких определений базовых классов в своей программе, но унаследованные члены производных классов программы происходят от непосредственного базового класса, а также от всех непрямых базовых классов из иерархии MFC. Чтобы понять, какие члены присутствуют в классах программы, вы должны знать, от каких классов они унаследованы. После того, как вы это узнаете, можете поискать их описание в справочной системе.

Еще одно, о чем не нужно беспокоиться — помнить о том, какие классы должны присутствовать в вашей программе, и какие базовые классы использовать в их определении. Как будет показано ниже, обо всем этом заботится Visual C++ 2008.

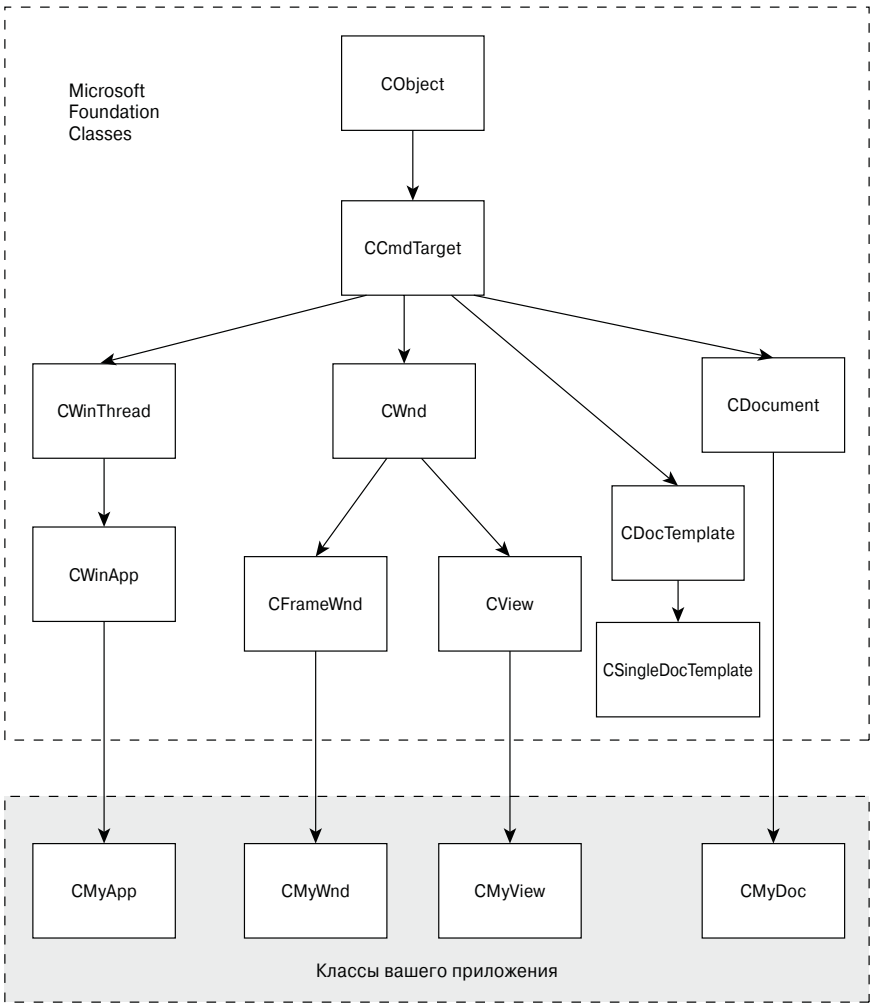

Рис. 13.3. Базовые классы, присутствующие почти в каждом Windows-приложении на основе MFC

Создание приложений MFC

При разработке Windows-программ на базе MFC используются четыре основных инструмента.

1. **Мастер создания приложений** (Application Wizard) для начального создания базового кода прикладной программы. Вы пользуетесь этим мастером при каждом создании проекта, в результате чего код генерируется автоматически.
2. Контекстное меню проекта в Class View (Представление классов) для добавления новых классов и ресурсов к вашему проекту. Контекстное меню отображается при щелчке правой кнопкой мыши в Class View, а новый класс создается выбором пункта Add⇨Class (Добавить⇨Класс) в этом меню. Ресурсы — это нечто, включающее неисполняемые данные, вроде графических изображений, пикто-

грамм, меню и диалоговых окон. Пункт Add⇨Resource (Добавить⇨Ресурс) того же контекстного меню позволяет добавить новый ресурс.

3. Контекстное меню класса используется в Class View для расширения и настройки существующих классов программы. Для этого применяются пункты Add⇨Add Function (Добавить⇨Добавить функцию) и Add⇨Add Variable (Добавить⇨Добавить переменную) этого меню.

4. **Редактор ресурсов** применяется для создания или модификации таких объектов, как меню и панели инструментов.

На самом деле доступно несколько редакторов ресурсов; каждый применяется в определенной конкретной ситуации, в зависимости от вида ресурса, который вы редактируете. Редактирование ресурсов рассматривается в следующей главе, но пока давайте забежим немного вперед и создадим приложение MFC.

Процесс создания приложения MFC почти так же прост, как создание консольной программы; на этом пути будет лишь немногим больше вариантов выбора. Как уже было показано, все начинается с создания нового проекта выбором пункта меню File⇨New⇨Project (Файл⇨Создать⇨Проект) или же нажатием комбинации клавиш <Ctrl+Shift+N>. После этого отображается диалоговое окно New Project (Новый проект), в котором можно выбрать MFC в качестве типа проекта и MFC Application (Приложение MFC) — в качестве используемого шаблона приложения. Также потребуется ввести имя проекта, которое может быть любым. Я использовал `TextEditor`, как показано на рис. 13.4. Вам не придется превращать этот конкретный пример в какое-то серьезное приложение, так что можете выбрать любое имя на свой вкус.

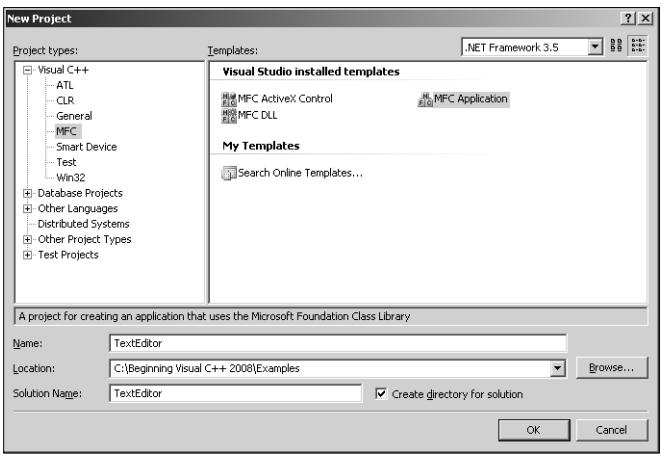

Рис. 13.4. Создание проекта `TextEditor`

Как вы знаете, имя, присвоенное проекту — `TextEditor` в данном случае — служит именем папки, содержащей все файлы проекта, однако оно также применяется и для создания имен классов, которые Application Wizard генерирует для вашего проекта. Когда вы щелкаете на кнопке OK в диалоговом окне New Project, то видите перед собой диалоговое окно MFC Application Wizard (Мастер создания приложений MFC), в котором можете указать опции своего приложения (рис. 13.5).

Как видите, диалоговое окно объясняет все активные установки проекта, и в правой его части доступен набор опций, которые можно выбрать. Вы можете выбрать любую из них, дабы посмотреть, что получится — у вас всегда есть возможность вер-

нуться к базовому диалоговому окну Application Wizard, щелкнув на кнопке Previous (Назад). Выбор любой из этих опций справа предоставит полный набор дальнейших уточняющих настроек, так что в сумме их очень много. Я не стану обсуждать их все, а только вкратце опишу те из них, которые, скорее всего, представят наибольший интерес, а остальные оставлю для вашего самостоятельного исследования. Изначально Application Wizard позволяет выбрать SDI-приложение, MDI-приложение либо приложение на основе диалогового окна. Давайте для начала создадим приложение SDI и рассмотрим все опции, которые потребуется выбрать в процессе.

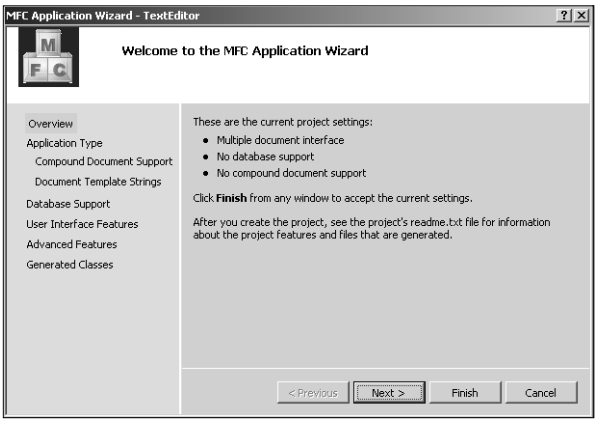

Рис. 13.5. Диалоговое окно MFC Application Wizard

Создание SDI-приложения

Выберите опцию Application Type (Тип приложения) из списка в правой части диалогового окна.

По умолчанию будет выбран переключатель Multiple documents (Множество документов), что означает многодокументный интерфейс (MDI), и внешний вид MDI-приложения представлен в левой верхней части диалогового окна, чтобы вы знали, чего ожидать. Выберите переключатель Single document (Один документ), и представление приложения слева вверху изменится, как показано на рис. 13.6.

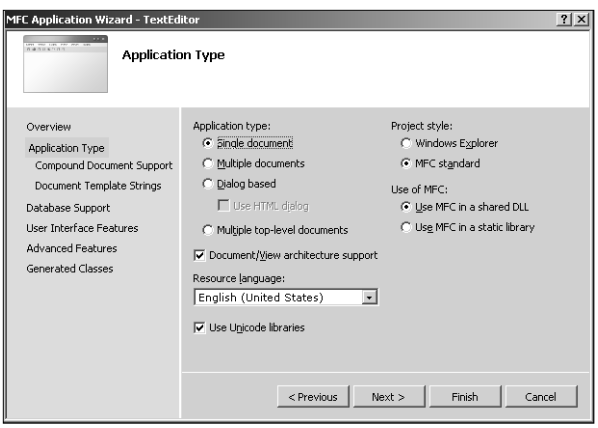

Рис. 13.6. Выбор в качестве типа приложения Single document

Теперь рассмотрим прочие опции, которые вы видите здесь для данного типа приложений; они перечислены в табл. 13.1.

Таблица 13.1. Опции SDI-приложений

Опция	Описание
Dialog based (Основанное на диалоге)	Окном приложения является диалоговое окно, а не обрамляющее окно
Multiple top-level documents (Множество документов верхнего уровня)	Документы отображаются в дочерних окнах рабочего стола, а не в дочерних окнах приложения, как это происходит в MDI-приложении
Document/View architecture support (Поддержка архитектуры "документ-представление")	Эта опция выбрана по умолчанию, так что вы получаете встроенный код для поддержки архитектуры "документ-представление". Если вы отключите эту опцию, то поддержка не предоставляется, и вы должны реализовать все необходимое самостоятельно
Resource language (Язык ресурсов)	Этот раскрывающийся список отображает выбор языков, применяемых в ресурсах вашего приложения, таких как меню и текстовые строки
Use Unicode libraries (Использовать библиотеки Unicode)	Поддержка Unicode обеспечивается Unicode-версиями библиотек MFC; если вы хотите использовать их, то должны пометить этот флажок

Вы должны снять отметку с флажка Use Unicode libraries (Использовать библиотеки Unicode), который по умолчанию отмечен. Если оставить его отмеченным, то приложение будет ожидать ввода в кодировке Unicode, и в файлах будут сохраняться символы Unicode. Это сделает их нечитаемыми для программ, ожидающих текста ASCII.

Вы также можете выбирать между Windows Explorer (Проводник Windows) и MFC standard (Стандартное MFC) для стиля проекта. Первый вариант реализует окно приложения с клиентской областью, разделенной на две панели: в левой отображаются данные в форме дерева, а в правой — стандартный текст.

Вы также можете выбрать вариант использования кода MFC для вашей программы. По умолчанию применяется библиотека MFC как разделяемая DLL (динамически подключаемая библиотека) — это означает, что ваша программа компонуется с процедурами библиотеки MFC во время выполнения. Это позволяет уменьшить размер исполняемого файла, который вы генерируете, но требует наличия MFC DLL на машине, где он должен запускаться. Эти два модуля вместе (исполняемый модуль `.exe` вашей программы и MFC `.dll`) могут оказаться больше, чем когда вы статически скомпонуете библиотеку MFC в исполняемый файл. Если выбрана статическая компоновка, процедуры библиотеки MFC включаются в исполняемый модуль вашей программы при ее сборке. Статически скомпонованные приложения работают немного быстрее, чем те, что компонуются с библиотекой MFC динамически, так что приходится выбирать компромисс между использованием памяти и скоростью выполнения. Если вы оставите включенной опцию по умолчанию, предусматривающую использование MFC как разделяемой DLL-библиотеки, то несколько программ, выполняющихся одновременно, могут разделять единственную копию библиотеки в памяти.

В диалоговом окне Document Template Strings (Строки шаблона документа) вы можете ввести расширение файлов, которые создает ваша программа. Для этого примера вполне подойдет расширение `.txt`. Вы можете также ввести в этом диалоговом окне Filter Name (Имя фильтра), что будет именем фильтра, который будет появляться в диалоговых окнах Open (Открытие) и Save As (Сохранение) для фильтрации файлов, так что будут отображаться только файлы с указанным расширением.

Если выбрать элемент User Interface Features (Возможность интерфейса пользователя) из списка в правой панели окна MFC Application Wizard, вы получите дальнейший набор опций, которые могут быть включены в ваше приложение (табл. 13.2).

Таблица 13.2. Дополнительные опции SDI-приложений

Опция	Описание
Thick Frame (Толстая рамка)	Позволяет изменять размер окна приложения, перетаскивая его границу. Выбрано по умолчанию
Minimize box (Кнопка сворачивания)	Эта опция также выбрана по умолчанию и предоставляет кнопку сворачивания в правом верхнем углу окна приложения
Maximize box (Кнопка разворачивания)	Эта опция также выбрана по умолчанию и предоставляет кнопку разворачивания в правом верхнем углу окна приложения
Minimized (Свернутое)	Если выбрана эта опция, приложение стартует с окном, свернутым в пиктограмму
Maximized (Развернутое)	Если выбрана эта опция, окно приложения при запуске будет развернуто на весь экран
Initial status bar (Начальная панель состояния)	Эта опция добавляет панель состояния в нижнюю часть окна приложения, включающую индикаторы клавиш <CAPS LOCK>, <NUM LOCK> и <SCROLL LOCK>, а также строку сообщений, отображающую вспомогательные сообщения для меню и кнопок панели инструментов. Эта опция также добавляет команды меню для сокрытия и отображения панели состояния
Split window (Разделить окно)	Эта опция предоставляет разделительную черту для каждого из основных представлений приложения
Standard docking toolbar (Стандартная стыкуемая панель инструментов)	Эта опция добавляет в окно приложения панель инструментов со стандартным набором кнопок, являющихся альтернативой стандартным пунктам меню. Панель инструментов предоставляется по умолчанию. Стыкуемая (dockable) панель инструментов может быть прикреплена к любой грани окна приложения, так что вы можете поместить ее там, где вам удобно. Добавление кнопок к панели инструментов будет описано в главе 13
Browser style toolbar (Панель инструментов в стиле браузера)	Это добавляет в окно приложения панель инструментов в стиле браузера Internet Explorer

Имеется еще пара средств в наборе опций Advanced Features (Дополнительные возможности), о которых вам следует знать. Одно из них — Printing and print preview (Печать и предварительный просмотр), которое выбрано по умолчанию, а другое — Context-sensitive help (Контекстно-чувствительная справка), которое вы получите, если отметите соответствующий флажок. Отметка флажка Printing and print preview добавляет стандартные пункты Page Setup (Параметры страницы), Print Preview (Предварительный просмотр) и Print (Печать) в меню File (Файл), а мастер Application Wizard предоставит код для их поддержки. Включение опции Context-sensitive help дает базовый набор возможностей поддержки контекстной подсказки. Понятно, что вы должны будете добавить специфическое содержимое в справочные файлы, если хотите использовать это средство.

Если в диалоговом окне MFC Application Wizard выбрать опцию Generated Classes (Сгенерированные классы), вы увидите список классов, которые Application Wizard генерирует в коде вашей программы, как показано на рис. 13.7.

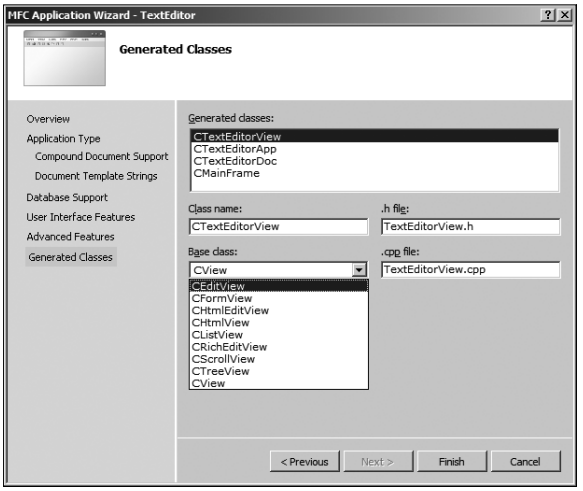

Рис. 13.7. Просмотр сгенерированных классов

Можно выделить любой класс в списке, щелкнув на его имени, и в находящихся ниже полях будет отображено имя класса, имя заголовочного файла, в котором находится его определение, имя базового класса, а также имя файла, содержащего реализацию функций-членов класса. Определение класса всегда содержится в файле `.h`, а исходный код функций-членов – в файле `.cpp`.

В случае класса `CTextEditorDoc` вы можете изменить все за исключением базового класса; однако если выбрать `CtextEditorApp`, то единственное, что можно будет изменить – это имя класса. Попробуйте щелкнуть на именах других классов в списке. Для `CMainFrame` вы сможете изменить все за исключением базового класса, а для класса `CTextEditorView` , показанного на рис. 13.7, – также и базовый класс. Щелкните на кнопке со стрелкой вниз для отображения списка классов, из которого можно выбрать базовый класс; этот список показан на рис. 13.7. Возможности, встраиваемые в ваш класс представления, зависят от выбранного базового класса (табл. 13.3).

Таблица 13.3. Базовые классы и возможности

Базовый класс	Возможности класса представления
`CEditView`	Предоставляет простую возможность редактирования многострочных текстов, включая поиск с заменой, а также печать
`CFormView`	Обеспечивает представление формы; форма — это диалоговое окно, содержащее элементы управления, управляющие отображением данных и пользовательским вводом. По сути это та же функциональность, что предлагается в приложении Windows Forms для CLR, о чем пойдет речь в главе 22
`CHtmlEditView`	Этот класс расширяет класс `CHtmlView` и добавляет возможность редактирования HTML-страниц
`CHtmlView`	Обеспечивает представление, в котором могут отображаться Web-страницы и локальные HTML-документы
`CListView`	Позволяет использовать архитектуру “документ-представление” со списочными элементами управления
`CRichEditView`	Предоставляет возможность отображения и редактирования документов, содержащих форматируемый текст

Окончание табл. 13.3

Базовый класс	Возможности класса представления
`CScrollView`	Обеспечивает представление, автоматически добавляющее линейки прокрутки, когда отображаемые данные того требуют
`CTreeView`	Предоставляет возможность использовать архитектуру "документ-представление" с древовидными элементами управления
`CView`	Предоставляет базовые возможности просмотра документов

Поскольку мы назвали приложение `TextEditor`, что говорит о его способности редактировать текст, выберем в качестве базового класса `CEditView`, чтобы автоматически получить базовые возможности редактирования.

Щелкните на кнопке Finish (Готово), чтобы получить исходные файлы полностью рабочей базовой программы, сгенерированные мастером MFC Application Wizard на основе выбранных вами опций.

Вывод мастера MFC Application Wizard

Все программные файлы, сгенерированные мастером создания приложений, сохраняются в папке проекта `TextEditor`, вложенной в папку решения с тем же именем. Кроме того, во вложенной в папку решения папке `res` будут находиться файлы ресурсов. IDE-среда поддерживает несколько способов просмотра информации, имеющих отношение к вашему проекту, как показано в табл. 13.4.

Таблица 13.4. Способы просмотра информации, относящейся к проекту

Вкладка/панель	Содержимое
Solution Explorer (Проводник решения)	Показывает файлы, включенные в ваш проект. Файлы размещены по категориям в виртуальных папках под названиями Header Files (Заголовочные файлы), Resource Files (Ресурсные файлы) и Source Files (Исходные файлы)
Class View (Представление классов)	Отображает классы, присутствующие в вашем проекте, вместе с их членами. Также выводятся глобальные сущности, определенные вами. Классы показаны в верхней панели, а в нижней — члены класса, выбранного в данный момент в верхней панели. Щелкая правой кнопкой мыши на элементах, отображаемых в Class View, можно получить меню, используемое для просмотра определения выбранной сущности
Resource View (Представление ресурсов)	Здесь отображаются ресурсы вроде пунктов меню и кнопок панели инструментов, использованных в вашем проекте. Щелчок правой кнопкой мыши на ресурсе отображает меню, позволяющее редактировать текущий ресурс или добавлять новые
Property Pages (Страницы свойств)	Здесь отображаются версии, которые вы можете собрать из вашего проекта. Отладочная версия включает дополнительные возможности для облегчения отладки кода. Рабочая версия дает в результате более компактный исполняемый файл и собирается тогда, когда код полностью протестирован и отлажен для производственного применения. Щелкнув правой кнопкой мыши на версии — Debug (Отладочная) или Release (Рабочая) — вы можете вызвать контекстное меню, в котором добавить панель свойств или отобразить свойства, установленные для версии в данный момент. Панель свойств позволяет установить настройки для компилятора и компоновщика

Если вы щелкнете правой кнопкой мыши на TextEditor в панели Solution Explorer и выберете в контекстном меню пункт Properties (Свойства), будет отображено окно свойств проекта, показанное на рис. 13.8.

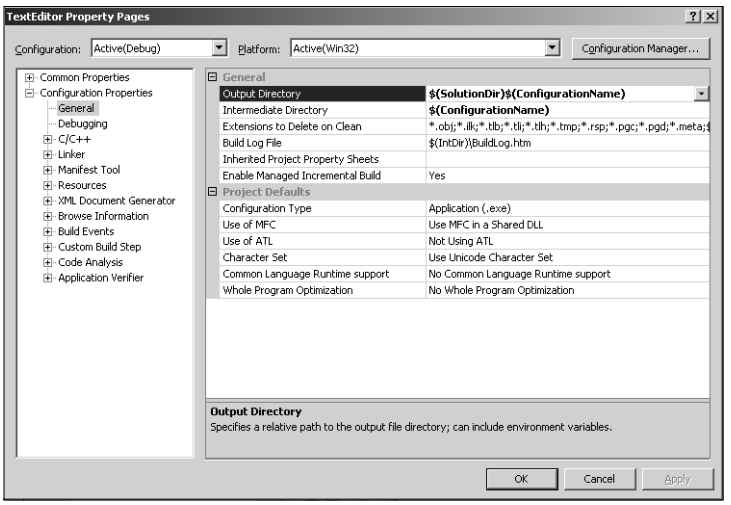

Рис. 13.8. Окно свойств проекта

В левой панели отображаются группы свойств, которые можно выбрать для показа в правой панели. В данный момент отображается группа свойств General (Общие), и вы можете менять значение свойства в правой панели щелчком на нем и выбором нового значения из раскрывающегося окна списка справа от имени свойства, либо в некоторых случаях в результате ввода нового значения с клавиатуры.

Просмотр файлов проекта

Если вы выберете вкладку Solution Explorer и развернете список щелчком на + рядом с TextEditor, а затем щелкнете на + рядом с каждой из папок Source Files, Header Files и Resource Files, то увидите полный список всех файлов проекта, как показано на рис. 13.9.

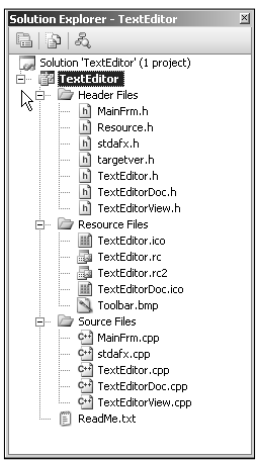

Рис. 13.9. Полный список всех файлов проекта

На рис. 13.9 показана панель в виде плавающего окна, чтобы сделать список файлов полностью видимым; вы можете сделать любую из панелей плавающей, щелкнув на стрелке вниз на вершине панели и выбрав из списка возможных позиций. Как видите, всего в проекте присутствует 17 файлов (без файла ReadMe.txt). Просмотреть содержимое любого файла можно, выполнив двойной щелчок на его имени. Содержимое выбранного файла отображается в окне редактора. Попробуйте это с файлом ReadMe.txt. Вы увидите, что он содержит краткое описание содержимого каждого из файлов, составляющих проект. Я не стану здесь повторять описание этих файлов, поскольку они предельно ясно описаны в ReadMe.txt.

Просмотр классов

Доступ к проекту, предоставляемый во вкладке Class View, часто более удобен, чем в Solution Explorer, потому что классы являются основой всей организации приложения. Когда вы хотите взглянуть на код, то обычно, прежде всего, вас интересуют определение класса или реализация функций-членов класса, и из Class View можно получить доступ к тому и другому. Иногда, однако, необходим и Solution Explorer. Если вы хотите проверить директивы `#include` в файле `.cpp`, то с помощью Solution Explorer сумеете открыть непосредственно интересующий файл.

В панели Class View можно развернуть элемент `TextEditor`, чтобы отобразить классы, определенные в приложении. Щелчок на имени любого класса приведет к отображению членов этого класса в нижней панели. В панели Class View, показанной на рис. 13.10, выбран класс `CTextEditorDoc`.

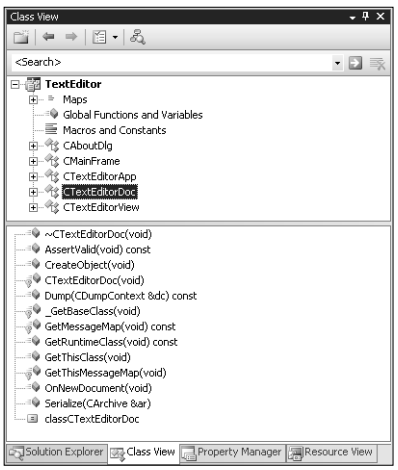

Рис. 13.10. Просмотр членов класса `CTextEditorDoc` в панели Class View

На рис. 13.10 показана панель Class View в пристыкованном состоянии. Пиктограммы символизируют различные сущности, которые можно отобразить, а описание их значения находится в документации по Class View.

Легко заметить, что имеется четыре класса, о которых говорилось выше, и которые представляют основу приложения MFC: `CTextEditorApp` — для приложения, `CMainFrame` — для обрамляющего окна приложения, `CTextEditorDoc` — для документа и `CTextEditorView` — для представления. Также есть класс `CAboutDlg`, определяющий объекты поддержки диалогового окна, которое появляется при выборе пункта меню Help⇨About (Справка⇨О программе) в приложении. Если выбрать Global Functions and Variables (Глобальные функции и переменные), вы увидите, что оно содержит два определения: объект приложения `theApp` и `indicators`, представляющий собой массив индикаторов, записывающих состояния клавиш <CAPS LOCK>, <NUM LOCK> и <SCROLL LOCK>, отображаемые в панели состояния.

Чтобы просмотреть код определения класса в панели редактора, вы просто выполняете двойной щелчок на имени соответствующего класса в дереве Class View. Аналогично, чтобы просмотреть код функции-члена, дважды щелкните на имени функции. Обратите внимание, что можно перетаскивать грани любой из панелей в окно IDE, чтобы увидеть содержимое кода. Можно скрыть или показать набор панелей Solution

Explorer, щелкнув на кнопке Autohide (Автоматически скрывать) в правой части панели заголовка этой панели.

Определения классов

Я не стану здесь погружаться в детали реализации классов — вы просто получите представление о том, как они выглядят, а я подчеркну несколько важных аспектов. Если выполнить двойной щелчок на имени класса в Class View, отображается код определения класса. Для начала взглянем на класс приложения — `CTextEditorApp`. Определение этого класса приведено ниже.

```
// TextEditor.h : главный заголовочный файл для приложения TextEditor
//
#pragma once
#ifndef __AFXWIN_H__
   #error "включите 'stdafx.h' перед включением этого файла для PCH"
#endif
#include "resource.h" // главные символы
// CTextEditorApp:
// В TextEditor.cpp находится реализация этого класса
//
class CTextEditorApp : public CWinApp
{
public:
   CTextEditorApp();
// Переопределения
public:
   virtual BOOL InitInstance();
// Реализация
  afx_msg void OnAppAbout();
  DECLARE_MESSAGE_MAP()
};
extern CTextEditorApp theApp;
```

Класс `CTextEditorApp` унаследован от `CWinApp` и включает конструктор, виртуальную функцию `InitInstance()`, функцию `OnAppAbout()` и макрос `DECLARE_MESSAGE_MAP()`.

Макрос — это не код C++. Это имя, определенное директивой препроцессора `#define`, заменяемое некоторым текстом, который обычно является кодом C++, но также может быть и константами или символами определенного рода.

Макрос `DECLARE MESSAGE_MAP()` касается определения того, какие сообщения Windows какими функциями-членами класса обрабатываются. Макрос присутствует в определении любого класса, который может обрабатывать сообщения Windows. Конечно, наш класс приложения наследует множество функций и данных-членов от базового класса, и вы ознакомитесь с ними по мере расширения примеров программ. Если посмотреть в начало кода определения класса, можно заметить директиву `#pragma once`, предотвращающую многократное включение данного файла. Далее идет группа директив препроцессора, гарантирующих включение файла `stdafx.h` перед данным файлом. Обрамляющее окно приложения нашей SDI-программы создается объектом класса `CMainFrame`, который определен следующим образом:

```
class CMainFrame : public CFrameWnd
{
  protected: // создавать только из 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 управляющий панелей
  CStatusBar m_wndStatusBar;
  CToolBar m_wndToolBar;
// Сгенерированные функции отображения сообщений
protected:
  afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
  DECLARE_MESSAGE_MAP()
};
```

Этот класс унаследован от `CFrameWnd`, который предоставляет большую часть функциональности, необходимой обрамляющему окну нашего приложения. Производный класс включает защищенные (`protected`) данные-члены — `m_wndStatusBar` и `m_wndToolBar`, которые являются экземплярами MFC-классов `CStatusBar` и `CToolBar` соответственно. Эти объекты создают и управляют панелью состояния, появляющейся в нижней части окна приложения, и панелью инструментов, предлагающей кнопки для быстрого доступа к стандартным функциям меню.

Определение класса `CTextEditorDoc`, созданного мастером MFC Application Wizard, выглядит так:

```
class CTextEditorDoc : public CDocument
{
protected: // создавать только из сериализации
  CTextEditorDoc();
  DECLARE_DYNCREATE(CTextEditorDoc)
// Атрибуты
public:
// Операции
public:
// Переопределения
public:
  virtual BOOL OnNewDocument();
  virtual void Serialize(CArchive& ar);
// Реализация
public:
  virtual ~CTextEditorDoc();
#ifdef _DEBUG
  virtual void AssertValid() const;
  virtual void Dump(CDumpContext& dc) const;
#endif
protected:
// Сгенерированные функции отображения сообщений
protected:
  DECLARE_MESSAGE_MAP()
};
```

Как и в случае предыдущего класса, большая часть "мяса" поступает от базового класса, и потому здесь не видна. Макрос DECLARE_DYNCREATE(), который следует за конструктором (он также использовался в классе CMainFrame), позволяет создавать объект данного класса динамически, синтезируя его из данных, прочитанных из файла. Когда вы сохраняете объект SDI-документа, обрамляющее окно, которое содержит представление, сохраняется наряду с вашими данными. Это позволяет потом все восстановить, прочтя обратно. Чтение и запись объекта документа в файл поддерживается процессом, называемым **сериализацией**. В последующих примерах, которые мы будет разрабатывать, вы увидите, как с помощью сериализации можно записывать свои собственные документы в файл и затем реконструировать их по данным этого файла.

Класс документа также содержит в своем определении макрос DECLARE_MESSAGE_MAP(), чтобы при необходимости позволить обрабатывать сообщения Windows функциями-членами класса.

Класс представления в нашем SDI-приложении определен так, как показано ниже:

```
class CTextEditorView : public CEditView
{
protected: // создавать только из сериализации
  CTextEditorView();
  DECLARE_DYNCREATE(CTextEditorView)
// Атрибуты
public:
   CTextEditorDoc* 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 ~CTextEditorView();
#ifdef _DEBUG
  virtual void AssertValid() const;
  virtual void Dump(CDumpContext& dc) const;
#endif
protected:
// Сгенерированные функции отображения сообщений
protected:
   DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG // отладочная версия в TextEditorView.cpp
  inline CTextEditorDoc* CTextEditorView::GetDocument() const
   { return reinterpret_cast<CTextEditorDoc*>(m_pDocument); }
#endif
```

Как мы специфицировали в диалоговом окне MFC Application Wizard, класс представления унаследован от класса CEditView, который уже включает базовые средства обработки текста. Функция GetDocument() возвращает указатель на объект документа, соответствующий представлению, и вы будете использовать его для доступа к данным в объекте документа, когда добавите свои собственные расширения к классу представления.

Создание исполняемого модуля

Чтобы откомпилировать и скомпоновать программу, выбурите пункт меню Build⇨Build Solution (Сборка⇨Собрать решение), нажмите <Ctrl+Shift+R> или щелкните на кнопке Build (Сборка) в панели инструментов.

В коде, сгенерированном мастером создания приложений, присутствуют две реализации функции `GetDocument()` — члена класса `CTextEditorView`. Одна из них находится в файле `.cpp` класса `CEditView` и используется в отладочной версии программы. Обычно именно ее вы и будете использовать в процессе разработки программы, поскольку она обеспечивает верификацию значения указателя на документ. (Документ хранится в унаследованном члене данных `m_pDocument` класса представления.) Версию, которая применяется в рабочей версии вашей программы, можно найти после определения класса в файле `TextEditorView.h`. Эта версия объявлена как `inline` и не проверяет указатель на документ. Функция `GetDocument()` просто предоставляет связь с объектом документа. Вы можете вызывать любую из этих функций в интерфейсе класса документа, используя указатель на документ, возвращенный функцией.

По умолчанию отладочные возможности включены в вашу программу. Наряду со специальной версией `GetDocument()`, в этом случае в код MFC включается множество разнообразных проверок. Если вы хотите изменить это, можете воспользоваться раскрывающимся списком в панели инструментов Build для выбора конфигурации версии, которая не содержит отладочного кода.

При компиляции вашей программы с включенной отладкой компилятор не обнаруживает неинициализированных переменных, так что может оказаться полезным иногда выполнять сборку рабочей версии даже на этапе тестирования программы.

Предварительно скомпилированные заголовочные файлы

Первый раз, когда вы компилируете и компонуете программу, это занимает некоторое время. Второй и все последующие разы это должно происходить немного быстрее благодаря средству Visual C++ 2008, называемому **предварительно скомпилированными заголовочными файлами** (precompiled headers). Во время начальной компиляции компилятор сохраняет вывод от компиляции заголовочных файлов в специальном файле с расширением `.pch`. При последующих сборках этот файл используется повторно, если источники в заголовках не изменились, что позволяет сэкономить время на компиляцию заголовков.

Вы можете определить, используются ли предварительно скомпилированные заголовки, и управлять тем, как они обрабатываются, через вкладку Properties (Свойства). Щелкните правой кнопкой мыши на `TextEditor` и выберите в контекстном меню пункт Properties. Если вы развернете узел C/C++ в отображенном диалоговом окне, то сможете выбрать Precompiled Headers (Предварительно скомпилированные заголовочные файлы), чтобы установить это свойство.

Запуск программы

Чтобы выполнить программу, нажмите <Ctrl+F5>. Поскольку в качестве базового для `CTextEditorView` выбран класс `CEditView`, программа представляет полностью функционирующий простой текстовый редактор. Вы можете вводить текст в окне, как показано на рис. 13.11.

Обратите внимание, что приложение имеет линейки прокрутки для просмотра текста за пределами видимой области внутри окна, и, конечно, вы можете изменять размеры окна, перетаскивая его границы. Все пункты меню полностью функционируют, так что можете сохранять и загружать файлы, вырезать и вставлять текст, а также

печатать текст в окне — и все это без написания ни единой строчки кода! При перемещении курсора над кнопками панели инструментов или над пунктами меню в панели состояния появляются сообщения, описывающие функции этих кнопок или пунктов меню, а если задержать курсор над кнопкой на секунду, всплывает подсказка, отображающая ее назначение. (Вы узнаете подробнее об этих всплывающих подсказках в главе 14.)

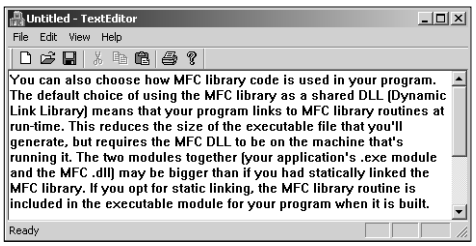

Рис. 13.11. Работа программы `TextEditor`

Как работает программа

Как и в тривиальном примере MFC-приложения, который рассматривался ранее в этой главе, объект приложения создается в глобальном контексте нашей SDI-программы. Вы можете видеть это, развернув элемент Global Functions and Variables (Глобальные функции и переменные) в Class View и затем дважды щелкнув на `theApp`. В окне редактора вы увидите следующий оператор:

```
CTextEditorApp theApp;
```

Это объявление объекта `theApp` как экземпляра нашего класса приложения `CTextEditorApp`. Приведенный оператор находится в файле `TextEditor.cpp`, который также содержит объявления функций-членов класса приложения и определение класса `CAboutDlg`.

После того, как создан объект `theApp`, вызывается функция `WinMain()`, предоставленная MFC. Она, в свою очередь, вызывает две функции-члена объекта `theApp`. Сначала вызывается `InitInstance()`, предназначенная для выполнения всей необходимой инициализации приложения, а затем `Run()`, выполняющая начальную обработку сообщений Windows. Функция `WinMain()` не присутствует явно в исходном коде проекта, поскольку она предоставляется библиотекой классов MFC и вызывается автоматически при запуске приложения.

Функция `InitInstance()`

Вы можете получить доступ к коду функции `InitInstance()` двойным щелчком на ее имени в Class View после выделения класса `CTextEditorApp` или, если вы спешите, можете просто взглянуть на код, следующий немедленно за определением объекта `theApp`. Версия, созданная мастером MFC Application Wizard, выглядит так:

```
BOOL CTextEditorApp::InitInstance()
{
  // InitCommonControlsEx() требуется под Windows XP, если манифест приложения
  // указывает на использование ComCtl32.dll версии 6 и последующих для
  // разрешения визуальных стилей. В противном случае создание окна
  // завершится неудачей.
  INITCOMMONCONTROLSEX InitCtrls;
  InitCtrls.dwSize = sizeof(InitCtrls);
```

```
    // Установите это для включения всех классов общих элементов
    // управления, которые хотите использовать в своем приложении.
    InitCtrls.dwICC = ICC_WIN95_CLASSES;
    InitCommonControlsEx(&InitCtrls);

    CWinApp::InitInstance();

    // Инициализировать библиотеки OLE
    if (!AfxOleInit())
    {
      AfxMessageBox(IDP_OLE_INIT_FAILED);
      return FALSE;
    }
    AfxEnableControlContainer();
    // Стандартная инициализация.
    // Если вы не используете эти средства и желаете уменьшить размер финального
    // исполняемого файла, можете удалить специфические процедуры инициализации,
    // в которых не нуждаетесь.
    // Измените ключ реестра, под которым будут сохраняться настройки.
    // TODO: вы должны модифицировать эту строку, указав конкретное название
    // компании или организации
    SetRegistryKey(_T("Local AppWizard-Generated Applications"));
    LoadStdProfileSettings(4); //Загрузить опции из стандартного INI-файла (включая MRU)
    // Зарегистрировать шаблоны документов приложения. Шаблоны документов служат
    // в качестве соединений между документами, обрамляющими окнами и представлениями
    CSingleDocTemplate* pDocTemplate;
    pDocTemplate = new CSingleDocTemplate(
      IDR_MAINFRAME,
      RUNTIME_CLASS(CTextEditorDoc),
      RUNTIME_CLASS(CMainFrame), // главное обрамляющее окно SDI
      RUNTIME_CLASS(CTextEditorView));
    if (!pDocTemplate)
      return FALSE;
    AddDocTemplate(pDocTemplate);

    // Разобрать командную строку для стандартных команд оболочки, DDE, открытия файла
    CCommandLineInfo cmdInfo;
    ParseCommandLine(cmdInfo);

    // Выполнить диспетчеризацию команд, заданных в командной строке.
    // Вернуть FALSE, если приложение было запущено
    // с /RegServer, /Register, /Unregserver или /Unregister.
    if (!ProcessShellCommand(cmdInfo))
      return FALSE;

    // Было инициализировано единственное окно, поэтому показать и обновить его
    m_pMainWnd->ShowWindow(SW_SHOW);
    m_pMainWnd->UpdateWindow();
    // Вызывать DragAcceptFiles, только если имеется суффикс.
    // В SDI-приложении это происходит после ProcessShellCommand
    return TRUE;
}
```

Те части кода, о которых я хочу рассказать, выделены полужирным. Строка, переданная функции `SetRegistryKey()`, используется для определения ключа реестра, под которым сохраняется информация программы. Вы можете изменить ее по своему желанию. Если я изменю аргумент на `"Horton"`, то информация о программе будет сохранена под следующим ключом реестра:

```
HKEY_CURRENT_USER\Software\Horton\TextEditor\
```

Под этим ключом сохраняются все настройки приложения, включая список последних использованных программой файлов. Вызов функции `LoadStdProfileSettings()` загружает настройки приложения, которые были сохранены в последний раз. Разумеется, при первом запуске программы ничего загружено не будет.

Объект шаблона документа создается динамически внутри `InitInstance()` следующим оператором:

```
pDocTemplate = new CSingleDocTemplate(
       IDR_MAINFRAME,
       RUNTIME_CLASS(CTextEditorDoc),
       RUNTIME_CLASS(CMainFrame), // главное обрамляющее окно SDI
       RUNTIME_CLASS(CTextEditorView));
```

Первый параметр конструктора `CSingleDocTemplate` — символ `IDR_MAINFRAME` — определяет меню и панель инструментов, используемую типом документа. Следующие три параметра определяют документа, главное обрамляющее окно и объекты класса представления, которые должны быть связаны вместе внутри шаблона документа. Поскольку здесь мы имеем дело с SDI-приложением, в программе присутствует только одно представление, управляемое через объект шаблона документа. `RUNTIME_CLASS()` — макрос, позволяющий определять тип объекта класса во время выполнения.

Здесь присутствует много чего другого для настройки экземпляра приложения, о чем беспокоиться не следует. Вы можете добавить в функцию `InitInstance()` свою собственную инициализацию, необходимую приложению.

Функция `Run()`

Класс `CTextEditorApp` наследует функцию `Run()` от базового класса приложения `CWinApp`. Поскольку функция объявлена как `virtual`, вы можете заменить ее версию из базового класса своей собственной, но обычно в этом нет необходимости, так что беспокоиться об этом не нужно.

Функция `Run()` получает все сообщения от Windows, предназначенные приложению, и обеспечивает доставку каждого из них соответствующей функции программы, которая, если она существует, должна его обработать. Таким образом, эта функция продолжается до тех пор, пока выполняется приложение. Она прекращает свою работу, когда вы закрываете приложение.

Таким образом, вы можете разделить работу приложения на четыре шага.

1. Создание объекта приложения `theApp`.
2. Выполнение функции `WinMain()`, которую предоставляет MFC.
3. `WinMain()` вызывает `InitInstance()`, которая, в свою очередь, создает шаблон документа, главное обрамляющее окно, документ и представление.
4. `WinMain()` вызывает `Run()`, которая выполняет главный цикл сообщений программы, получая и обрабатывая сообщения Windows.

Создание MDI-приложения

Теперь давайте создадим MDI-приложение, используя мастер MFC Application Wizard. Назовем проект `Sketcher` — с тем прицелом, чтобы в последующих главах расширить его, построив в конечном итоге программу для рисования. Вам не придется слишком ломать голову над этой процедурой, потому что от процесса создания SDI-приложения, который вы прошли в предыдущем разделе, ее отличают всего три момента. Нужно будет оставить опцию по умолчанию — MDI, а не заменять ее на SDI и все равно отказываться от использования библиотек Unicode. В наборе опций

Document Template Settings (Настройки шаблона документов) диалогового окна MFC Application Wizard потребуется специфицировать расширение файла `ske`. Кроме того, вы должны будете оставить `CView` в качестве базового класса для `CSketcherView` в наборе опций Generated Classes (Сгенерированные классы).

В диалоговом окне Generated Classes (Сгенерированные классы) видно, что для вашего приложения сгенерирован дополнительный класс по сравнению с примером `TextEditor`, как показано на рис. 13.12.

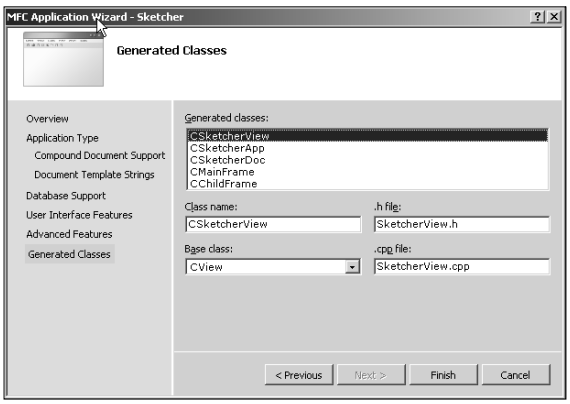

Рис. 13.12. Диалоговое окно Generated Classes для проекта `Sketcher`

Этим дополнительным классом является `CChildFrame`, унаследованный от MFC-класса `CMDIChildWnd`. Этот класс предоставляет обрамляющее окно для представления документа, появляющегося внутри окна приложения, созданного объектом `CMainFrame`. В SDI-приложении имеется только один документ с единственным представлением, так что представление отображается в клиентской области главного обрамляющего окна. В MDI-приложении может существовать множество открытых документов, а каждый документ может иметь множество представлений. Чтобы обеспечить это, каждое представление документа в программе имеет свое собственное дочернее обрамляющее окно, созданное объектом класса `CChildFrame`. Как было показано ранее, на самом деле представление отображается в отдельном окне, которое в точности заполняет клиентскую область обрамляющего окна.

Запуск программы

Выполните сборку программы точно так же, как в предыдущем примере. Затем, после ее запуска, будет получено окно приложения, показанное на рис. 13.13.

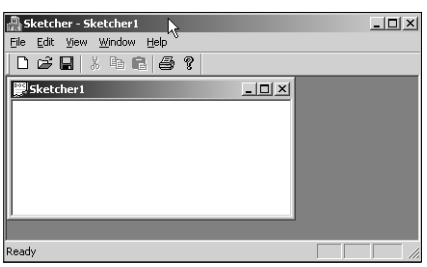

Рис. 13.13. Работа приложения `Sketcher`

В дополнение к главному окну приложения вы получаете отдельное окно документа с заголовком Sketch1, который является начальным именем документа по умолчанию, и если вы сохраните его, он получит расширение `.ske`. Вы можете создавать дополнительные представления для документа, выбирая пункт меню Window⇨New Window (Окно⇨Новое окно). Можно также создать новый документ, выбирая пункт меню File⇨New (Файл⇨Создать), так что в приложении будет одновременно присутствовать два активных документа. Ситуация с двумя активными документами, для каждого из которых открыто по два представления, показана на рис. 13.14.

Пока вы не можете создавать никаких данных в приложении, поскольку для этого не был добавлен соответствующий код, однако весь код для создания документов и представлений уже включен мастером MFC Application Wizard.

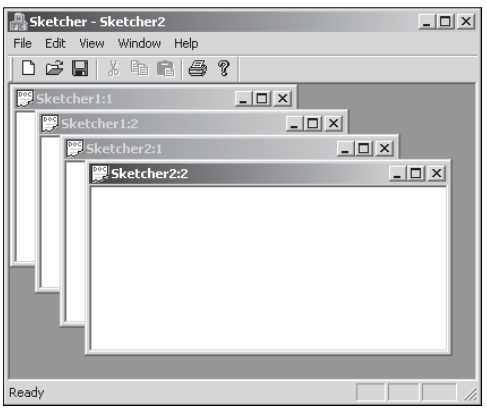

Рис. 13.14. Работа приложения `Sketcher` с двумя активными документами

Резюме

В этой главе внимание было сосредоточено в основном на способах использования мастера MFC Application Wizard. Вы увидели базовые компоненты программ MFC, генерируемые этим мастером как для SDI-, так и для MDI-приложений. Все примеры MFC создавались с помощью мастера MFC Application Wizard, так что хорошей идеей будет иметь в виду общую структуру и отношения между классами. Возможно, вы пока не чувствуете себя уверенно в том, что касается деталей, но пока беспокоиться не стоит. Когда мы начнем разрабатывать приложения в последующих главах, все существенно прояснится.

Ниже перечислены ключевые моменты, раскрытые в этой главе.

- Мастер MFC Application Wizard генерирует полный работающий каркас приложения Windows, которое вы можете адаптировать под свои требования.
- Мастер создания приложений может генерировать приложения с однодокументным интерфейсом (SDI), которые работают с единственным документом в одном представлении, или программы с многодокументным интерфейсом (MDI), которые могут обрабатывать множество документов с множеством представлений одновременно.

- Четыре важнейших класса SDI-приложения, которые унаследованы от фундаментальных классов:
  - класс приложения;
  - класс обрамляющего окна;
  - класс документа;
  - класс представления.
- Программа может иметь только один объект приложения. Он определяется автоматически мастером MFC Application Wizard в глобальном контексте.
- Объект класса документа сохраняет специфичные для приложения данные, а класс представления отображает содержимое объекта документа.
- Объект шаблона документа используется для связывания документа, представления и окна. Для SDI-приложения это делает класс `CSingleDocTemplate`, а для MDI-приложения — класс `CDocTemplate`. Оба эти класса — фундаментальные и, как правило, нет необходимости наследовать от них какие-то специфичные для приложения версии.

Упражнения

Предложить примеры программирования для этой главы не представляется возможным, поскольку на самом деле в ней были представлены лишь базовые механизмы создания приложений MFC. Также нет особого смысла придумывать упражнения и ответы к ним, поскольку вы либо и так видите ответы на экране, либо можете найти их в тексте.

Тем не менее, исходные коды упражнений и их решения можно загрузить с Web-сайта издательства.

1. Что такое отношение между документом и представлением?
2. В чем состоит назначение шаблона документов в MFC-программе для Windows?
3. Почему следует проявлять внимательность и планировать структуру программы заранее при использовании мастера создания приложений?
4. Напишите код простой программы текстового редактора. Выполните сборку отладочной и рабочей версий и оцените типы и размеры файлов, полученных в обоих случаях.
5. Сгенерируйте приложение текстового редактора несколько раз, пробуя различные оконные стили в окне Advanced Options (Дополнительные параметры) мастера создания приложений.