bhv
www.bhv.ru
www.bhv.kiev.ua

Ю. Тихомиров

MFC

САМОУЧИТЕЛЬ

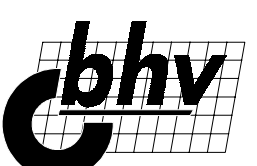
- Обзор классов
- Поддержка работы с графикой
- Архитектура “документ/представление”
- Работа с базами данных

Дискета содержит примеры приложений и тексты программ

Библиотека для разработки Windows-приложений

Юрий Тихомиров

САМОУЧИТЕЛЬ MFC



Дюссельдорф ◆ Киев ◆ Москва ◆ Санкт-Петербург

УДК 681.3.06

Библиотека MFC — мощный и гибкий инструмент разработки Windows-приложений на базе языка Visual C++. Книга содержит подробную информацию об основных классах библиотеки и их компонентах. Рассмотрены принципы создания одно- и многодокументных приложений, все типы окон, элементы управления и работа с базами данных. Практические примеры помогут усвоить теоретический материал и одновременно освоить современный стиль программирования. В приложении излагаются основы языка C++, что позволяет рекомендовать книгу не только подготовленным программистам, желающим познакомиться с библиотекой MFC, но и новичкам.

Для программистов, разрабатывающих приложения для Windows

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зав. редакцией	*Наталья Таркова*
Редактор	*Ирина Агафонова*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн обложки	*Татьяны Емельяновой*
Зав. производством	*Николай Тверских*

Тихомиров Ю. В.
Самоучитель MFC. — СПб.: БХВ — Санкт-Петербург, 2000. — 640 с.: ил.

ISBN 5-8206-0096-7

Лицензия ЛР № 065953 от 15.06.98. Подписано в печать 17.02.2000.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 51,6.
Тираж 5000 экз. Заказ
"БХВ — Санкт-Петербург", 198005, Санкт-Петербург, Измайловский пр., 29.

Гигиеническое заключение на продукцию, товар, № 77.99.1.953.П.950.3.99
от 01.03.1999 г. выдано Департаментом ГСЭН Минздрава России.

Отпечатано с готовых диапозитивов
в Академической типографии "Наука" РАН.
199034, Санкт-Петербург, 9 линия, 12.

Содержание

Введение

Получаю письма: "Помогите стать актером".
Отвечаю: "Бог поможет!"
Фаина Раневская

Приходит программист на стрельбы. Делает десять выстрелов — все мимо. Инструктор в недоумении разводит руками: "Как же так?" Программист подходит вплотную к мишени, нажимает на курок — естественно, попадает в "десятку" и говорит: "У меня все в порядке — это у вас не работает". Это, конечно же, анекдот, но любой человек, занимающийся программированием, подтвердит, что сам не раз оказывался в ситуации, когда был полностью уверен, что протестировал свою программу вдоль и поперек и "выловил" все ошибки, и все же программа, оказавшись в руках заказчика, непостижимым образом начинала выдавать всякую чушь. Не зря же считается, что любая прекрасно работающая программа содержит как минимум две ошибки.

К чему я это рассказываю? В названии книги, которую вы держите в руках, есть слово "самоучитель". Другими словами, вы в той или иной степени — новичок. И именно поэтому вам мой первый совет — разрабатывая любую, пусть самую простую программу, всегда старайтесь предусмотреть все, даже самые невероятные ситуации, которые могут возникнуть в процессе ее эксплуатации. Помните: ответственность за сбои лежит на программисте, а не на пользователе.

Данная книга посвящена программированию для системы Windows. Основная проблема, с которой сталкивается программист, начиная работать в некоторой операционной системе, состоит в понимании фундаментальных принципов и моделей, включенных в ее архитектуру. И только проникнувшись "образом мышления Windows", начинаешь понимать, что программирование под Windows не сложнее любого другого.

Какой бы язык программирования вы не выбрали, очевидно, что базой будет Windows API (Application Programming Interface, Интерфейс прикладного программирования): только освоив его, можно писать программы, использующие все возможности, предоставляемые операционной системой. Одна-

ко сложность современных приложений поднялась на такой уровень, что разработка коммерческого программного обеспечения с использованием только Windows API и языка C уже не может удовлетворить программиста. В настоящее время предлагается два подхода к построению сложного ПО. Первый — это использование систем визуального программирования, а второй — применение библиотек классов, которые самостоятельно выполняют массу черновой работы и при этом гораздо в большей степени, нежели сама система Windows, поддерживают программиста, структурируя и облегчая его деятельность. Не будем заниматься сравнением достоинств и недостатков — каждый вправе сам решить, что ему больше подходит.

Но если вас заинтересовала эта книга, то рискну предположить, что вы, как и я, предпочитаете создавать все сами, а не полагаться на готовые решения, несмотря на то, что поначалу много времени будет уходить на рутину. А затем, по мере накопления собственного опыта, можно перейти на хорошо известный принцип "ножниц и клея", перенося уже написанные и отлаженные фрагменты из одной программы в другую. Это обычная и, на мой взгляд, правильная практика.

Основной темой книги является изучение библиотеки MFC (Microsoft Foundation Classes, Базовые классы Microsoft). Несмотря на то, что эта библиотека сложна и многогранна, и на сегодняшний день существует более десяти ее версий, для начала работы с ней вполне достаточно знакомства с языком программирования C++. Более того, если вы еще не знакомы с этим языком — не отчаивайтесь. Поставляемые исходные тексты библиотеки MFC могут значительно помочь одновременно изучить и C++. Новичкам даже не потребуется знание принципов программирования в Windows. Дело в том, что в компилятор Visual C++ включено специальное средство, позволяющее автоматизировать процесс разработки приложения, — мастер MFC AppWizard, который создаст для вас работающую программу. Конечно, это будет только шаблон, и наполнять его конкретным содержанием придется самостоятельно. Но первоначальное знакомство со сгенерированным кодом принесет начинающим программистам значительную пользу.

Теперь несколько слов о методике изложения материала. Она несколько отличается от принятой для самоучителей и базируется на двух основных подходах:

- освоение нового материала на примере подробного разбора конкретного кода;
- разбор теоретических основ с описанием некоторых наиболее часто употребляемых функций и переменных различных классов.

На мой взгляд, такая методика позволит значительно лучше освоить спектр возможностей, предоставляемых библиотекой MFC. Сначала вы рассматриваете конкретные примеры, а потом пытаетесь реализовать уже собственные

идеи, используя приведенные описания компонентов различных классов. На основе своего, достаточно богатого опыта, могу вас заверить, что без самостоятельной работы, ограничиваясь только разбором кем-то написанного кода, научиться программировать невозможно.

И еще одно замечание по этому поводу. Первоначально я предполагал в конце каждой главы поместить вопросы и упражнения, чтобы помочь закрепить прочитанный материал. Однако в процессе работы над книгой я отказался от этой идеи. И связано это опять же с попыткой подтолкнуть вас к самостоятельной работе. На прилагаемой дискете вы найдете многочисленные примеры, в которых реализованы подходы и возможности либо совсем не описанные, либо только упоминающиеся в тексте. Попробуйте их найти и самостоятельно разработать на их основе работающие приложения. Примеры написаны для версии компилятора Visual C++ 5.0, в которую включена библиотека MFC версии 4.22. Однако рассматриваются и некоторые особенности библиотеки версии 4.23, поставляемой с пакетом Visual C++ 6.0.

Материала по MFC настолько много, что мне приходилось достаточно долго решать, какие именно вопросы изложить подробно, какие просто упомянуть, а какие вообще не затрагивать, и в какой последовательности все это изложить. В результате структура книги, состоящей из 21 главы, оказалась следующей:

- Первые две главы знакомят со структурой библиотеки MFC и процессом создания вашего первого приложения.
- В главе 3 проводится подробный разбор минимальной программы для Windows, написанной на базе библиотеки MFC и класса приложения *CwinApp*.
- Глава 4 знакомит вас с различными типами окон и одним из основополагающих классов библиотеки — *CWnd*.
- Глава 5 посвящена подробному изложению одного из основных механизмов Windows — работы с сообщениями.
- О том, как "распределить нагрузку" между классами библиотеки при создании любого приложения, рассказывается в главе 6.
- Следующие четыре главы — с 7 по 10 — посвящены вопросам организации процесса вывода информации на экран, а точнее — в окно.
- Вопросы взаимодействия программы и пользователя рассматриваются в главах 11 и 12. В главе 11 подробно разбираются вопросы работы с панелями инструментов и строкой состояния, а в главе 12 вы на примерах познакомитесь с разнообразными меню.
- Глава 13 целиком посвящена описанию процесса создания многодокументных приложений.

- В главе 14 рассматриваются основные принципы создания и работы с блоками диалога. Описывается класс *CDialog*, предназначенный для решения этих задач, и класс *CColorDialog*, поддерживающий работу с системными (стандартными) блоками диалога, использование которых позволяет настраивать практически все параметры (открытие и сохранение файла, выбор шрифта и цвета, поиск и т. д.), необходимые для работы приложений.
- В главе 15 собрана информация, которая позволит создавать наборы свойств, состоящие из нескольких блоков диалога с вкладками, и мастера, также представляющие собой наборы блоков диалога. Различие между ними заключается в том, что в наборе свойств пользователь переходит от одного блока диалога к другому, выбирая соответствующую вкладку, доступную в произвольный момент времени, а в мастере переход от одного к другому может осуществляться только последовательно. Несмотря на имеющиеся различия, эти элементы управления по способу работы с ними очень схожи и базируются на одних и тех же классах — *CPropertySheet*, *CPropertyPage* и *CTabCtrl*.
- В главе 16 сосредоточена информация об элементах управления (Controls). Естественно, что я не смог рассмотреть даже большую их часть, но то, что вы найдете в этой главе, поможет вам легко разобраться и со всеми остальными элементами управления.
- В части VI (главы с 17 по 19) сосредоточено все, что необходимо знать о поддержке со стороны библиотеки MFC архитектуры "документ/представление" (document/view architecture). Данная архитектура напрямую не поддерживается Windows, поэтому описания только классов и функций явно недостаточно. Здесь представлена взаимосвязь различных классов, последовательность вызовов тех или иных функций. Эти знания позволят вам в дальнейшем "минимизировать трудозатраты" и переопределять только те функции, новое содержание которых наполнит нужным вам смыслом разрабатываемые приложения.
- Две заключительные главы — 20 и 21 — знакомят вас с вопросами создания приложений для работы с базами данных.
- Для тех, кто не знаком с языком программирования C++, я включил в книгу приложение 1, в котором вы найдете введение в объектно-ориентированное программирование и язык C++, а также некоторые нюансы его использования.
- Система Windows — это система, основанная на сообщениях. Поэтому я включил в книгу приложение 2, где представлены различные типы используемых сообщений.
- Неотъемлемой частью книги являются приложения-примеры, записанные на прилагаемой дискете. Их достаточно много и они охватывают как

то, что можно найти на ее страницах, так и те вопросы, которым из-за недостатка места не было уделено внимание.

В заключение я хотел бы поблагодарить редактора книги Ирину Агафонову, а также всех сотрудников издательства "БХВ — Санкт-Петербург", принимавших участие в подготовке рукописи к печати.

Все замечания и пожелания можно присылать по электронной почте по адресу **bhv@mail.nevalink.ru**.

Часть I

Библиотека Microsoft Foundation Classes (MFC)

ГЛАВА 1

Знакомьтесь — библиотека классов MFC

Чего не понимают, тем не владеют.
Иоганн Вольфганг Гете

Для того чтобы выполнять какую-либо работу на профессиональном уровне, недостаточно поверхностного знакомства с тем инструментарием, которым собираешься пользоваться. Необходимо совершенно четко и ясно представлять себе внутреннюю структуру и логику работы используемого средства. В полной мере это относится и к широко применяемой в настоящее время для построения (создания) программных продуктов библиотеке классов MFC (Microsoft Foundation Class). К MFC следует подходить именно как к инструменту, который, принимая на себя большую часть черновой работы, требует от нас не просто знакомства с ним, но и глубокого изучения. Только знание всех нюансов построения и возможностей этой библиотеки позволит нам быстро и легко создавать программы любой степени сложности. В противном случае никогда не будет уверенности, что созданное приложение при любых условиях работает правильно — слишком много скрыто от программиста. Функции вызываются не только из недр Windows, но и из недр библиотеки. Ни о каком последовательном вызове функций речь не идет. Данные готовятся неизвестно где и неизвестно кем и когда обрабатываются. Так что какая уж тут уверенность.

Что же представляет собой библиотека MFC? Это набор классов, охватывающих большую часть функциональных возможностей операционных систем Microsoft Windows, а также предоставляющих разработчику значительное количество не только очень мощных дополнительных классов, но и целые механизмы, которые, не нарушая идеологию операционной системы, существенно ее расширяют и ... упрощают.

Перед создателями библиотеки стояла задача создания объектно-ориентированного интерфейса для Windows, удовлетворяющего следующим основным целям проектирования программных продуктов:

- сокращение усилий по программированию приложений для Windows;
- скорость выполнения программ, написанных с использованием библиотеки, должна быть сопоставима с программами, написанными на языке C с использованием Win32 API;
- минимальный размер вспомогательного кода;
- способность напрямую вызывать любую C-функцию Win32 API;
- легкость использования Win32 API в C++ должна быть такая же, как и при использовании традиционного C.

Надо сказать, что поставленная задача была решена на очень высоком уровне. Созданная библиотека классов охватывает все компоненты Windows — окна, блоки диалога, контексты устройств, общие объекты GDI (битовые образы и кисти), элементы управления и многие другие стандартные элементы. Суть программирования под Windows — обработка событий — предоставлена программистам в удобном и привычном виде. Классы библиотеки полностью вобрали в себя многочисленные операторы *switch*, которые так загромождают программы, написанные на языке C. Наряду с этим вы можете совершенно свободно смешивать вызовы библиотеки классов с прямыми вызовами Win32 API. Кроме того, за сравнительно небольшой промежуток времени корпорацией Microsoft было разработано несколько версий библиотеки MFC, которые становились все мощнее и удобнее. Вместе с Visual C++ 6.0 поставляется версия 4.23 библиотеки, и есть все основания считать, что ее развитие будет продолжено. К тому же поддержка библиотеки MFC компиляторами и средствами разработки ПО, созданными другими компаниями, позволяет сконцентрироваться именно на ней. Но давайте рассмотрим все по порядку. Начнем с макросов, глобальных функций и переменных, которые, хотя и не входят непосредственно в библиотеку, очень тесно с ней связаны и значительно облегчают программирование, а затем перейдем к общему обзору библиотеки.

Макросы, глобальные функции и переменные

Знакомство с библиотекой будет неполным и не даст существенных преимуществ при работе с ней, если мы не рассмотрим включенные в нее макросы, глобальные функции и переменные.

Все макросы, глобальные функции и переменные можно разбить на несколько категорий: основные, работа с базами данных, работа в Internet,

OLE и элементы управления OLE. Поскольку в данной книге мы не будем подробно рассматривать работу с базами данных, в сети и с использованием OLE (этим вопросам будут посвящены следующие книги), то ограничимся изучением только основных макросов, глобальных функций и переменных. Начнем, естественно, с начала. Практически все глобальные функции начинаются с префикса "Afx". Исключение составляют большинство функций для работы с базами данных и функции, обеспечивающие обмен данными. Для всех без исключения глобальных переменных применяется префикс "Afx", а все макросы записываются заглавными буквами.

Типы данных

Большинство типов данных полностью соответствуют представленным в SDK, однако есть типы, специфичные только для MFC.

BOOL	Булевское значение
BSTR	32-битный указатель на символ
BYTE	8-битное целое без знака
COLORREF	32-битное значение, используемое для задания цвета
DWORD	32-битное целое без знака или адрес
LONG	32-битное целое со знаком
LPARAM	32-битное значение, посылаемое в качестве параметра в оконную процедуру или функцию обратного вызова
LPCSTR	32-битный указатель на константную строку символов
LPSTR	32-битный указатель на строку символов
LPCTSTR	32-битный указатель на константную строку символов, которая переносима в Unicode и DBCS
LPTSTR	32-битный указатель на строку символов, которая переносима в Unicode и DBCS
LPVOID	32-битный указатель на неопределенный тип данных
LRESULT	32-битное значение, возвращаемое из оконной процедуры или функции обратного вызова
UINT	32-битное целое без знака для Win32 и 16-битное — для Windows 3.x
WNDPROC	32-битный указатель на оконную процедуру
WORD	16-битное целое без знака
WPARAM	Значение, посылаемое в качестве параметра в оконную процедуру или функцию обратного вызова; 32-битное для Win32 и 16-битное для Windows 3.x

POSITION	32-битное целое без знака, используемое для обозначения позиции элемента в коллекции
LPCRECT	32-битный указатель на немодифицируемую структуру RECT
HINSTANCE	32-битное целое без знака для идентификации дескриптора экземпляра приложения

Примечание

Функция обратного вызова или CallBack-функция вызывается операционной системой по указателю, переданному ей ранее приложением. Это весьма удобный и часто используемый механизм подключения к стандартным функциям специфических обработчиков.

Имеются также некоторые другие типы данных, которые будут описаны в соответствующих местах.

Получение информации о приложении

Когда вы пишете приложение с использованием классов MFC, вы создаете единственный объект, производный от *CWinApp*. Для получения из любого места программы информации об этом объекте и связанных с ним параметрах разработчики предоставили следующие функции.

`CWinApp* **AfxGetApp**()`

Возвращает указатель на единственный объект *CWinApp* приложения.

`LPCNSRT **AfxGetAppName**()`

Возвращает строку, заканчивающуюся нулем, содержащую имя приложения.

`HINSTANCE **AfxGetInstanceHandle**()`

Возвращает дескриптор текущего приложения для исполняемого файла (.EXE); в случае, если функция вызвана из динамически подключаемой библиотеки, связанной с USRDLL-версией MFC, то возвращается HINSTANCE этой библиотеки DLL.

`CWnd* **AfxGetMainWnd**()`

Возвращает значение переменной *m_pMainWnd* объекта приложения, если оно не является сервером OLE, или указатель на активное главное окно приложения — в противном случае.

`HINSTANCE **AfxGetResourceHandle**()`

Возвращает дескриптор загруженных по умолчанию ресурсов приложения.

`void **AfxSetResourceHandle**(HINSTANCE hInstResource)`

Устанавливает дескриптор экземпляра приложения или модуля .EXE или .DLL, из которого будут загружены ресурсы приложения.

```
LPCTSTR AFXAPI AfxRegisterWndClass(
  UINT nClassStyle,
  HCURSOR hCursor = 0,
```

```
HBRUSH hbrBackground = 0,
HICON hIcon = 0)
```

Возвращает строку, заканчивающуюся нулем, которая содержит имя зарегистрированного класса. В качестве параметров в эту функцию передаются: *nClassStyle* — стиль или комбинация стилей класса окна Windows, которые будут рассмотрены при описании регистрации класса окна, *hCursor* — дескриптор ресурса курсора, *hbrBackground* — дескриптор ресурса кисти, определяющей цвет фона, *hIcon* — дескриптор ресурса-пиктограммы.

```
BOOL AFXAPI AfxRegisterClass(WNDCLASS* lpWndClass)
```

Возвращает значение TRUE при успешной регистрации класса окна и FALSE при ошибке. В качестве параметра в функцию передается указатель на структуру WNDCLASS, которая должна содержать информацию, необходимую для успешной регистрации. При использовании этой функции для регистрации класса окна в модуле DLL следует иметь в виду, что имя зарегистрированного класса автоматически удаляется из системы при выгрузке библиотеки из памяти.

```
HINSTANCE AFXAPI AfxLoadLibrary(LPCTSTR lpszModuleName)
```

Загружает в память модуль (.DLL или .EXE файл), имя которого передается в качестве параметра *lpszModuleName*. Если загрузка прошла успешно, то возвращается дескриптор модуля и увеличивается счетчик ссылок для этой библиотеки; в противном случае возвращается NULL.

```
BOOL AFXAPI AfxFreeLibrary(HINSTANCE hInstLib)
```

Уменьшает счетчик ссылок для загруженного модуля DLL, определяемого параметром *hInstLib*. Когда счетчик ссылок становится равным нулю, модуль выгружается из адресного пространства текущего процесса.

Модель объекта времени выполнения (run-time object)

Как будет показано дальше, все классы, производные от *CObject* и *CruntimeClass*, приобретают определенные возможности по обслуживанию объектов, включающие динамическую информацию о классе (run-time class information), сериализацию и динамическое создание объекта.

Здесь описаны макросы, которые облегчают использование этих возможностей.

```
DECLARE_DYNAMIC(className)
```

Разрешает доступ к динамической информации об объекте класса (должен использоваться при объявлении класса); *className* — действительное C++-имя класса без кавычек.

```
IMPLEMENT_DYNAMIC(className, baseClassName)
```

Генерирует код, необходимый для получения динамической информации об объекте класса, производного от *CObject* (должен использоваться в реализации класса).

DECLARE_DYNCREATE(className)

Разрешает динамическое создание объекта, производного от *CObject*, и получение доступа к динамической информации об этом объекте (должен использоваться при объявлении класса).

IMPLEMENT_DYNCREATE(className, baseClassName)

Разрешает динамическое создание объекта (должен использоваться в реализации класса) и доступ к динамической информации о классе. Приложение использует эту возможность для динамического создания нового объекта, например, когда требуется прочитать объект из файла на диске во время процесса сериализации.

При использовании этих макросов вы получаете возможность использовать макрос RUNTIME_CLASS и функцию *CObject::IsKindOf* для определения класса вашего объекта во время выполнения программы.

DECLARE_SERIAL(className)

Генерирует C++-код, необходимый для сериализации объекта класса, производного от *CObject* (должен использоваться при объявлении класса) и доступа к динамической информации о классе. Определение этого макроса включает в себя все функциональные возможности, предоставляемые обоими макросами DECLARE_DYNAMIC и DECLARE_DYNCREATE.

IMPLEMENT_SERIAL(className, baseClassName, wSchema)

Генерирует C++-код, необходимый для сериализации объекта класса, производного от *CObject* (должен использоваться в реализации класса) и доступа к динамической информации о классе. Параметр *wSchema* — переменная типа UINT, определяющая "номер версии" приложения, которой может быть закодирован архив, для возможности применения процесса десериализации. Этот номер не должен быть равен –1.

RUNTIME_CLASS(className)

Для объектов, производных от *CObject* и объявленных с использованием макросов DECLARE_DYNAMIC, DECLARE_DYNCREATE или DECLARE_SERIAL, возвращает указатель на структуру *CRuntimeClass*, которая соответствует имени класса *className*.

Диагностика объектов

Диагностический сервис, предоставляемый библиотекой MFC, значительно упрощает отладку приложений. Он включает в себя макросы и глобальные функции, которые позволяют отслеживать распределение памяти, содержимое дампа объекта и осуществлять печать отладочной информации во время выполнения.

Основные макросы

ASSERT(booleanExpression)

Прерывает выполнение программы (в отладочной версии библиотеки), если вычисляемое выражение *booleanExpression* равно FALSE, и печатает сообщение об ошибке в следующей форме:

`assertion failed in file <имя_файла> in line <номер_строки>`,

где <номер строки> определяет строку, в которой произошла ошибка.

ASSERT_KINDOF(className, pObject)

Проверяет, является ли *pObject* объектом класса *className*, где *className* — имя класса, производного от класса *Cobject*; этот макрос работает только, если в области объявления класса используется макрос DECLARE_DYNAMIC или DECLARE_SERIAL.

ASSERT_VALID(pObject)

Используется для оценки доступности внутреннего состояния *pObject* (сначала проверка *pObject* на NULL, затем вызов его метода *AssertValid)*. Если хотя бы одна проверка приводит к возникновению ошибки, то выводится сообщение, аналогичное ASSERT.

TRACE(exp)

Позволяет вывести на экран форматированную строку, определяемую выражением *exp*, аналогично функции *printf* для C-программ в консольных программах, например, `TRACE("Проверка вывода строк %s и чисел %d", "Отладка программы", 5)`.

TRACE0 — TRACE3

Упрощенные версии макроса TRACE, позволяющие вывести форматированную строку с числом аргументов от 0 до 3.

Рассмотренные макросы работают только в отладочной версии библиотеки.

VERIFY(booleanExpression)

Действует аналогично макросу ASSERT, но для рабочей версии библиотеки.

Основные глобальные переменные

CDumpContext **afxDump**

Предопределенная переменная, которая позволяет послать информацию в окно отладчика.

BOOL **afxTraceEnable**

Используется для разрешения или запрещения работы макроса TRACE.

Форматирование строк и окна сообщений

Эта группа содержит три глобальные функции: две для работы со строками и одну для вывода произвольного сообщения в блок диалога.

```
void AfxFormatString1(
  CString& rString,
  UINT nIDS,
  LPCTSTR lpsz)
```

Загружает строку из ресурсов, определяемую идентификатором *nIDS*, в объект *rString* с одновременной заменой шаблона "%1" на строку, указанную в *lpsz*. Например, если в ресурсах определена строка "Для профессионального программирования 32-разрядных приложений — читайте книгу %1", а *lpsz* указывает на

строку "<Программирование на Visual C++ 6.0>", то после выполнения этой функции *rString* будет содержать строку "Для профессионального программирования 32-разрядных приложений — читайте книгу <Программирование на Visual C++ 6.0>".

```
void AfxFormatString2(
  CString& rString,
  UINT nIDS,
  LPCTSTR lpsz1,
  LPCTSTR lpsz2)
```

Аналогична функции *AfxFormatString1*, но используется два шаблона ("%1" и "%2") и две строки для подстановки, соответственно.

Последняя функция этой группы имеет две формы: со строкой текста и с идентификатором ресурса:

```
int AfxMessageBox(
  LPCTSTR lpszText,
  UINT nType = MB_OK,
  UINT nIDHelp = 0)
```

и

```
int AfxMessageBox (
  UINT nIDPrompt,
  UINT nType = MB_OK,
  UINT nIDHelp = (UINT) -1)
```

Выводит текст в окно сообщений. Первая форма этой функции выводит в окно сообщений строку *lpszText* и использует идентификатор *nIDHelp* для перехода к соответствующей теме справки при нажатии клавиши <F1>. Вторая форма функции использует для определения текста идентификатор строки ресурса, задаваемый параметром *nIDPrompt*. Если в качестве *nIDHelp* используется значение по умолчанию, равное –1, то *nIDPrompt* задает тему справки.

Обе функции возвращают одно из следующих значений:

- IDABORT — нажата кнопка Abort
- IDCANCEL — нажата кнопка Cancel или нажата клавиша <Esc>
- IDIGNORE — нажата кнопка Ignore
- IDNO — нажата кнопка No
- IDOK — нажата кнопка OK
- IDRETRY — нажата кнопка Retry
- IDYES — нажата кнопка Yes
- 0 — окно сообщений не может быть создано.

Параметр *nType* определяет тип окна сообщений и может принимать одно из следующих значений:

- MB_ABORTRETRYIGNORE — содержит три кнопки Abort, Retry и Ignore (Стоп, Повтор и Пропустить)

- MB_OK — содержит только кнопку OK
- MB_OKCANCEL — содержит две кнопки OK и Cancel (OK и Отмена)
- MB_RETRYCANCEL — содержит две кнопки Retry и Cancel (Повтор и Отмена)
- MB_YESNO — содержит две кнопки Yes и No (Да и Нет)
- MB_YESNOCANCEL — содержит три кнопки Yes, No и Cancel (Да, Нет и Отмена)

Кроме того, можно использовать следующие константы для определения степени модальности окна сообщений:

- MB_APPLMODAL — модальность на уровне пользователя
- MB_SYSTEMMODAL — модальность на уровне системы
- MB_TASKMODAL — модальность на уровне процесса

Примечание

В скобках приведены названия для русскоязычной версии системы Windows.

Помимо кнопок, в окне сообщений можно разместить одну из предопределенных (системных) пиктограмм (рис. 1.1—1.4).

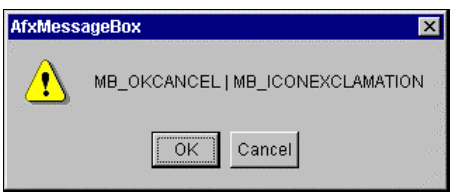


Рис. 1.1. MB_ICONEXCLAMATION — восклицательный знак

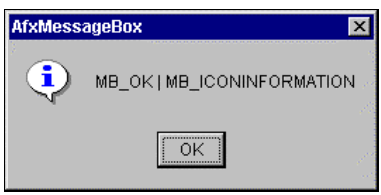


Рис. 1.2. MB_ICONINFORMATION — пиктограммы в виде символа "i"

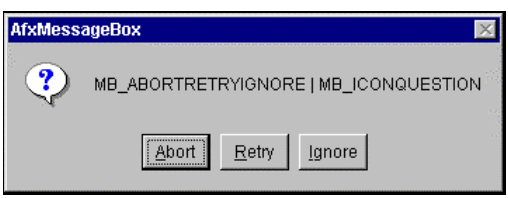


Рис. 1.3. MB_ICONQUESTION — знак вопроса

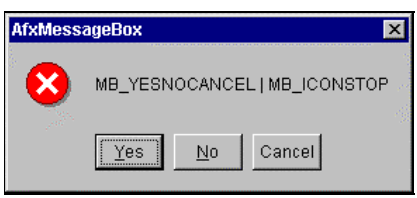


Рис. 1.4. MB_ICONSTOP — пиктограмма "стоп"

При использовании в блоке сообщений более одной кнопки можно также установить кнопку по умолчанию:

- MB_DEFBUTTON1 — по умолчанию первая кнопка
- MB_DEFBUTTON2 — по умолчанию вторая кнопка
- MB_DEFBUTTON3 — по умолчанию третья кнопка

А теперь переходим собственно к структуре библиотеки MFC.

Иерархия классов MFC

На рис. 1.5 представлена иерархия основных категорий классов библиотеки MFC. Как и положено, каждый новый производный класс обладает как свойствами, унаследованными им от родительских классов, так и приобретает новые, характерные только для данного класса. На вершине иерархии находится единственный базовый класс *CObject*. В зависимости от отношения к этому классу все остальные классы библиотеки MFC можно условно разбить на две группы: классы, производные от *Cobject*, и классы, не зависящие от него.

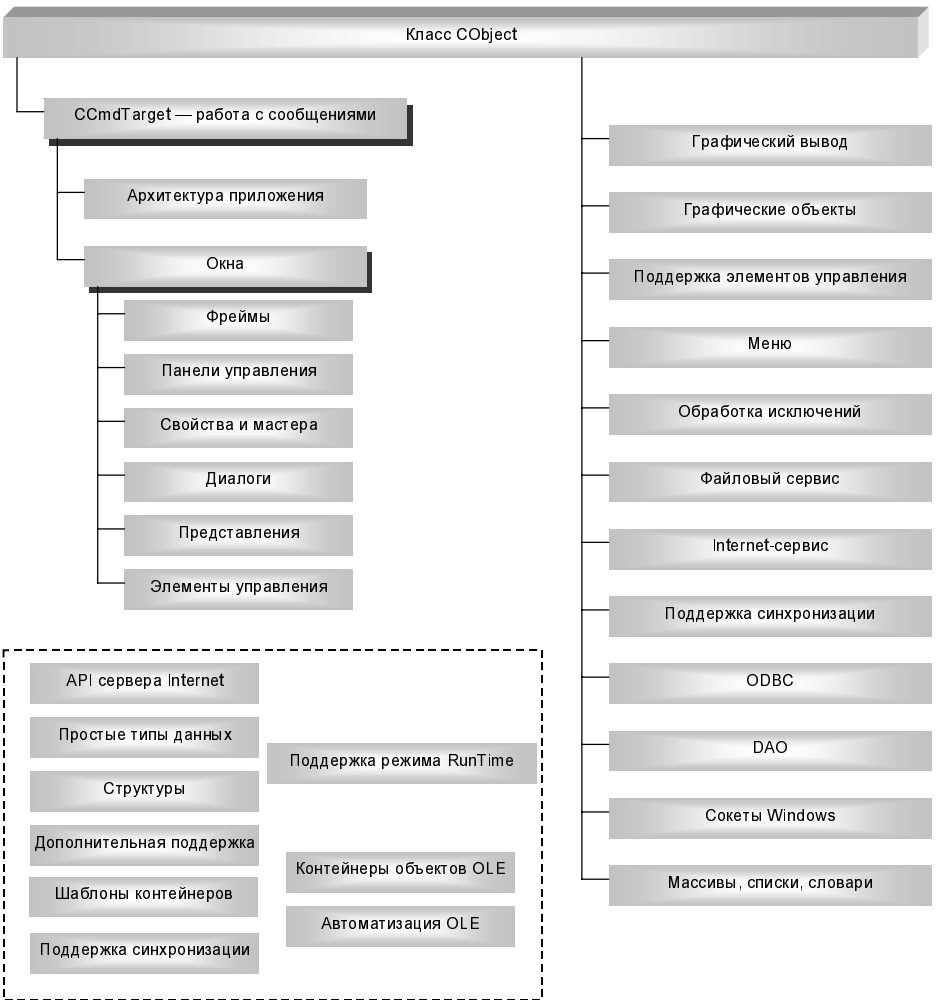


Рис. 1.5. Иерархия классов MFC по категориям

О структуре и возможностях библиотеки классов MFC мы будем говорить на протяжении всей книги. Здесь же я приведу описание только вершины представленной иерархии — класса *CObject*, который незримо входит практически во все остальные классы, с которыми мы будем работать.

CObject — вершина иерархии классов

Большинство классов в библиотеке MFC являются производными от единственного класса *CObject* — корня иерархической структуры. Методы и элементы данных этого класса представляют наиболее общие свойства производных от него классов MFC. Основное назначение данного класса заключается в предоставлении всем своим производным классам следующих возможностей:

- хранение информации о классе объекта во время выполнения;
- поддержка сериализации и диагностики объекта.

Единственной переменной класса *CObject* является статическая переменная *classObject* типа *CRuntimeClass*, которая хранит информацию об ассоциированном с классом *Cobject* объекте во время выполнения программы.

Класс *CObject* не поддерживает множественного наследования, т. е. создаваемые вами классы могут иметь только один *CObject* в качестве базового, и он должен быть "самым левым" в иерархии. Однако такой производный класс позволяет иметь "справа от себя" в качестве базовых несколько структур и/или классов, не являющихся производными от него. Например,

```
class A : public Cobject          class B          class C
{                                 {                {
};                                };               };
class D : public A, B, C
{
};
```

Как уже отмечалось, каждый класс, производный от *Cobject*, ассоциируется со структурой *CRuntimeClass*, которую можно использовать для получения информации об объекте или его базовом классе во время выполнения программы.

Для получения динамической информации использование этой структуры должно сопровождаться обязательным включением одного из макросов DECLARE_DYNAMIC, DECLARE_DYNCREATE, DECLARE_SERIAL в области объявления и соответствующих им макросов IMPLEMENT_DYNAMIC, IMPLEMENT_DYNCREATE, IMPLEMENT_SERIAL в области реализации.

Наряду с рассмотренными функциями, для получения структуры класса времени выполнения широко используется макрос RUNTIME_CLASS, который позволяет получить эту структуру по имени класса C++.

В заключение обзора класса *CObject* — корня иерархической структуры библиотеки MFC — рассмотрим четыре уровня функциональных возможностей, которые он нам предоставляет:

- базовые функциональные возможности — отсутствует поддержка динамической информации о классе и сериализации, но включается управление диагностикой памяти;
- базовые функциональные возможности плюс поддержка динамической информации о классе;
- базовые функциональные возможности плюс поддержка динамической информации о классе и динамическое создание;
- базовые функциональные возможности плюс поддержка динамической информации о классе, динамическое создание и сериализация.

При создании классов, производных от *CObject*, уровень их функциональных возможностей задается выбором определенных макросов объявления и реализации.

Динамическую информацию о классе *CObject* поддерживает посредством своего метода *IsKindOf*, который позволяет определить, относится ли объект к определенному классу или, может быть, является производным от него. Эта функция позволяет осуществить безопасное приведение типа в производном классе. Ее аргументом является объект класса *CRuntimeClass*, который можно получить, используя макрос RUNTIME_CLASS с именем класса. Метод *IsKindOf* не поддерживает множественного наследования или виртуальных классов, хотя при необходимости можно использовать множественное наследование для классов, производных от классов библиотеки MFC.

CObject также поддерживает динамическое создание, т. е. создание объекта определенного класса во время выполнения программы. Объект создается методом *CreateObject* структуры *CRuntimeClass*, как показано в примере:

```
CRuntimeClass* pRuntimeClass = RUNTIME_CLASS(CMyClass);
CObject* pObject = pRuntimeClass->CreateObject();
ASSERT(pObject->IsKindOf(RUNTIME_CLASS(CMyClass)));
```

Надеюсь, теперь у вас сложилось общее впечатление о возможностях и мощи библиотеки классов MFC и можно перейти к рассмотрению вопросов построения приложения на ее базе. Как и любое другое средство, данная библиотека требует выполнения некоторых (очень немногих) условий. Заодно познакомимся с мастером AppWizard, который выполнит за вас большую часть подготовительной работы.

ГЛАВА 2

Создание приложений на базе библиотеки классов MFC

Лучше делать новости, чем рассказывать о них.
Уинстон Черчиль

Для того чтобы написать какое-либо приложение, прежде всего необходимо создать проект. Разработчики фирмы Microsoft постарались (уже давно) и создали специальные мастера, которые позволяют — после ответов на ряд вопросов — получить работающий каркас приложения. Вам остается только наполнить его конкретным содержанием. Вызвав диалог New, выберите в нем тип приложения MFC AppWizard (exe), а в поле Project Name — имя создаваемого приложения FrameWork. После нажатия кнопки OK запустится мастер AppWizard, и вы увидите его первое окно (рис. 2.1).

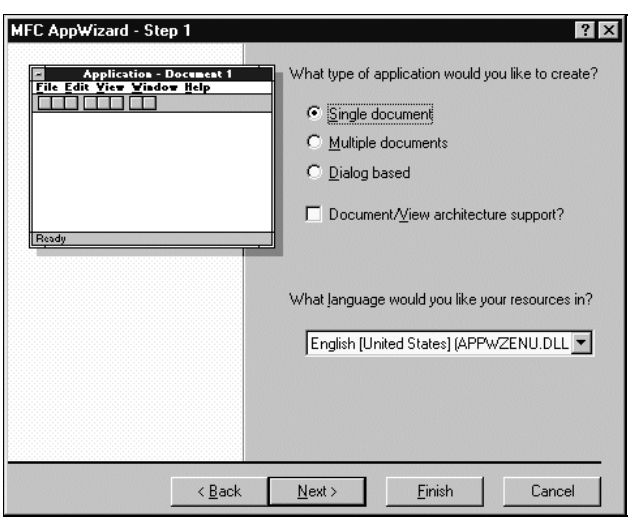


Рис. 2.1. Первое окно мастера AppWizard