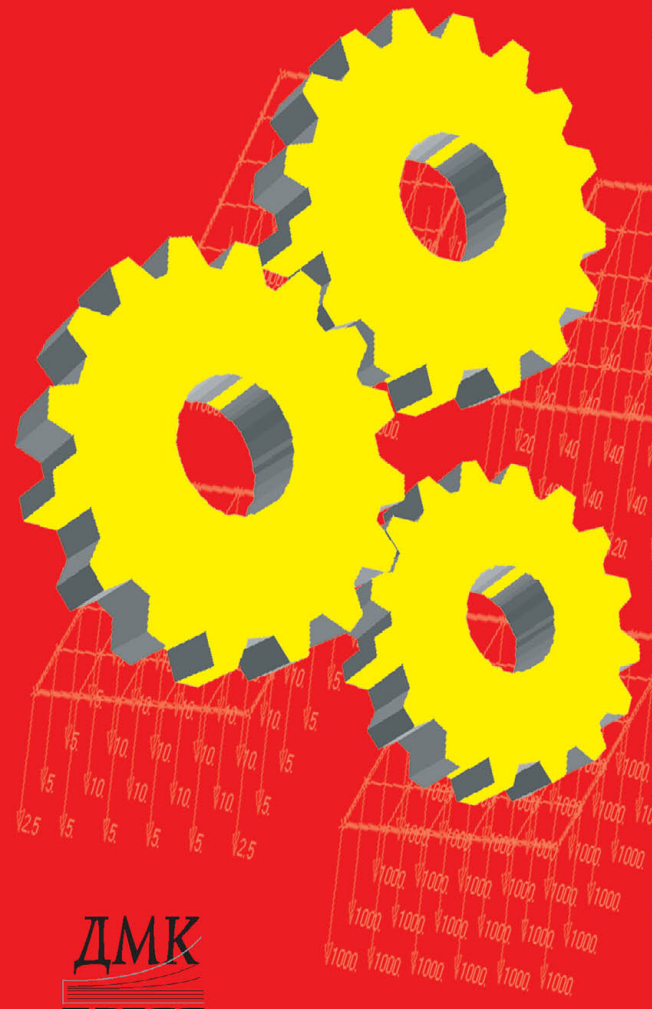
Pocket PC

Руководство разработчика



Создание приложений для мобильных устройств на платформе Microsoft - от Windows CE до Windows Mobile

Методы проектирования платформенно-независимых программ

Оптимизация и отладка приложений для Pocket PC

Написание программ с учетом малого размера экрана Pocket PC

БРЮС Е. КРЕЛЛЬ

ДМК ПРЕСС

УДК 004.4
ББК 32.973.26-018.2
К79

Брюс Е. Крелль
К79 Pocket PC. Руководство разработчика. – М.: Издательский дом ДМК-Пресс, 2010. – 352 с.: ил.

ISBN 5-9706-0031-8

Из этой книги вы узнаете, как можно создавать эффективные программы для КПК (карманных персональных компьютеров) на базе операционных систем Windows CE и Windows Mobile. Вы найдете здесь библиотеки и инструменты, которые помогут заметно сократить время разработки проектов. На примере работающих программ продемонстрирована техника построения графических интерфейсов на маленьком экране Pocket PC.

В издании рассмотрена архитектура Windows CE и Windows Mobile, работа с COM-объектами, проектирование многопоточных приложений и синхронизация, а также оптимизация и отладка программ и их компонентов.

УДК 004.4
ББК 32.973.26-018.2

ISBN 5-9706-0031-8

Содержание

Глава 2. Типичная программа для Pocket PC

В этой главе мы рассмотрим основы реализации программ для Pocket PC. На примере простой программы будут продемонстрированы все требования. Затем мы покажем, как преобразовать программу для настольного компьютера в программу для Windows CE, проанализируем возникающие при этом проблемы. В ходе анализа будут вскрыты изъяны, присущие прямолинейному подходу, и заложены основы каркаса, который будет подробнее рассмотрен в следующей главе.

Рассматриваемую программу реализовать не слишком сложно. По существу, она содержит лишь шаги, необходимые для запуска и завершения любого приложения на платформе Windows CE. В ней мы увидим функции WinMain и WndProc, упомянутые в предыдущей главе. Кроме того, интерфейс будет содержать единственную кнопку для выхода из программы. В самом начале работы программа создаст эту кнопку путем обращения к функции, входящей в состав Windows API. Обработчик нажатия на нее завершит приложение.

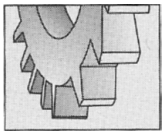
Уникальные особенности программ для Windows CE

При разработке любой программы для Windows CE нужно учитывать требования сверх тех, что были описаны в предыдущей главе. Они проистекают из того факта, что Windows CE – сокращенная версия Windows 2000, и это налагает определенные ограничения.

Точнее, программа должна удовлетворять следующим дополнительным требованиям:

- приложению доступно лишь подмножество функций Win32 API;
- приложению доступно лишь подмножество функций из стандартной библиотеки ANSI C времени исполнения;
- для некоторых операций необходимы совершенно другие функции Win32 API;
- некоторые функции Win32 API ведут себя иначе;
- все строки должны быть представлены в кодировке Unicode, а не ASCII.

Хотя поначалу эти требования могут обескуражить, но по мере работы следование им войдет в привычку.

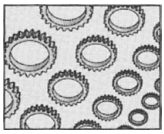


СОВЕТ

Каркас библиотеки, который мы начнем разрабатывать в следующей главе, скрывает большую часть специфических требований и позволяет писать программы, работающие как на настольном ПК, так и на Pocket PC.

В интерфейсе Win32 API для Windows 2000 насчитывается более 4000 функций. Среда Embedded Visual C++ 3.0 предоставляет программе для Pocket PC примерно 2000 функций. Как будет видно из последующих глав, этого вполне достаточно для создания любого приложения для Pocket PC.

Отсутствие некоторых возможностей в Embedded Visual C++ бросается в глаза. Прежде всего это поддержка консольных приложений, безопасности и сервисов, работающих в фоновом режиме. Есть и еще кое-какие нереализованные функции, например SetWindowPos. Но поскольку перемещение окон не входит в графический интерфейс пользователя, поддерживаемый Windows CE, то отсутствие этой функции не должно вызывать удивления; это было бы лишь пустой тратой ценной памяти.



ПРИМЕЧАНИЕ

Microsoft предлагает стандартную конфигурацию средств, поддерживаемых Win32 API на платформе Windows CE. Но производители оборудования, в частности фирма Casio, вольны модифицировать эту конфигурацию. Для этого предназначена программа Platform Builder. В результате состав API для конкретного КПК может отличаться от стандартного.

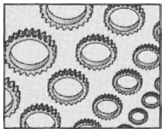
Такому же сокращению подверглась и стандартная библиотека ANSI C, поддерживаемая компилятором Embedded Visual C++. Самое очевидное изъятие – это все функции, объявленные в заголовке stdio.h. Это логично, так как они касаются ввода / вывода на консоль, а этого механизма в Windows CE нет. Отсутствует также ряд функций, обычно объявляемых в заголовке stdlib.h. Из тех, что могли бы оказаться полезны, отметим функции atof (для преобразования текстовых строк в число с плавающей точкой), calloc (для выделения и обнуления памяти) и bsearch (двоичный поиск). Впрочем, при необходимости эти задачи можно решить другими способами или найти переносимый код в сети.

Иногда бывает так, что для реализации задачи приходится применять непривычные функции. Самый очевидный пример – это функции для вывода меню. В версии Visual C++ для настольных ПК для вывода меню в клиентской области служат функции LoadMenu и SetMenu. Тот же эффект в Windows CE достигается с помощью функций CommandBar_Create и CommandBar_InsertMenubar. Различие объясняется тем, что меню является частью полосы команд, в которой могут располагаться и другие элементы управления, например кнопки и списки. Такое использование полосы команд позволяет строить более сложные и удобные интерфейсы. Однако за гибкость приходится платить – в данном случае другим набором функций для создания меню.

Поведение некоторых поддерживаемых функций в Windows CE изменилось. Например, это относится к функциям ReadFile и WriteFile, предназначенным для чтения и записи файла. При выводе строки в файл с помощью функции WriteFile для настольного ПК она записывается в файл без изменения. С помощью любого текстового редактора, например Notepad или WordPad, этот файл можно открыть и увидеть ASCII-символы. Но в Windows CE вы увидите «мусор»; дело в том, что файл содержит строку в кодировке Unicode. Перед записью в файл программа для Pocket PC должна преобразовать строку в ASCII, а затем передать ее функции WriteFile.

Быть может, самое существенное различие между программами для настольного ПК и Windows CE – это необходимость представлять все текстовые строки в кодировке Unicode. Unicode – двухбайтовая кодировка. Кодирование каждого символа двумя байтами значительно расширяет диапазон представимых символов. Для сравнения – в кодировке ASCII каждый символ представляется одним байтом (8 битов), что дает всего 256 различных символов. Хотя для латинского алфавита этого вполне достаточно, но, например, для японского алфавита кандзи, в котором свыше 5000 иероглифов, явно не хватает. В двухбайтовой же кодировке можно представить 65 536 символов.

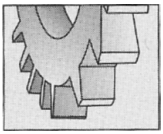
Каждый символ в кодировке Unicode – это кодовая точка (code point). Для разных языков отведены различные диапазоны кодовых точек. Так, первые и последние 256 кодовых точек зарезервированы для ASCII-кодов символов латиницы. Сейчас примерно половина всех кодовых точек еще не распределена по языкам.



ПРИМЕЧАНИЕ

Наличие лишь функций для работы с Unicode-строками – еще одно свидетельство того, что Windows CE – сокращенная версия Windows 2000. Внутри Windows 2000 текст всегда представляется в кодировке Unicode. Если приложение вызывает функцию, принимающую в качестве аргумента ASCII-строку, то Windows 2000 сначала преобразует ее в Unicode, выполняет операцию, а затем преобразует результирующую строку назад в кодировку ASCII.

Для программ, в которых используются символьные строки, в Windows CE предусмотрены два набора функций и макросов. Один из них принимает в качестве аргументов Unicode-строки. Основной тип данных для таких функций – это WCHAR. Здесь буква W означает *wide* (широкий). Широкий символ занимает два байта и точно соответствует кодировке Unicode. Во втором наборе функций и макросов применяются переносимые типы, и базовым является тип *TCHAR*. По определению, TCHAR отображается либо на стандартный тип char для представления ASCII-символов, либо на тип WCHAR – *в зависимости от целевой платформы*. Visual C++ решает, чему соответствует тип TCHAR, ориентируясь на флаги, заданные при компиляции программы. Таким образом, за счет использования переносимого типа мы можем автоматически получить программу, предназначенную для выбранной платформы.



СОВЕТ

Ради обеспечения кросс-платформенного тестирования (о нем речь пойдет в следующей главе) во всех примерах из этой книги используются тип данных TCHAR и соответствующие ему функции и макросы.

Тип TCHAR — основа переносимого механизма обработки строк

К сожалению, для работы со строками переносимым образом нужно приложить некоторые усилия. Ниже перечислены шаги, которые нужно выполнить для перехода от обработки ASCII-строк к работе с типом TCHAR.

1. Включить заголовочный файл <tchar.h>.
2. Объявить все строковые переменные как имеющие тип TCHAR.
3. Для указателей на строки символов использовать тип TCHAR* или LPTSTR.
4. Погрузить строковые литералы в макрос __TEXT().
5. Использовать переносимые функции для работы со строками, например _tcscpy вместо strcpy.
6. При вычислении объема памяти для массивов символов умножать число символов на sizeof(TCHAR).

Автоматическое преобразование кодировки может сказаться на арифметике указателей. Во многих программах для перемещения по буферу применяются арифметические операции над указателями. Впрочем, перекодировка на этапе компиляции на это не влияет. Все вычисления с указателями выполняются корректно с учетом целевой платформы и выбранного представления символов.

Процедуру преобразования можно проиллюстрировать на простом примере. В следующем фрагменте обрабатываются исключительно ASCII-строки:

```
#include <string.h>

char  String1[50];
LPSTR String2;

String2 = (LPSTR) malloc(20 * sizeof(char) );

strcpy( String1, «abcdef» );
strcpy( String2, «xxyyzz» );

free( String2 );
```

После преобразования код принимает такой вид:

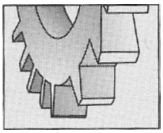
```
#include <tchar.h>           // Шаг 1

TCHAR  String1[50];          // Шаг 2
LPTSTR String2;              // Шаг 3
String2 = (LPTSTR) malloc(20 * sizeof(TCHAR) );    // Шаг 6
```

```
_tcscpy( String1, __TEXT(«abcdef») );       // Шаги 5, 4
_tcscpy( String2, __TEXT(«xxyyzz») );       // Шаги 5, 4

free( String2 );
```

Комментарий в конце каждой строки показывает, какое к ней было применено правило преобразования. Рассмотрим, к примеру, первую из строк, начинающихся с _tcscpy. Правило 5 говорит, что вместо функции strcpy надо использовать _tcscpy, а правило 4 – что строковый литерал «abcdef» нужно окружить макросом __TEXT.



СОВЕТ

Хотя Windows CE поддерживает главным образом Unicode-функции, но есть несколько функций для работы с ASCII-строками. Приложение, которое ими пользуется, может объявить переменную типа char. Например, перед записью в файл можно преобразовать Unicode-строку в кодировку ASCII с помощью функции wcstombs, которая принимает на входе строку в кодировке Unicode, а возвращает ASCII-строку в переданном буфере типа char. Однако такие функции, как strcpy для копирования ASCII-строк, не поддерживаются и не компилируются.

Анатомия простой программы для Windows

В этом разделе мы рассмотрим простую программу для Windows на концептуальном уровне. В частности, будет затронут ряд деталей реализации, которые еще не обсуждались: конкретный графический интерфейс пользователя; структуры данных, необходимые любой программе, работающей под Windows; формат сообщений, которые Windows посылает приложению, и двоичная сигнатура программы. Все это нужно для понимания того, как пишутся и работают программы на платформе Windows CE.

На рис. 2.1 изображен графический интерфейс простой программы, которую мы разработаем в этой главе.

Окно программы содержит полосу заголовка, в которой выводится надпись «HelloWorld Program». В верхнем левом углу клиентской области находится кнопка. При нажатии на нее программа завершается. Многие программы для Windows еще выводят в правом верхнем углу полосы заголовка иконки для управления сворачиванием и разворачиванием окна, но здесь этого нет. Такую возможность Windows CE поддерживает, но в программах из этой книги она не используется. За счет этого удается более строго контролировать интерфейс пользователя. Пользователь может либо работать с программой, либо выйти из нее. Сворачивание и разворачивание хороши для настольных ПК, где можно работать одновременно с несколькими программами. Но в такой среде, как Windows CE, человек обычно занимается одной задачей, для решения которой нужно одно приложение, так что эта функциональность просто излишня.

Дополнительный побочный эффект сворачивания заключается в том, что свернутая программа может оставить в неопределенном состоянии такие ресурсы, как файлы и последовательные порты. При работе с Pocket PC из-за малень-

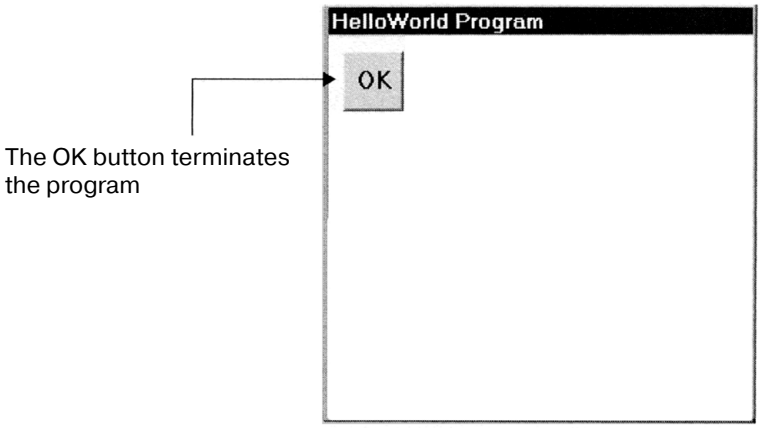
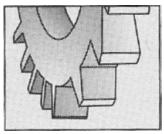


Рис. 2.1. Графический интерфейс простой программы

кого экрана пользователь может свернуть программу, а потом забыть про нее. Но когда программа свернута, она не закрывает файлы и не освобождает ресурсы, а в результате другие программы, нуждающиеся в тех же ресурсах, могут работать неправильно. Если аккумулятор разрядится, то файл так и останется незакрытым, а это может привести к потере данных. Всех этих нежелательных эффектов можно избежать, если приложение не поддерживает сворачивания.

В любой программе для Windows есть ряд важных структур данных. Речь, в частности, идет об определении и создании оконного класса и обработке полей сообщения. На уровне API оконный класс определяется путем заполнения некоторой таблицы.



СОВЕТ

Помещение данных в таблицу заметно сокращает время разработки. В этом случае отладка обычно не нужна. А ведь на отладку логики программы, разыменования указателей и прочих деталей уходят многие часы.

Первая структура, которую мы рассмотрим, имеет тип WNDCLASSEX. На рис. 2.2 представлены ее поля.

Структура WNDCLASSEX определяет общие свойства оконного класса. В верхней части списка показаны два самых важных ее поля. Поле ClassName задает имя класса приложения. С его помощью Windows ссылается на оконный класс приложения. Второе поле – WindowProcedure (оконная процедура) – содержит указатель на функцию, обрабатывающую все сообщения, посылаемые окну конкретного класса. Остальные поля мы обсудим по ходу изложения.

После того как общие характеристики оконного класса определены, программа создает конкретный экземпляр этого класса. При создании экземпляров или оконных объектов заполняется вторая таблица. Атрибуты оконного объекта так-

же показаны на рис. 2.2. Первое поле структуры совпадает с определенным ранее именем класса. Это позволяет Windows связать с конкретным оконным объектом общие характеристики класса. Остальные поля таблицы относятся к конкретному оконному объекту. Ясно, что положение и размер свои для каждого окна.

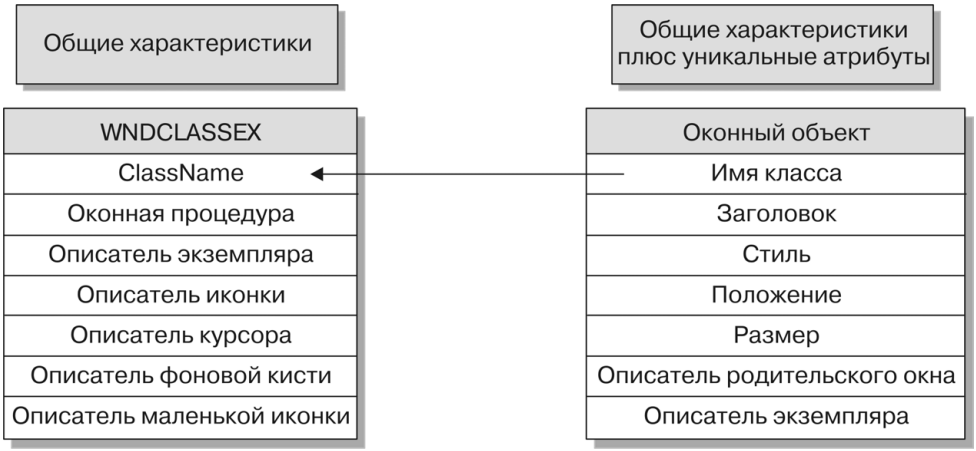


Рис. 2.2. Структуры, описывающие класс, окно и данные приложения для Windows

Оконная процедура занимается обработкой всех поступающих приложению сообщений. Все сообщения имеют один и тот же формат. На рис. 2.3 он показан на примере сообщения WM_COMMAND.

Код сообщения уникален для каждого вида сообщений. В одном из заголовочных файлов Windows перечислены коды всех сообщений, которые Windows может отправить приложению. Эти числовые коды представлены в символьном виде, например WM_COMMAND.

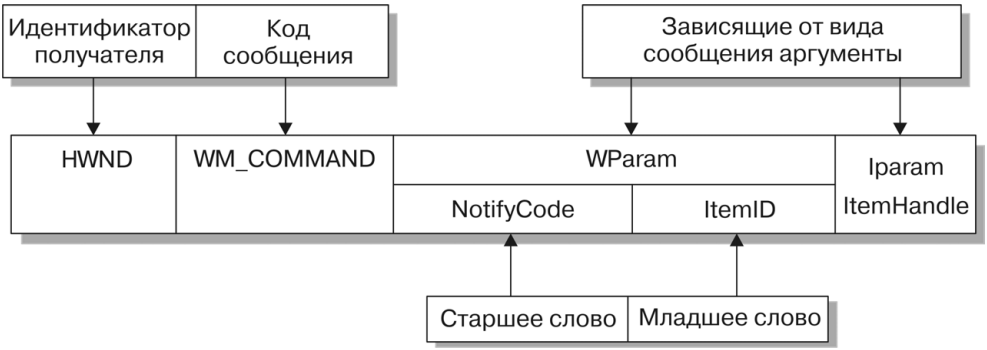
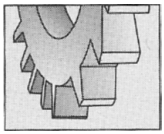


Рис. 2.3. Формат типичного сообщения Windows



СОВЕТ

Каждое адресованное окну сообщение проходит через оконную процедуру приложения. Важные сообщения обрабатываются самой процедурой, а остальные передаются системе для обработки по умолчанию.

Если сообщение не обработано и не передано системе, возможны странные эффекты. Например, некоторая последовательность сообщений заставляет Windows нарисовать полосу заголовка для окна приложения. Если не передать эти сообщения для обработки по умолчанию, то полоса заголовка вообще не появится. Такое окно вряд ли понравится пользователю.

Помимо кода, в каждом сообщении есть два параметра, которые называются wParam и lParam. Эти имена восходят к ранним версиям Windows, когда первый параметр был 16-разрядным числом типа WORD, а второй – 32-разрядным числом типа LONG. Сейчас оба параметра 32-разрядные, но изменять документацию очень трудоемко. Поэтому остались старые имена.

Интерпретация этих параметров зависит от кода сообщения. Ответственность за правильную интерпретацию ложится на оконную процедуру. Выяснить смысл параметров можно несколькими способами. Первый – прочитать документацию. Так, из рис. 2.3 видно, что параметр wParam на самом деле состоит из двух частей. В старшем слове находится код извещения, который показывает, какая операция выполнена над элементом управления. Например, поле ввода посылает извещение EDIT_CHANGE при модификации его содержимого. В младшем слове wParam находится числовой идентификатор элемента управления (скажем, кнопки), который отправил сообщение WM_COMMAND оконной процедуре. Существует несколько макросов для выделения этих частей из параметра wParam, в частности HIWORD и LOWORD. Мы покажем, как применять эти макросы, в примере ниже.

Прежде чем приступать к написанию первой программы для Windows, надо сказать еще об одной вещи. Многие вызовы Win32 API требуют в качестве аргумента *описатель экземпляра*. На рис. 2.4 приведена простая иллюстрация этого понятия.

Приложению необходим какой-то способ идентифицировать объект, представляющий загруженную в память программу. Можно было бы использовать для этой цели полный путь к исполняемому файлу, но это слишком накладно. Ведь всякий раз, как программа захочет сослаться на собственный экземпляр в памяти, пришлось бы передавать одну и ту же строку. Поэтому в Win32 API применяется целочисленный идентификатор, который и называется описателем экземпляра. Его создает Windows и передает программе на стадии инициализации.

На рис. 2.4 представлена и другая важная концепция. Каждая загруженная в память программа Windows состоит из двух частей. Первая – это, конечно, ее двоичный код. А вторая – набор двоичных ресурсов: меню, диалогов, инструментальных панелей, иконок, растровых изображений и строк. Все они находятся в таблице ресурсов. Во время работы приложение может получить доступ к своим ресурсам и использовать их для взаимодействия с пользователями.

Общая логическая структура программ для Windows

Как следует из предыдущей и настоящей глав, функция WinMain в любой программе должна выполнять следующие действия.

1. Зарегистрировать новый оконный класс, заполнив некоторую структуру данных.
2. Создать окно на основе зарегистрированного класса, заполнив другую структуру.
3. Показать окно пользователю.
4. Выбирать сообщения из очереди.
5. Передавать сообщения оконной процедуре для обработки.

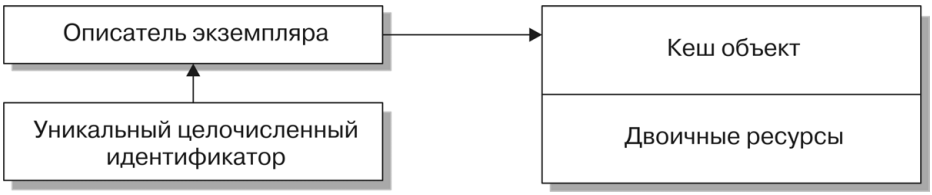


Рис. 2.4. Структура программы для Windows

Именно так устроена функция WinMain в следующем ниже листинге 2.1.

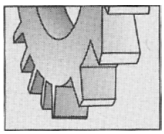
Функция WndProc также организована определенным образом, а именно.

1) имеется предложение switch, осуществляющее ветвление по коду сообщения;
2) для каждого сообщения, представляющего интерес для программы:
   - ❑ из сообщения извлекаются параметры;
   - ❑ выполняется некоторое действие;
3) необработанные сообщения передаются Windows для обработки по умолчанию.

В листинге 2.2 приведена полная реализация функции WndProc согласно описанной схеме.

Типичная программа для Windows

В этой книге принят единый подход: сначала приводится полный исходный текст программы, а затем производится его построчный анализ. Таким образом мы можем показать общую организацию кода и его составных частей, а впоследствии пояснить назначение каждой строки.



СОВЕТ

Ради экономии места мы обычно опускаем объявления переменных стандартных типов, например целых или с плавающей точкой. Но в исходных текстах на сопроводительном сайте (http://www.osborne.com) они присутствуют.

Полный текст функции WinMain

В листинге 2.1 приведен полный текст функции WinMain для программы, исполняемой на настольном ПК.

Листинг 2.1. Полный текст функции WinMain

```
/*********************************************
 *
 * File: WinMain.c
 *
 * copyright, SWA Engineering, Inc., 2001
 * All rights reserved.
 *
 *********************************************/

#include <windows.h>
#include <windowsx.h>

BOOL CALLBACK WinProc(HWND hWnd, UINT message, WPARAM wParam,
                      LPARAM lParam);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                   LPTSTR lpCmdLine, int nCmdShow)
{
   HWND        hwnd;
   HICON       hicon ;
   HCURSOR     hcursor ;
   HBRUSH      hbrush ;
   WNDCLASSEX  wclass ;

   MSG         msg ;

   hicon   = LoadIcon( NULL , IDI_APPLICATION ) ;
   hcursor = LoadCursor( NULL, IDC_ARROW ) ;
   hbrush  = GetStockObject( WHITE_BRUSH ) ;

   wclass.cbSize         =   sizeof(WNDCLASSEX) ;
   wclass.style          =   CS_HREDRAW | CS_VREDRAW  ;
   wclass.lpfnWndProc    =   (WNDPROC)WinProc ;
   wclass.hInstance      =   hInstance ;
   wclass.hIcon          =   hicon ;
   wclass.hCursor        =   hcursor ;
   wclass.hbrBackground  =   hbrush ;
   wclass.lpszMenuName   =   NULL ;
   wclass.lpszClassName  =   "HelloWorld Class" ;
   wclass.hIconSm        =   hicon ;
   wclass.cbClsExtra     =   0 ;
   wclass.cbWndExtra     =   0 ;

   RegisterClassEx( &wclass ) ;

   hwnd = CreateWindowEx( 0 , "HelloWorld Class" , "HelloWorld Program" ,
```

```
        WS_OVERLAPPED ,
        0, 0 ,
        288,375,
        NULL , NULL , hInstance , NULL ) ;

   ShowWindow( hwnd , nCmdShow ) ;
   UpdateWindow( hwnd ) ;

   while ( GetMessage( &msg, NULL , 0 , 0 ))
   {
      TranslateMessage( &msg ) ;
      DispatchMessage( &msg ) ;
   }

   return msg.wParam ;
}
```

Эта программа следует общей схеме, типичной для всех Windows-программ. Сначала идут определение, создание и отображение главного окна. Затем начинается цикл выборки сообщений с передачей их оконной процедуре для обработки.

Построчный анализ функции WinMain

Теперь можно заняться подробным анализом. Мы не будем останавливаться на таких очевидных элементах, как объявление переменных часто используемых типов или фигурные скобки, отмечающие начало и конец блока.

```
#include <windows.h>
#include <windowsx.h>
```

Эти два файла включают множество других заголовочных файлов, в которых содержатся объявления типов данных и функций. Так, windows.h включает, в частности, файл winuser.h. Его имя говорит о том, что в нем содержатся объявления, относящиеся к компоненту USER.

```
BOOL CALLBACK WinProc(HWND hWnd, UINT message, WPARAM wParam,
      LPARAM lParam);
```

В этой строке объявляется сигнатура оконной процедуры, которая должна быть в каждом приложении. Сигнатура включает тип возвращаемого значения, имя функции и список аргументов. Кроме того, в нем есть часть, характерная именно для программирования в Windows. Это слово CALLBACK, которое часто называют *уточнением* (adornment). Уточнения несут информацию, необходимую компилятору для правильной генерации кода. В данном случае слово CALLBACK определяет порядок помещения в стек аргументов, передаваемых функции.

Сигнатура функции WinProc допускает некоторую гибкость. Имена самой функции и ее аргументов могут отличаться от указанных в сигнатуре. Но тип возвращаемого значения, набор уточнений и типы аргументов должны в точности совпадать с объявленными. Это необходимо для того, чтобы размеры и порядок аргументов в стеке соответствовали тем, чего ожидает Windows. В противном случае могут произойти неприятности. В самом деле, Windows поместит в стек пра-

вильные значения, а оконная процедура начнет интерпретировать их совсем по-другому. В результате сообщения будут обрабатываться некорректно, и поведение программы станет непредсказуемым.

```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
      LPTSTR lpCmdLine, int nCmdShow)
```

Этот заголовок функции точно соответствует сигнатуре WinMain, требуемой Windows. Здесь мы встречаем еще одно уточнение – WINAPI. Его смысл точно такой же, как у описанного выше уточнения CALLBACK. Первый аргумент функции, hInstance, – это уникальный числовой идентификатор загруженной в память программы. Во втором аргументе, hPrevInstance, Windows всегда передает 0. Он присутствует только для совместимости с предыдущими версиями, и для вновь разрабатываемых программ совершенно бесполезен. Однако если его опустить, то многие существующие программы перестали бы работать. Следующий аргумент lpCmdLine содержит указатель на строку, содержащую параметры, переданные программе при запуске. Для большинства программ эта строка пуста. И последний аргумент nCmdShow определяет начальное состояние окна. Почти всегда он равен SW_SHOW, то есть окно должно быть видимо.

В этом заголовке меняться могут только имена аргументов. Если вы измените что-то еще, программа просто не откомпилируется или не будет работать. Совместно Windows и Visual C++ определяют WinMain как точку входа в программу. Если функции с таким именем и сигнатурой не будет обнаружено, то Visual C++ недвусмысленно заявит об этом.

```
HWND        hwnd;
HICON       hicon ;
HCURSOR     hcursor ;
HBRUSH      hbrush ;
WNDCLASSEX  wclass ;
MSG         msg ;
```

Эти переменные служат для хранения различной информации, используемой в разных частях программы. Типы данных специфичны для Windows. Например, тип HWND говорит, что в переменной хранится описатель окна. В терминологии, принятой в Windows, описателем называется уникальное целое число, используемое для доступа к внутренней структуре данных. Если приложение желает как-то манипулировать этой структурой, то должно передать соответствующей функции ее описатель. Напрямую программа никогда не обращается к внутренним данным, только через описатель и функцию API. Пресекая все попытки несанкционированного изменения внутренних данных, Windows может гарантировать целостность операционной системы.

Интерес представляют еще некоторые типы данных, встречающиеся в объявлениях. Структура WNDCLASSEX необходима для определения оконного класса приложения. В переменную типа MSG помещаются сообщения, извлекаемые из очереди основного потока.

```
hicon   = LoadIcon( NULL , IDI_APPLICATION ) ;
hcursor = LoadCursor( NULL, IDC_ARROW ) ;
```