WAITE GROUP

Microsoft Windows API大全

JAMES L. CONGER 著
沈 民 张 燕 译

北京科海培训中心

Wait Group

Microsoft Windows API 大全

JAMES L. CONGER 著
沈民　张燕　等译

北京科海培训中心

简　介

本书作者通过在实际工作中积累了大量的 Windows 程序开发经验，根据 Windows 的特点系统地讲解了 Windows 函数的功能与使用技巧，并配以程序实例分门别类地加以分析。除此以外，本书还由浅入深地介绍了 Windows 特有的消息驱动程序的特点与编写方法及其他有关技术。

Windows 系统正处在高速发展与更新阶段，Windows 版本更新迅速，令读者目不暇接，难以应付。本书作者力图突破时间约束，在内容与结构上去粗求精，从本质上分析 Windows 文化，极力为读者建立一个举一反三的知识基础。

本书是一本经典的 Windows API 大全，是一本难得的 Windows 力作，无论对 Windows 的初学者还是高级程序员都是一本极好的工具书。

目　录

第一章　Windows 编程总览 …… 1
1.1　Windows 编程概述 …… 1
1.2　Windows 程序的结构 …… 1
1.3　Windows 程序 GENERIC 范例 …… 2
1.4　Windows 程序是如何编译和连接的 …… 7
1.5　Windows 程序是如何工作的 …… 7
1.6　Windows 的命名规则——WINDOWS.H …… 8
1.7　GENERIC 的开发 …… 10
1.8　实例和消息循环 …… 11
1.9　本书中的程序范例的约定 …… 12

第二章　创建窗口 …… 14
2.1　已存在的类与 CreateWindow( )函数的使用 …… 14
2.2　用不同消息处理程序创建新窗口类 …… 16
2.3　CreateWindow( )函数产生的消息 …… 19
2.4　控制窗口的其它功能 …… 20
2.5　函数的描述 …… 21

第三章　Windows 系统支持函数 …… 33
3.1　窗口属性的直接改变 …… 33
3.2　改变类数据 …… 33
3.3　窗口及类的数据 …… 34
3.4　注意：列举函数 …… 35
3.5　注意事项 …… 37
3.6　函数描述 …… 38

第四章　菜单 …… 116
4.1　主菜单及弹出菜单 …… 116
4.2　在资源文件里建立菜单 …… 116
4.3　在程序的窗口中增加菜单 …… 118
4.4　改变菜单 …… 118
4.5　位图作为菜单项 …… 119
4.6　检取标志位图 …… 119
4.7　自画菜单项（Owner-Drawn） …… 120
4.8　菜单消息 …… 122

4.9 菜单函数综述 …… 123

第五章 滚动杆 …… 157
5.1 滚动杆概念 …… 157
5.2 滚动杆的位置和范围 …… 158
5.3 滚动杆消息 …… 158
5.4 滚动杆函数一览表 …… 159

第六章 鼠标和光标函数 …… 173
6.1 鼠标消息概述 …… 173
6.2 通常的鼠标消息 …… 174
6.3 鼠标函数 …… 175
6.4 插字符函数 …… 176
6.5 鼠标、光标函数一览表 …… 176

第七章 键盘支持 …… 203
7.1 虚键 …… 203
7.2 键盘消息 …… 205
7.3 非英语键盘消息 …… 207
7.4 键盘加速器 …… 207
7.5 键盘函数概述 …… 210

第八章 消息处理函数 …… 229
8.1 消息流程 …… 229
8.1.1 处理消息 …… 230
8.1.2 程序控制 …… 230
8.1.3 消息来源 …… 231
8.1.4 函数重入 …… 231
8.2 消息钩子函数 …… 232
8.3 注意事项 …… 232
8.4 消息函数一览表 …… 232

第九章 Windows 消息 …… 274
9.1 传递消息 …… 274
9.2 传递按钮消息概述 …… 275
9.3 按钮通知码 …… 278
9.4 按钮通知码概述 …… 279
9.5 按钮通知码描述 …… 280
9.6 组合框消息 …… 282

9.7 自画组合框 …… 283
9.8 组合框消息总结 …… 285
9.9 组合框消息描述 …… 286
9.10 组合框通知码总结 …… 295
9.11 组合框通知码描述 …… 295
9.12 对话框窗口消息 …… 298
9.13 编辑控制消息 …… 299
9.14 编辑控制消息总结 …… 300
9.15 编辑控制描述 …… 302
9.16 编辑控制通知消息 …… 310
9.17 编辑控制通知消息描述 …… 311
9.18 列表框消息 …… 313
9.19 列表框消息描述 …… 316
9.20 列表框通知码 …… 327
9.21 列表框通知码描述 …… 328
9.22 静态控制消息 …… 329
9.23 窗口消息 …… 329
9.24 窗口消息描述 …… 334

第十章 设备描述表、文字输出和打印 …… 380
10.1 设备描述表 …… 380
10.2 处理 WM_PAINT 消息 …… 381
10.3 对象选入设备描述表 …… 381
10.4 私有设备描述表 …… 382
10.5 保存设备描述表 …… 382
10.5.1 映射方式 …… 383
10.5.2 字体 …… 384
10.6 打印机支持 …… 385
10.7 打印设备驱动器 …… 388
10.8 文字和设备描述表函数一览表 …… 389
10.9 文字和设备描述表函数描述 …… 391

第十一章 屏幕绘图 …… 471
11.1 WM_PAINT 消息 …… 471
11.2 无效矩形 …… 472
11.3 设备描述表 …… 472
11.4 为设备描述表选择对象 …… 473
11.5 缺省和库存对象 …… 474
11.6 颜色 …… 475

11.7 区域 …… 476
11.8 绘图函数一览表 …… 476
11.9 绘图函数描述 …… 479

第十二章 彩色调色板控制 …… 582
12.1 硬件调色板 …… 582
12.2 Windows 中的彩色调色板 …… 583
12.3 逻辑调色板 …… 583
12.4 创建一个逻辑调色板 …… 584
12.5 Windows 的彩色调色板消息 …… 584
12.6 调色板函数概述 …… 585
12.7 调色板函数的描述 …… 585

第十三章 对话框 …… 608
13.1 对话框的例子 …… 608
13.2 对话框类型 …… 611
13.3 间接和变量对话框函数 …… 612
13.4 与对话框控制进行交流 …… 612
13.5 对话框键盘界面 …… 614
13.6 动态对话框 …… 615
13.7 对话框模板语句描述 …… 616
13.8 对话框控制语句 …… 618
13.9 对话框函数一览表 …… 623
13.10 对话框函数描述 …… 623

第十四章 内存管理 …… 672
14.1 局部和全局内存 …… 672
14.2 段和偏移量 …… 673
14.3 局部堆中的内存分配 …… 673
14.4 全局堆中的内存分配 …… 675
14.5 可移动、固定和可抛弃内存块 …… 675
14.6 要避免的陷井 …… 676
14.7 Windows 内存结构 …… 677
14.8 可移动程序代码段 …… 677
14.9 编译器内存模式 …… 678
14.10 锁定、固定和分页锁定内存块 …… 679
14.11 运行其他模块 …… 679
14.12 模块定义语句 …… 680
14.13 模块定义语句描述 …… 680

14.14 内存函数概述 …… 684
14.15 内存函数描述 …… 686

第十五章 位图 …… 735
15.1 DDB 位图格式 …… 735
15.2 使用 DDB 位图 …… 736
15.3 内存设备描述表 …… 736
15.4 拉伸和绘制位图图像 …… 737
15.5 老位图格式存在的问题 …… 738
15.6 设备无关的位图（DIB） …… 738
15.6.1 如何运用 DIB …… 740
15.7 DIB 范例 …… 741
15.8 位图函数一览表 …… 742
15.9 位图函数描述 …… 743

第十六章 图标 …… 781
16.1 图标的使用 …… 781
16.2 运行中创建图标 …… 782
16.3 图标函数一览表 …… 783
16.4 图标函数的描述 …… 783

第十七章 裁剪板（Clipboard） …… 791
17.1 裁剪板的使用 …… 791
17.2 裁剪板的格式 …… 792
17.3 复合（Multiple）裁剪板格式 …… 793
17.4 裁剪板数据的延迟补偿 …… 794
17.5 位图和元文件裁剪板格式 …… 794
17.6 裁剪板观察器程序 …… 795
17.7 裁剪板函数摘要 …… 796
17.8 裁剪板函数的描述 …… 796

第十八章 声音函数 …… 817
18.1 声音资源 …… 817
18.2 声音驱动器 …… 817
18.3 声音和音符队列 …… 818
18.4 声音阈值 …… 819
18.5 声音函数错误代码 …… 819
18.6 声音函数总结 …… 820
18.7 声音函数描述 …… 820

第十九章　字符集和字符串 …… 839
19.1　字符集 …… 839
19.2　字符集转换 …… 840
19.3　字体和字符集 …… 840
19.4　字符串函数 …… 841
19.5　字符集和字符串函数总结 …… 841
19.6　字符集和字符串函数描述 …… 842

第二十章　MS-DOS 与访问磁盘文件 …… 859
20.1　磁盘文件 …… 859
20.2　文件名表 …… 860
20.3　初始化文件 …… 861
20.4　MS-DOS 和磁盘文件函数总结 …… 862
20.5　MS-DOS 和磁盘文件函数的描述 …… 862

第二十一章　通信函数 …… 897
21.1　通信支持 …… 897
21.2　从接收数据队中读取数据 …… 898
21.3　通信函数一览表 …… 898
21.4　通信函数描述 …… 899

第二十二章　原子函数 …… 920
22.1　原子表 …… 920
22.2　原子数据结构 …… 920
22.3　数据交换 …… 921
22.4　原子函数一览表 …… 921
22.5　原子函数描述 …… 922

第二十三章　元文件 …… 932
23.1　创建并引用内存元文件 …… 932
23.2　创建并显示磁盘元文件 …… 933
23.3　元文件磁盘格式 …… 934
23.4　修改元文件数据 …… 934
23.5　元文件的一些限制 …… 935
23.6　元文件函数一览表 …… 936
23.7　元文件函数描述 …… 936

第二十四章　定时器 …… 947
24.1　定时器的使用方法 …… 947
24.2　定时器精度 …… 948
24.3　其它时间函数 …… 948
24.4　定时器函数总结 …… 948
24.5　函数描述 …… 948

第二十五章　资源 …… 954
25.1　资源编译器 …… 954
25.2　资源描述文件 …… 955
25.3　串表 …… 956
25.4　自定义资源 …… 957
25.5　资源函数概要 …… 958
25.6　资源函数描述 …… 958

第二十六章　程序的剖析和调试 …… 975
26.1　剖析器工作原理 …… 975
26.2　启动剖析器前的准备工作 …… 975
26.3　使用剖析器注意事项 …… 977
26.4　调试函数 …… 977
26.5　剖析函数及调试函数概述 …… 978
26.6　剖析函数和调试函数描述 …… 978

第二十七章　Help 文件支持 …… 988
27.1　建立 Help 文件 …… 988
27.2　HELP 文本特殊字符 …… 989
27.3　定义超级文本跳转和索引入口 …… 990
27.4　添加检索字符串和书签 …… 990
27.5　添加位图图形 …… 991
27.6　编译 HELP 文件 …… 991
27.7　HELP 项目文件选项 …… 992
27.8　项目文件的[Files]段 …… 992
27.9　项目文件的[BuildTags]段 …… 992
27.10　项目文件的[Options]段 …… 993
27.11　项目文件的[Alias]段 …… 994
27.12　项目文件的[Map]段 …… 994
27.13　项目文件的[Bitmaps]段 …… 995
27.14　使用 HELP 系统 …… 995

第二十八章　动态链接库 …… 999
28.1　什么是 DLL? …… 999
28.2　创建 DLL …… 1000
28.3　使用 DLL 中的函数 …… 1003
25.4　调用 DLL 函数的其它方法 …… 1004
28.5　调用 Windows 库函数 …… 1005
28.6　编译 DLLs 的问题 …… 1006
28.7　调试 DLLs …… 1007
28.8　动态链接库函数总结 …… 1007
28.9　动态链接库函数描述 …… 1008

第二十九章　多文本界面 …… 1015
29.1　MDI 框架窗口和子窗口 …… 1015
29.2　MDI 应用程序的结构 …… 1016
29.3　MDI 接口存在的问题 …… 1017
29.4　MDI 范例程序 …… 1017
29.5　MDI 函数概述 …… 1025
29.6　MDI 函数介绍 …… 1025

第三十章　动态数据交换 …… 1029
30.1　如何交换 DDE 数据 …… 1029
30.2　应用程序、主题和项标识符 …… 1030
30.3　冷 DDE 链 …… 1030
30.4　热 DDE 链 …… 1031
30.5　暖 DDE 链 …… 1032
30.6　通用 DDE 对话 …… 1033
30.7　数据传送的其它 DDE 消息 …… 1034
30.8　添加一新的程序组到程序管理器(ProgramManager) …… 1035
30.9　从 Microsoft Excel 获得文件名 …… 1039
30.10　DDE 消息概述 …… 1042
30.11　DDE 消息描述 …… 1043

附录 A　其他技术参考书 …… 1052

附录 B　WINDOWS.H 中的宏指令 …… 1054

第一章　Windows 编程总览

本章主要介绍 Windows 的编程以及 GENERIC.C 程序的开发。GENERIC.C 程序将做为本书中所有例子的基础。

1.1　Windows 编程概述

习惯于在 DOS 和微机环境下编写程序的程序员，第一次看到 Windows 程序时，可能会有些不知所措。因为 Windows 程序与传统程序不同，其不同点可归结如下：

1. Windows 程序不是每次告诉计算机一步一步做什么，而是一直在那里等待消息的输入。这些消息是这样一些语句，例如："用户按了鼠标器的某个键，程序要完成某些任务!"。

2. Windows 的环境为诸如屏幕显示、存储器、鼠标器、键盘、打印机等所有基本硬件提供驱动程序。Windows 把所有最新硬件技术都考虑在内，使编程人员可以一心一意地开发应用程序。这样，程序员只需花费时间学习和使用 600 个 Windows 函数，而不必自己编写程序去支持各种各样的打印机、显示卡等等。

3. Windows 可以在内存中移动程序和数据，以便为其它程序和数据提供空间。这种移动可使有限的内存用于更多的程序，但同时也意味着程序员不能认为任何东西都可以长久保留。Windows 提供了所需要的一切工具处理可移动的内存，但这需要一些时间去适应这种变化。

除了这些区别之外，Windows 环境不难使用。当渡过了最初编写简单程序的障碍之后，Windows 提供的强大内部功能会使人感到头痛。不仅如此，如果要使用更底层的系统功能，那么会有更大的困难。

对于使用 Windows 环境的新手，最重要的经验是"来试试"。经过一段时间就会觉得大部分 Windows 程序是十分简单的。学会其中一个，第二个程序只需要稍加改动就行了。这本书的主要目的之一就是给 Windows 函数提供一些工作实例，说明如何使用每一个函数，从而节省时间。为了提高效率，下一部分中介绍的简单的"GENERIC"程序是大部分程序范例的基础。

1.2　Windows 程序的结构

大部分 Windows 程序一般有两个 C 语言函数，WinMain( )和 WndProc( )。虽然 WndProc( )几乎在每个 Windows 程序中都出现，但只有 WinMain( )是必需的。可以任意给 WndProc( )取别的名字，但大部分编程人员都把它命名为 WndProc( )。与普通 C 语言程序中的 main( )一样，WinMain( )必须命名为"WinMain"。任何大的程序都会有很多其它函数为 WndProc( )工作，但上述两个函数是基本函数。

WinMain( )：调用一些函数，告诉 Windows 系统关于程序主窗口的属性。这包

括窗口上的颜色、程序初始化时的图符名称，以及如何找到程序菜单等。WinMain( )也包括一些标准代码，完成 Windows 与程序之间的消息交换。如 main( )在一般 C 语言程序中一样，WinMain( )也是 Windows 程序的出口和入口。

WndProc( ): 普通程序，由程序员编写。通常称作"消息处理函数"，解释消息并完成相应动作。

请看一个简单范例。该程序创建一个窗口，标题为"generic"，带两个菜单："Do it!"和"Quit"。如图 1-1 所示。当程序初次运行时，窗口显示出来。

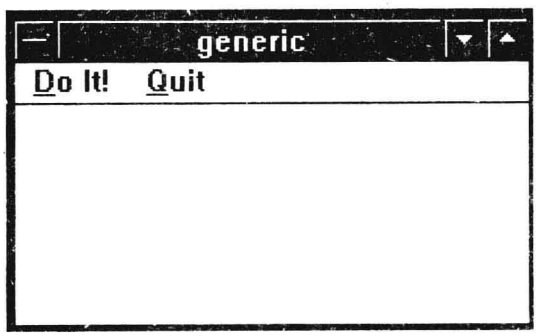

图 1-1 GENERIC 程序的窗口

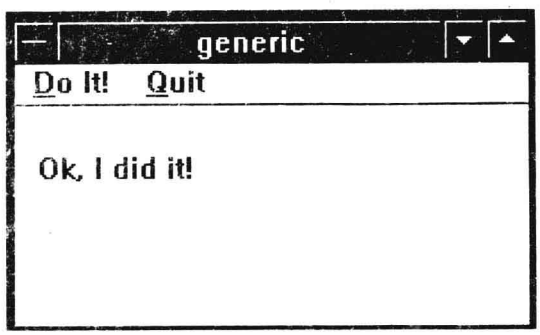

图 1-2 点取菜单项"Do it!"之后的 GENERIC

移动鼠标到"Do it!"的位置，然后按鼠标器左边键，就会象图 (1-2) 显示的那样在窗口内出现："OK，I did it!"的文字提示。如果点取"Quit"位置，就会使程序停止，窗口消失。右上角的最大化、最小化控制和左上角的系统控制是 Windows 的标准控制形式。

1.3 Windows 程序 GENERIC 范例

程序 1-1 列出了 GENERIC.C 所需要所有的 C 语言代码。虽然一眼看上去，这些代码很复杂，但它是很短的。记住，这个程序产生了一个窗口，这个窗口可以在屏幕上移动位置、变换大小，最小化到一个图符、最大化到整个屏幕，这个窗口中有一个命令菜单。

程序 1-1 GENERIC.C

Listing 1-1. GENERIC.C

```
/* generic.c   generic windows application */
#include <windows.h>                /* window's header file - always included */
#include "generic.h"                /* the application's header file */

int PASCAL WinMain (HANDLE hInstance, HANDLE hPrevInstance, LPSTR lpszCmdLine, int nCmdShow)
{                                   /* variable types defined in windows.h */

        HWND            hWnd ;      /* a handle to a message */
        MSG             msg ;       /* a message */
        WNDCLASS        wndclass ;  /* the window class */

        ghInstance = hInstance ; /* store instance handle as global var. */

        if (!hPrevInstance)         /* load data into window class struct. */
        {
```

```
        wndclass.style                  = CS_HREDRAW | CS_VREDRAW ;
        wndclass.lpfnWndProc            = WndProc ;
        wndclass.cbClsExtra             = 0 ;
        wndclass.cbWndExtra             = 0 ;
        wndclass.hInstance              = hInstance ;
        wndclass.hIcon                  = LoadIcon (hInstance, gszAppName) ;
        wndclass.hCursor                = LoadCursor (NULL, IDC_ARROW) ;
        wndclass.hbrBackground          = GetStockObject (WHITE_BRUSH) ;
        wndclass.lpszMenuName           = gszAppName ;
        wndclass.lpszClassName          = gszAppName ;
                        /* register the window class */
        if (!RegisterClass (&wndclass))
                return FALSE ;
    }
    hWnd = CreateWindow (               /* create the program's window here */
        gszAppName,                     /* class name */
        gszAppName,                     /* window name */
        WS_OVERLAPPEDWINDOW,            /* window style */
        CW_USEDEFAULT,                  /* x position on screen */
        CW_USEDEFAULT,                  /* y position on screen */
        CW_USEDEFAULT,                  /* width of window */
        CW_USEDEFAULT,                  /* height of window */
        NULL,                           /* parent window handle (null = none) */
        NULL,                           /* menu handle (null = use class menu) */
        hInstance,                      /* instance handle */
        NULL) ;                         /* lpstr (null = not used) */

    ShowWindow (hWnd, nCmdShow) ;               /* make window visible */
    UpdateWindow (hWnd) ;                       /* send first WM_PAINT message */
                        /* the next while() loop is the "message loop" */

    while (GetMessage (&msg, NULL, 0, 0))       /* wait for a message */
    {
        TranslateMessage (&msg) ;               /* does some key conversions */
        DispatchMessage (&msg) ;                /* sends message to WndProc() */
    }
    return msg.wParam ;                         /* returns application's exit code */
}

long FAR PASCAL WndProc (HWND hWnd, unsigned iMessage, WORD wParam, LONG lParam)
{
    HDC     hDC ;                               /* device context handle */

    switch (iMessage)                           /* process windows messages */
    {
        case WM_COMMAND:                        /* process menu items */
            switch (wParam)
            {
            case IDM_DOIT:                      /* User hit the "Do it" menu item */
                hDC = GetDC (hWnd) ;            /* get device context */
                TextOut (hDC, 10, 20, "Ok, I did it!", 13) ;
                ReleaseDC (hWnd, hDC) ;  /* release device context */
                break ;
            case IDM_QUIT:                      /* send end of application message */
                DestroyWindow (hWnd) ;
                break ;
            }
            break ;
        case WM_DESTROY:                        /* stop application */
            PostQuitMessage (0) ;
            break ;
        default:                                /* default windows message processing */
            return DefWindowProc (hWnd, iMessage, wParam, lParam) ;
    }
    return (0L) ;
}
```

WinMain( )函数看上去比它实际复杂。幸运的是这个函数在不同的程序中几乎保持不变。在编写新程序时，只需要把它照抄就是了。等一会儿，会解释这段程序是做什么的。让我们先来看一看 WndProc( )函数。

WndProc( )函数处理 Windows 消息。这些消息都是整型数，但为了清楚，在

Windows 的头文件 WINDOWS.H 中定义了名称。GENERIC.C 需要处理的两条消息是 WM__COMMAND（执行菜单）和 WM__DESTROY（停止执行、关闭窗口）。两个菜单项以 IDM__DOIT 和 IDM__QUIT 为标号。这些名字在程序 GENERIC.H 文件中定义。程序 1-2 列出它们。

程序 1-2　GENERIC.H 文件

```
/ * generic.h
#define IDM__DOIT      1                / * menu item values * /
#define IDM__OUIT      2
          / * global variables * /
int       ghInstance;
char      gszAppName [ ] = "generic" ;
          / * function prototypes * /
long FAR PASCAL WndProc (HWND,unsigned,WORD,LONG);
```

当用户点取“Do it!”菜单项后，Windows 发给 GENERIC 程序一条 WM__COMMAND 消息。这个消息的一部分是菜单项的编号。在本例情况下，IDM__DOIT 在 GENERIC.H 中定义为 1。当 GENERIC.C 中的 WinProc( )函数接到这个消息后，它就执行以下代码：

```
case IDM  DOIT: * /
          hDC = GetDC (hWnd) ;              / * get device context * /
          TextOut (hDC,10,20,"ok,I did it!",13);
          ReleaseDC (hWnd,hDC);             / * release device context * /
          break;
```

这段程序通过使用 Windows 中的 GetDC( )函数获得屏幕显示信息。根据这个信息（称作设备描述表），程序会利用 Windows 中的 TextOnt( )函数在屏幕上显示出“OK，I did it!”（我已经做好了!）。最后通过使用 Release( )函数释放与屏幕信息相关的内存。这三个函数使文字呈现在屏幕上。

同样地，如果用户点取“Quit”菜单项，程序调用 DestroyWindow( )函数。DestroyWindow( )函数清除程序的窗口，程序终止。这叫做“终止”应用程序。Windows 向程序发出 WM__DESTROY 消息，这一消息处理后程序退出。

我们已经讲完了 WndProc( )函数的运行方法。那么 GENERIC.C 程序中的 WinMain( )函数又是怎样的呢？这段代码的大部分是用来创建一个程序的主窗口的。创建一个窗口有三个步骤：

1. 首先，必须登记一个窗口类。类用一组填入一个称作 Wndclass 的结构数据描述。这里有一个这样的例子：

```
wndclass.hIcon = LoadIcon (hInstance,gszAppName);
```

在本例中，用上述类创建的每一个窗口将有一个和程序“generic”名字一样的图符。GENERIC.H 头文件定义了全局变量 gszAppName。填完全部数据后，就可以用 RegisterClass( )函数告知 Windows 已经创立了一个新的窗口类。

2. 其次，利用 CreateWindow( )函数根据窗口类生成一个或多个窗口。

CreateWindow( )函数向窗口输送另外的信息，比如，窗口的风格，背景颜色等

等。

3. 最后，利用 ShowWindow( )函数显示出窗口。在 WinMain( )函数的最下面，可以看到一个很奇怪的循环：

```
while (GetMessage (&msg,NULL,0,0))                    / * the message loop * /{
        TranslateMessage (&msg);
        DispatchMessage (&msg);
}
```

这个称作“消息循环”的循环存在于每一个 Windows 程序中。Windows 通过这个循环中的函数向程序发送所有消息。还有一小部分函数可以在这个消息循环中使用，它们用于特殊目的，比如，菜单加速键装载函数，但通常该循环和本例一样。

如果输入 GENERIC.C 程序，并且想编译、运行它，就需要一些其它小文件。这些文件是一些资源文件，定义了菜单，图标和程序所用的其它资源；定义文件在生成程序时给予编译器指示；“make”文件帮助自动编译和连接程序。

GENERIC.RC 资源文件比较简单，它包括一个图标文件 GENERIC.ICO，这个文件是由 Windows Software Development Kit 中的 SDKPaint 应用程序生成的。资源文件中还定义了菜单。注意，菜单项在头文件中定义了 ID 号。

程序 1-3　GENERIC.RC 资源文件

```
#include <windows.h>
#include "generic.h"

generic             ICON    generic.ico

generic             MENU
BEGIN
  MENUITEM "&Do It"          IDM_DOIT
  MENUITEM "&Quit",          IDM_QUIT
END
```

以.DEF 为后缀的文件向连接器提供如何汇编最终程序的信息。第十四章“内存管理”中对所有可以在“.DEF”文件中使用的语句进行了充分讨论。这里只对这个例子文件作扼要描述。

DESCRIPTION 字符串在文件中通常用来保存版权信息。WINDOWS 的 EXETYPE 告诉连接器这将是“Windows 3.0 版本程序”。STUB 语句命名一个小文件，这个文件将放在最终程序的开头。WINSTUB 文件是一些代码，如果用户想从 DOS 中执行 Windows 程序，它就会显示出警告。

CODE 和 DATA 语句决定了系统如何为这个程序去管理内存。程序 1-4 说明了通常的配置。HEAPSIZE 和 STACKSIZE 决定了为程序所拥有的堆和栈数据所安排的内存容量。最后，EXPORTS 部分列出了程序中需要 Windows 调用的全部函数。(不包括命令 WinMain( )在内)。

程序 1-4　GENERIC.DEF 定义文件

```
NAME            GENERIC
DESCRIPTION     'generic windows program'
EXETYPE         WINDOWS
```

```
STUB            'WINSTUB.EXE'
CODE            PRELOAD MOVEABLE
DATA            PRELOAD MOVEABLE MULTIPLE
HEAPSIZE        1024
STACKSIZE       5120
EXPORTS         WndProc
```

NMAKE.EXE 是一个可执行另一个程序的程序，特别是调用编译器和连接器。NMAKE 依据 NMAKE 控制文件("make"文件) 自动编译程序。按常规，NMAKE 文件名字和 C 语言主程序一样，但没有扩展名。例如，GENERIC.C 的 NMAKE 文件就是 GENERIC，例 1-5 说明了这点。GENERIC 程序从 ALL 语句开始。它告诉 NMAKE 我们正在试图生成 GENERIC.EXE 文件，任何比 GENERIC.EXE 更新的有关文件都将在下次编译时编入.EXE 文件。

下面两行程序定义了宏指令。每次"$ (CFLAGS)"出现时，编译器就会代之以字符串"-c -D LINE_ARGS -A -Os -Gsw -W2"。这些是编译一个 Windows C 语言程序的标准编译器开关。同样地，LFLAGS 会被"/NOD"连接器控制开关所代替。这个标志将会在第十四章"内存管理"中讨论。

剩下几行语句告诉 NMAKE 程序哪些文件要比较，从而决定一个文件是否需要再编译。例如，如果 GENERIC.C 和 GENERIC.H 中的任何一个文件是在 GENERIC.OBJ 文件之后存入，底下一行语句就会执行。资源编译器 RC.EXE 是由下一组指令控制。最后一组指令控制连接器。注意，RC.EXE 会在 NMAKE 文件最后再次运行。资源编译器把编译过的资源数据 (从上面的资源文件中得到的) 加入程序文件中，然后对已编译成的程序标以 Windows3.0 应用软件。

程序 1-5　GENERIC_NMAKE 文件

```
All: generic.exe

CFLAGS=-c -D LINT_ARGS -AS -Os -Gsw -W2
LFLAGS=/NOD

generic.obj : generic.c generic.h              ; compile the C file
    $(CC) $(CFLAGS) generic.c

generic.res: generic.rc generic.ico            ; compile the resource file
    rc -r generic.rc

generic.exe : generic.obj generic.def generic.res    ;Link'm together
    link $(LFLAGS) generic , , , libw slibcew,generic
    rc generic.res
```

最后一个要创建的文件是程序的图标。用 SDKPaint 应用程序能够完成图标文件类型的创建。用 GENERIC.ICO 来保存创建的图标。一旦得到了所有文件，就可以在 DOS 下通过执行"NMAKE GENERIC"命令来建立工作程序了。如果还没有这样做过，我们建议试一试。GENERIC 应用程序是这本书大部分程序范例的基础。可以在文件管理器 (File Manager) 中，通过双点取 GENERIC.EXE 文件来运行它；也可从通过程序管理器 (Program Manager) 中的"Program Run"来运行程序；还可以直接在 DOS 下通过执行"WIN GENERIC"命令来运行程序。