

第七部分 附录

附录A 建立环境

读者要想建立本书中的示例程序，必须要对编译程序和链接程序的开关选项进行设置。笔者试图将这些设置方面的细节从示例程序中隔离出来，把所有这些设置放在一个头文件里。这个头文件就是CmmHdr.h，它包含在所有示例程序的源代码文件中。

因为无法将所有的设置都放在这个头文件里，我们对每个示例程序的项目设置做了一些改变。对每个项目，我们显示Project Settings对话框，然后做下面所说的改变。

- 在General栏，设定Output Files目录，这样所有最终的.exe和.dll文件都在一个目录之下。
- 在C/C++栏，选择 Code Generation条目，并对 Use Run-Time Library字段选择 Multithreaded DLL。

这样就可以了。我只明确改变了两个设置，而接受了其他所有的默认设置。注意要对每个项目的Debug建立和Release建立都做上述两个改变。我可以在源代码中设定所有其他的编译程序和链接程序的设置，当你在你的项目中使用这里的任何源代码模块时，这些设置都将起作用。

A.1 CmmHdr.h头文件

所有的示例程序都要包含CmmHdr.h头文件，并且要在其他头文件之前包含。笔者编写的CmmHdr.h列在清单A-1里。这个文件给笔者带来不少便利。这个文件包含宏、链接程序指令、还有一些其他所有示例程序公用的内容。当我想做某些实验时，我只需修改并重建（rebuild）所有的示例程序。CmmHdr.h在所附光盘的根目录下。

这个附录的其余部分将分别讨论CmmHdr.h文件的每一节，解释每一节的基本原理，并描述在重建所有示例程序之前，如何及为什么要对这个文件进行修改。

A.1.1 Windows版本建立选项

因为有些示例程序调用了Microsoft Windows 2000中提供的新函数，本节定义_WIN32_WINNT符号如下：

```
#define _WIN32_WINNT 0x0500
```

这样做是因为新的Windows 2000函数在Windows头文件中被定义成下面这样的原型：

```
#if (_WIN32_WINNT >= 0x0500)
```

```
：
```

```
WINBASEAPI
```

```
BOOL
```

```
WINAPI
```

```
AssignProcessToJobObject(
```

```
    IN HANDLE hJob,
```

```
    IN HANDLE hProcess
```

```
);
```

⋮

```
#endif /* _WIN32_WINNT >= 0x0500 */
```

除非像我这样专门定义 `_WIN32_WINNT`（在包含 `Windows.h` 之前），否则这些新函数的原型就没有被声明，当试图调用这些函数时，编译程序将产生错误。微软用 `_WIN32_WINNT` 符号来保护这些函数，以使程序员开发的应用程序能够运行在 Windows 98 及 Windows NT 的多个版本上。

A.1.2 Unicode 建立选项

笔者编写的所有这些示例程序既可按 ANSI 来编译，也可按 Unicode 来编译。当针对 x86 CPU 体系结构来编译这些程序时，ANSI 为默认选择，这样程序可以在 Windows 98 上执行。但对其他 CPU 体系结构建立程序就要用 Unicode，这样程序可以占用较少的内存，并且执行得更快。

为了对 x86 体系结构建立 Unicode 版本，只需将定义 `UNICODE` 的那一行代码的注释符去掉，并重建程序。通过在 `CmmHdr.h` 定义 `UNICODE` 宏，可以很容易地控制如何建立示例程序。关于 Unicode 的详细内容，可参见第 2 章。

A.1.3 窗口定义和第 4 级警告

笔者在开发软件时，总是想保证代码的编译不受错误和警告的限制。我还喜欢在可能最高警告级上进行编译，这样编译程序可以替我做大多数工作，甚至为我检查很小的细节。对于 Microsoft C/C++ 编译程序，这将意味着我要使用第 4 级警告来建立示例程序。

遗憾的是，微软的操作系统开发部在关于使用第 4 级警告做编译方面，与我没有共同的思想。其结果，当我使用第 4 级警告编译示例程序时，Windows 头文件中的许多行引起编译器产生警告。幸好，这些警告并不表示代码中有问题。大多数情况是由于 C 语言中非传统的用法所引起的，这些用法依赖编译程序的扩展，几乎所有与 Windows 兼容的编译程序厂商都实现了这些扩展。

本节我确保警告级设定为 3，而且 `CmmHdr.h` 包含标准的 `Windows.h` 头文件。当包含了 `Windows.h` 时，在我编译其余代码时就设置第 4 级警告。在第 4 级警告上，编译程序对那些我不认为有问题的内容发出“警告”，这样我通过使用 `#pragma warning` 指令显式地告诉编译程序忽略某些良性的警告错。

A.1.4 Pragma 消息帮助宏

在我编写代码时，我喜欢让代码的某些部分能够立即运行起来，然后再完善它。为了提醒自己要注意某些代码，我习惯于加入下面这样一行代码：

```
#pragma message("Fix this later")
```

当编译程序对这一行进行编译时，它会输出一个字符串提醒我还需要再做一些工作。但这条消息不怎么有用。我决定寻找一种办法，让编译程序输出源代码文件的名称，以及 `pragma` 出现的行号。这样，我不光知道要做一些工作，而且能够立刻确定在什么地方做。

为了达到这个目的，需要使用一系列宏来修饰 `pragma message` 指令。可以这样使用 `chMSG` 宏。

```
#pragma chMSG(Fix this later)
```

当编译程序编译上面这一行代码时，会产生这样一行内容：

使用 Microsoft Visual Developer Studio，在输出窗口上双击这一行，将会自动定位到相应文件的确切位置上。

```
C:\CD\CmnHdr.h(82):Fix this later
```

还有一个方便之处，chMSG宏不要求对文本串使用引号。

A.1.5 chINRANGE和chDIMOF宏

我时常在编写程序时使用这两个方便有用的宏。第一个宏 chINRANGE，用来查看一个数值是否在另外两个数值之间。第二个宏 chDIMOF，只是返回一个数组中元素的数目。这个宏是用sizeof操作符先计算整个数组的字节数，然后再用这个数除以数组中一个数据项所占的字节数，从而得出结果。

A.1.6 chBEGINTHREADEX宏

本书中的所有多线程示例程序都使用了微软的 C / C++ 运行时函数库中的 _beginthreadex 函数，而不是操作系统的 CreateThread 函数。我使用这个函数是因为 _beginthreadex 函数为新线程做好了准备，使新线程能够使用 C/C++ 运行时函数库中的函数，而且还因为它保证在线程返回时清除每个线程的 C/C++ 运行时库信息（见第6章有关细节）。但遗憾的是 _beginthreadex 函数的原型是这样的。

unately, the *_beginthreadex* function is protc

```
unsigned long __cdecl _beginthreadex(
    void *,
    unsigned,
    unsigned (__stdcall *)(void *),
    void *,
    unsigned,
    unsigned *);
```

尽管 _beginthreadex 函数用的参数值同 CreateThread 函数用的参数值是一样的，但二者的参数的数据类型都不相匹配。CreateThread 函数的原型是这样的：

```
typedef DWORD (WINAPI *PTHREAD_START_ROUTINE)(PVOID pvParam);
```

```
HANDLE CreateThread(
    PSECURITY_ATTRIBUTES psa,
    DWORD cbStack,
    PTHREAD_START_ROUTINE pfnStartAddr,
    PVOID pvParam,
    DWORD fdwCreate,
    PDWORD pdwThreadId);
```

微软在建立 _beginthreadex 函数的原型时没有使用 Windows 数据类型。这是因为微软的 C/C++ 运行时库开发组不想对操作系统开发组有任何依赖。这使得 _beginthreadex 函数的使用更加困难。

微软定义 _beginthreadex 函数原型的方式实际上存在着两个问题。首先，用于这个函数的一些数据类型同用于 CreateThread 函数的原始类型不匹配。例如 Windows 数据类型 DWORD 的定义是这样的：

```
typedef unsigned long DWORD;
```

这个数据类型用于 CreateThread 函数的 cbStack 参数以及 fdwCreate 参数。问题是函数 _beginthreadex 将这两个参数的原型定义为 unsigned，实际意思是 unsigned int。编译程序将 unsigned int 看成是与 unsigned long 不同的东西，并且产生一个警告。_beginthreadex 函数不属于标准的 C/C++ 运行时函数库，只是作为调用 CreateThread 函数的替代手段而存在，所以微软应该按下面的形式来定义 _beginthreadex 的原型，这样就不会产生警告了：

```
unsigned long __cdecl _beginthreadex(
```

```
void *psa,
unsigned long cbStack,
unsigned (__stdcall *) (void *pvParam),
void *pvParam,
unsigned long fdwCreate,
unsigned long *pdwThreadId);
```

第二个问题是第一个问题的小变种。_beginthreadex函数返回一个unsigned long型的值，代表新建立线程的句柄。程序中通常用HANDLE型数据变量来保存这个返回值：

```
HANDLE hThread = _beginthreadex(...);
```

上面这行代码又使编译程序产生另一个警告错。为了避免编译程序警告，必须改写这一行代码，引入一个转换（cast）：

```
HANDLE hThread = (HANDLE) _beginthreadex(...);
```

这又是一个不方便之处。为了方便起见，我在 CmnHdr.h中定义了一个 chBEGINTHR EAD EX宏，替我执行所有这些转换：

```
typedef unsigned (__stdcall *PTHREAD_START) (void *);
```

```
#define chBEGINTHREAD EX(psa, cbStack, pfnStartAddr, \
    pvParam, fdwCreate, pdwThreadId) \
    ((HANDLE) _beginthreadex( \
        (void *) (psa), \
        (unsigned) (cbStack), \
        (PTHREAD_START) (pfnStartAddr), \
        (void *) (pvParam), \
        (unsigned) (fdwCreate), \
        (unsigned *) (pdwThreadId)))
```

A.1.7 对x86平台的调试断点改进

即使进程没有在一个调试程序下运行，有时候我也想在程序代码中强制一个断点。在Windows中要做这件事，可以让线程调用DebugBreak函数。这个函数在kernel32.dll中，可以使一个调试程序同进程挂接。当调试程序被挂接上时，指令指针就定位在引起断点的CPU指令上。这个指令包含在kernel32.dll中的DebugBreak函数里，所以为了看到我的源代码，我必须在DebugBreak函数之外单步执行。

在x86体系结构上，通过执行“int3”CPU指令来做一个断点。所以，在x86平台之上，我定义DebugBreak作为这个内联的汇编语言指令。当我的DebugBreak执行时，我不是在kernel32.dll中调用。断点发生在我的代码中，指令指针定位在下一个C/C++语句中。这样就方便多了。

A.1.8 建立软件异常代码

当处理软件异常时，必须建立你自己的32位异常代码。这些代码遵循特定的格式（见第24章的讨论）。为了更容易地建立这些代码，我使用MAKESOFTWAREEXCEPTION宏。

A.1.9 chMB宏

chMB宏只是显示一个消息框。消息框的标题是调用进程可执行代码的全路径名。

A.1.10 chASSERT和chVERIFY宏

在我开发这些示例程序时，为了查找潜在的问题，我在整个代码中多处使用hASSERT宏。这

一个宏测试由x所标识的表达式是否为TRUE,如果不是,则显示一个消息框指出失败的文件、行和表达式。在程序的发行建立中,这个宏什么也不做。chVERIFY宏与chASSERT宏差不多,区别在于不论是调试建立(debug build)还是发行建立(release build),chVERIFY都要对表达式进行测试。

A.1.11 chHANDLE_DLGMSG宏

当你通过对话框使用消息分流器时,不应该使用微软的 WindowsX.h头文件中的HANDLE_MSG宏,因为这个宏并不能返回TRUE或FALSE来指出消息是否由对话框的过程来处理。我定义的chHANDLE_DLGMSG宏会通知窗口消息的返回值,适当地处理返回值,以便在一个对话框过程中使用。

A.1.12 chSETDLGICONS宏

由于多数示例程序使用一个对话框作为主窗口,你必须手工改变对话框图标,以便让它正确地显示在Taskbar(任务条)、任务切换窗口和程序本身的标题上。当对话框接收到一个WM_INITDIALOG消息时,总要调用chSETDLGICONS宏,以正确设置图标。

A.1.13 OS版本检查内联函数

本书的大多数示例程序可运行在所有平台上,但也有一些程序要求一些 Windows 95和Windows 98所不支持的特性,有些程序要求一些只在Windows 2000中提供的特性。每个程序在初始化时要检查宿主系统的版本,如果要求更适用的操作系统时,就显示一个通知。

对那些不能在Windows 95和Windows 98上运行的程序,你会看到,在程序的_tWinMain函数中有一个对Windows9xNotAllowed函数的调用。对于要求Windows 2000的示例程序,你会看到在程序的_tWinMain函中有一个对chWindows2000Required函数的调用。

A.1.14 确认宿主系统是否支持Unicode

Windows 98不能像Windows 2000那样完全支持Unicode。实际上,调用Unicode函数的程序不能在Windows 98上运行。但遗憾的是,如果调用一个为Unicode编译的程序,Windows98不会给出任何通知信息。对本书中的程序,这意味着这些程序从开始到结束,都不会有它们想执行的提示信息。

这确实是一个难题。我需要有一种办法能够知道我的程序是对Unicode建立的,但可能在Windows 98系统上运行。所以我建立了一个CUnicodeSupported C++类。这个类的构造函数只是检查宿主系统是不是对Unicode有良好的支持,如果不是,就显示一个消息框,并且进程结束。

读者会看到在CmnHdr.h中,我建立了这个类的一个全局的静态实例。当我的程序启动时,C/C++运行时库启动代码调用这个对象的构造函数。如果这个构造函数检测到操作系统完全支持Unicode,构造函数返回而程序继续执行。通过建立这个类的全局实例,我不需要在每个示例程序的源代码模块中再增加特殊的代码。对于非Unicode的程序建立,不需要声明或实例化上述的C++类。让程序只管运行就是。

A.1.15 强制链接程序寻找(w)WinMain进入点函数

本书以前版本的一些读者,将书中我的源代码模块添加到他们自己的 VisualC++项目中,但在建立项目时出现链接错误。问题的原因是他们创建了 Win32 Console Application项目,导

致链接程序去寻找(w)main进入点函数。因为本书中所有示例程序都是 GUI程序，所以我的代码都有一个_tWinMain进入点函数。这就是链接程序为什么要报错。

我的回答是，他们应该删除原来的项目，用 Visual C++ 建立新的 Win32 Application 项目（注意在项目类型中不能出现“ Console ”一词），再将我的源代码加进去。链接程序寻找一个(w)WinMain进入点函数，而这在我的代码中已提供，项目应该能够建立。

为了减少我收到的有关这个问题的电子邮件的数量，我在 CmnHdr.h 中加入了一个 pragma，强制链接程序去寻找(w)WinMain进入点函数，即使是用 Visual C++ 建立了一个 Win32 Console Application 项目。

在第4章，我详细说明了 Visual C++ 项目类型的有关内容，链接程序如何选择进入点函数，及如何重载链接程序的默认动作等。下面的清单 A-1 是 Cmn Hdr. h 头文件。

清单A-1 CmnHdr.h头文件

CmnHdr.h

```

/*****
Module: CmnHdr.h
Notices: Copyright (c) 2000 Jeffrey Richter
Purpose: Common header file containing handy macros and definitions
        used throughout all the applications in the book.
        See Appendix A.
*****/

#pragma once // Include this header file once per compilation unit

////////// Windows Version Build Option //////////

#define _WIN32_WINNT 0x0500
// #define WINVER 0x0500

////////// Unicode Build Option //////////

// If we are not compiling for an x86 CPU, we always compile using Unicode.
#ifndef _M_IX86
#define UNICODE
#endif

// To compile using Unicode on the x86 CPU, uncomment the line below.
// #define UNICODE

// When using Unicode Windows functions, use Unicode C-Runtime functions too.
#ifdef UNICODE
#define _UNICODE
#endif

////////// Include Windows Definitions //////////
#pragma warning(push, 3)

```



```
#include <Windows.h>
#pragma warning(pop)
#pragma warning(push, 4)

////////// Verify that the proper header files are being used //////////

#ifdef WT_EXECUTEINPERSISTENTTHREAD
#pragma message("You are not using the latest Platform SDK header/library ")
#pragma message("files. This may prevent the project from building correctly.")
#endif

////////// Allow code to compile cleanly at warning level 4 //////////

/* nonstandard extension 'single line comment' was used */
#pragma warning(disable:4001)

// unreferenced formal parameter
#pragma warning(disable:4100)

// Note: Creating precompiled header
#pragma warning(disable:4699)

// function not inlined
#pragma warning(disable:4710)

// unreferenced inline function has been removed
#pragma warning(disable:4514)

// assignment operator could not be generated
#pragma warning(disable:4512)

////////// Pragma message helper macro //////////

/*
When the compiler sees a line like this:
    #pragma chMSG(Fix this later)
it outputs a line like this:

    c:\CD\CmnHdr.h(82):Fix this later

You can easily jump directly to this line and examine the surrounding code.
*/

#define chSTR2(x)      #x
#define chSTR(x)      chSTR2(x)
#define chMSG(desc) message(__FILE__ "(" chSTR(__LINE__) "):" #desc)

////////// chINRANGE Macro //////////
```

```
// This macro returns TRUE if a number is between two others.
#define chINRANGE(low, Num, High) (((low) <= (Num)) && ((Num) <= (High)))

//////////////////////////////// chDIMOF Macro //////////////////////////////////

// This macro evaluates to the number of elements in an array.
#define chDIMOF(Array) (sizeof(Array) / sizeof(Array[0]))

//////////////////////////////// chBEGINTHREADEX Macro //////////////////////////////////

// This macro function calls the C runtime's _beginthreadex function.
// The C runtime library doesn't want to have any reliance on Windows' data
// types such as HANDLE. This means that a Windows programmer needs to cast
// values when using _beginthreadex. Since this is terribly inconvenient,
// I created this macro to perform the casting.
typedef unsigned (__stdcall *PTHREAD_START) (void *);

#define chBEGINTHREADEX(psa, cbStack, pfnStartAddr, \
    pvParam, fdwCreate, pdwThreadId) \
    ((HANDLE)_beginthreadex( \
        (void *) (psa), \
        (unsigned) (cbStack), \
        (PTHREAD_START) (pfnStartAddr), \
        (void *) (pvParam), \
        (unsigned) (fdwCreate), \
        (unsigned *) (pdwThreadId)))

//////////////////////////////// DebugBreak Improvement for x86 platforms //////////////////////////////////

#ifdef _X86_
#define DebugBreak() __asm { int 3 }
#endif

//////////////////////////////// Software Exception Macro //////////////////////////////////

// Useful macro for creating your own software exception codes
#define MAKESOFTWAREEXCEPTION(Severity, Facility, Exception) \
    ((DWORD) ( \
        /* Severity code */ (Severity) | \
        /* MS(0) or Cust(1) */ (1 << 29) | \
        /* Reserved(0) */ (0 << 28) | \
        /* Facility code */ (Facility << 16) | \
        /* Exception code */ (Exception << 0)))

//////////////////////////////// Quick MessageBox Macro //////////////////////////////////
```



```

inline void chMB(PCSTR s) {
    char szTMP[128];
    GetModuleFileNameA(NULL, szTMP, chDIMOF(szTMP));
    MessageBoxA(GetActiveWindow(), s, szTMP, MB_OK);
}

//////////////////////////////////// Assert/Verify Macros //////////////////////////////////////

inline void chFAIL(PSTR szMsg) {
    chMB(szMsg);
    DebugBreak();
}

// Put up an assertion failure message box.
inline void chASSERTFAIL(LPCSTR file, int line, PCSTR expr) {
    char sz[128];
    wsprintfA(sz, "File %s, line %d : %s", file, line, expr);
    chFAIL(sz);
}

// Put up a message box if an assertion fails in a debug build.
#ifdef _DEBUG
#define chASSERT(x) if (!(x)) chASSERTFAIL(__FILE__, __LINE__, #x)
#else
#define chASSERT(x)
#endif

// Assert in debug builds, but don't remove the code in retail builds.
#ifdef _DEBUG
#define chVERIFY(x) chASSERT(x)
#else
#define chVERIFY(x) (x)
#endif

//////////////////////////////////// chHANDLE_DLGMSG Macro //////////////////////////////////////

// The normal HANDLE_MSG macro in WindowsX.h does not work properly for dialog
// boxes becauseDlgProc return a BOOL instead of an LRESULT (like
// WndProcs). This chHANDLE_DLGMSG macro corrects the problem:
#define chHANDLE_DLGMSG(hwnd, message, fn) \
    case (message): return (SetDlgMsgResult(hwnd, uMsg, \
        HANDLE_##message((hwnd), (wParam), (lParam), (fn))))

//////////////////////////////////// Dialog Box Icon Setting Macro //////////////////////////////////////

// Sets the dialog box icons
inline void chSETDLGICONS(HWND hwnd, int idi) {
    SendMessage(hwnd, WM_SETICON, TRUE, (LPARAM)

```

```

        LoadIcon((HINSTANCE) GetWindowLongPtr(hwnd, GWLP_HINSTANCE),
            MAKEINTRESOURCE(idi));
    SendMessage(hwnd, WM_SETICON, FALSE, (LPARAM)
        LoadIcon((HINSTANCE) GetWindowLongPtr(hwnd, GWLP_HINSTANCE),
            MAKEINTRESOURCE(idi)));
}

//////////////////// OS Version Check Macros //////////////////////

inline void chWindows9xNotAllowed() {
    OSVERSIONINFO vi = { sizeof(vi) };
    GetVersionEx(&vi);
    if (vi.dwPlatformId == VER_PLATFORM_WIN32_WINDOWS) {
        chMB("This application requires features not present in Windows 9x.");
        ExitProcess(0);
    }
}

inline void chWindows2000Required() {
    OSVERSIONINFO vi = { sizeof(vi) };
    GetVersionEx(&vi);
    if ((vi.dwPlatformId != VER_PLATFORM_WIN32_NT) && (vi.dwMajorVersion < 5)) {
        chMB("This application requires features present in Windows 2000.");
        ExitProcess(0);
    }
}

//////////////////// UNICODE Check Macro //////////////////////

// Since Windows 98 does not support Unicode, issue an error and terminate
// the process if this is a native Unicode build running on Windows 98.

// This is accomplished by creating a global C++ object. Its constructor is
// executed before WinMain.

#ifdef UNICODE

class CUnicodeSupported {
public:
    CUnicodeSupported() {
        if (GetWindowsDirectoryW(NULL, 0) <= 0) {
            chMB("This application requires an OS that supports Unicode.");
            ExitProcess(0);
        }
    }
};

// "static" stops the linker from complaining that multiple instances of the
// object exist when a single project contains multiple source files.
static CUnicodeSupported g_UnicodeSupported;

```

```
#endif
```

```
////////// Force Windows subsystem //////////
```

```
#pragma comment(linker, "/subsystem:Windows")
```

```
////////// End of File //////////
```
