

《深入浅出MFC》2/e

电子书开放自由下载 声明

致亲爱的大陆读者

我是侯捷（侯俊杰）。自从华中理工大学于1998/04 出版了我的《深入浅出MFC》1/e 简体版（易名《深入浅出Windows MFC 程序设计》）之后，陆陆续续我收到了许许多多的大陆读者来函。其中对我的赞美、感谢、关怀、殷殷垂询，让我非常感动。

《深入浅出MFC》2/e 早已于1998/05 于台湾出版。之所以迟迟没有授权给大陆进行简体翻译，原因我曾于回复读者的时候说过很多遍。我在此再说一次。

1998 年中，本书之发行公司松岗（UNALIS）即希望我授权简体版，然因当时我已在构思3/e，预判3/e 繁体版出版时，2/e 简体版恐怕还未能完成。老是让大陆读者慢一步看到我的书，令我至感难过，所以便请松岗公司不要进行2/e 简体版之授权，直接等3/e 出版后再动作。没想到一拖经年，我的3/e 写作计划并没有如期完成，致使大陆读者反而没有《深入浅出MFC》2/e 简体版可看。

《深入浅出MFC》3/e 没有如期完成的原因是，MFC 本体架构并没有什么大改变。《深入浅出MFC》2/e 书中所论之工具及程序代码虽采用VC5+MFC42，仍适用于目前的VC6+MFC421（唯，工具之画面或功能可能有些微变化）。

由于《深入浅出MFC》2/e 并无简体版，因此我时时收到大陆读者来信询问购买繁体版之管道。一来我不知道是否台湾出版公司有提供海外邮购或电购，二来即使有，想必带给大家很大的麻烦，三来两岸消费水平之差异带给大陆读者的负担，亦令我深感不安。

因此，此书虽已出版两年，鉴于仍具阅读与技术上的价值，鉴于繁简转译制作上的费时费工，鉴于我对同胞的感情，我决定开放此书内容，供各位免费阅读。我已为《深入浅出MFC》2/e 制作了PDF 格式之电子文件，放在 <http://www.jjhou.com> 供自由下载。北京<http://expert.csdn.net/jjhou> 有侯捷网站的一个GBK mirror，各位也可试着自该处下载。

我所做的这份电子书是繁体版，我没有精力与时间将它转为简体。这已是我能为各位尽力的极限。如果（万一）您看不到文件内容，可能与字形的安装有关-虽然我已尝试内嵌字形。anyway，阅读方面的问题我亦没有精力与时间为您解决。请各位自行开辟讨论区，彼此交换阅读此电子书的solution。请热心的读者告诉我您阅读成功与否，以及网上讨论区（如有的话）在哪里。

曾有读者告诉我，《深入浅出MFC》1/e 简体版在大陆被扫描上网。亦有读者告诉我，大陆某些书籍明显对本书侵权（详细情况我不清楚）。这种不尊重作者的行为，我虽感遗憾，并没有太大的震惊或难过。一个社会的进化，终究是一步一步衍化而来。台湾也曾经走过相同的阶段。但盼所有华人，尤其是我们从事智能财产行为者，都能够尽快走过灰暗的一面。

在现代科技的协助下，文件影印、文件复制如此方便，智财权之尊重有如「君子不欺暗室」。没有人知道我们私下的行为，只有我们自己心知肚明。《深入浅出MFC》2/e 虽免费供大家阅读，但此种作法实非长久之计。为计久长，我们应该尊重作家、尊重智财，以良好（至少不差）的环境培养有实力的优秀技术作家，如此才有源源不断的好书可看。

我的近况，我的作品，我的计划，各位可从前述两个网址获得。欢迎各位写信给我（jjhou@ccca.nctu.edu.tw）。虽然不一定能够每封来函都回复，但是我乐于知道读者的任何点点滴滴。

关于《深入浅出 MFC》2/e 电子书

《深入浅出 MFC》2/e 电子书共有五个档案：

档名	内容	大小 bytes
dissecting MFC 2/e part1.pdf	chap1~chap3	3,384,209
dissecting MFC 2/e part2.pdf	chap4	2,448,990
dissecting MFC 2/e part3.pdf	chap5~chap7	2,158,594
dissecting MFC 2/e part4.pdf	chap8~chap16	5,171,266
dissecting MFC 2/e part5.pdf	appendix A,B,C,D	1,527,111

每个档案都可个别阅读。每个档案都有书签（亦即目录连接）。每个档案都不需密码即可打开、选择文字、打印。

请告诉我您的资料

每一位下载此份电子书的朋友，我希望您写一封 email 给我（jjhou@ccca.nctu.edu.tw），告诉我您的以下资料，俾让我对我的读者有一些基本瞭解，谢谢。

姓名：

现职：

毕业学校科系：

年龄：

性别：

居住省份（如是台湾读者，请写县市）：

对侯捷的建议：

-- the end



欲흥 그 후 20년



深入淺出 MFC
2nd Edition

Visual C++ 整合开发环境

如果MFC是箭，Visual C++ IDE（整合开发环境）便是弓。

强壮的弓，让箭飞得更远。

看过重量级战斗吗？重量级战斗都有「一棒击沉」的威力。如果现实生活中发生重量级战斗--使生涯结束、生活受威胁的那种，那么战况之激烈不言可知。如果这场战斗关系到你的程序员生涯，铃声响起时你最好付出高度注意力。

我说的是application framework。换个角度来说，我指的是整合型（全套服务的）C++ 软件开发平台。目前，所有重要厂商包括Microsoft、Borland、Symantec、Metaware 和Watcom 都已投入这个战场。在PC 领域，最著名的application framework 有两套（注）：MFC（Microsoft Foundation Class）和OWL（ObjectWindow Library），但整合开发环境（IDE）却呈百家争鸣之势。

注：第三套可以说是IBM VisualAge C++ 的Open Class Library。VisualAge C++ 和Open Class Library 不单是OS/2 上的产品，IBM 更企图让它们横跨Windows 世界。

在这一章中，我将以概观的方式为你介绍Visual C++ 的整合环境，目的在认识搭配在MFC 周遭的这些强棒工具的操作性与功能性，实地了解这一整套服务带给我们什么样的便利。除非你要以你的PE2 老古董把程序一字一句co co co 地敲下去，否则Visual C++ 的这些工具对软件开发的重要性不亚于MFC。我所使用的Visual C++ 版本是v5.0（搭配MFC 4.21）。

安装与设定

VC++ 5.0 采CD-ROM 包装，这是现代软件日愈肥胖后的趋势。内存最好有16MB，跑起来才会舒服些；硬盘空间的需求量视不同的安装方式（图4-1f）而定，你可以从画面上清楚看到；只要硬盘够大，我当然建议采用Typical Installation。

Visual C++ 5.0 光盘片中有AUTORUN.INF 文件，所以其Setup 程序会在Windows 95 和 Windows NT 4.0 的autoplay 功能下自动执行。Setup 程序会侦测你的环境，如果没有找到Internet Explorer（IE）3.01，它会建议你安装或更新之（图4-1a）。VC++ 5.0 盘中附有IE 3.01（英文版）。为什么要先安装Internet Explorer 呢？因为微软的所有Visual Tools（包括Visual C++、Visual Basic、Visual FoxPro、Visual J++、Visual InterDev 等）都集中由所谓的Visual Studio（图4-1c）管理，而这些工具有一个极大的目标，就是要协助开发Internet 应用软件，所以它们希望能够和Internet Explorer 有所搭配。

如果你原已有Visual C++ 4.x，Setup 程序会侦测到并给你一个警告消息（图4-1e）。通常你可能会想保留原有的版本并试用新的版本（至少我的心态是如此），因此你可能担心Visual C++ 5.0 会不会覆盖掉4.x 版。放心，只要你在图4-1f 中指定安装目的地（子目录）和原版本不同，即可避免所谓覆盖的问题。以我的情况为例，我的Visual C++ 4.2 放在E:\MSDEV 中，而我的Visual C++ 5.0 安装在E:\DEVSTUDIO 中。



图4-1a Visual C++ 5.0 建议你安装最新的IE 3.01（英文版）。



图4-1b 当你安装IE 3.01 (英文版) 时，可能会和你现有的IE 中文版有些版本冲突。我的经验是依其建议，保留现有的文件。

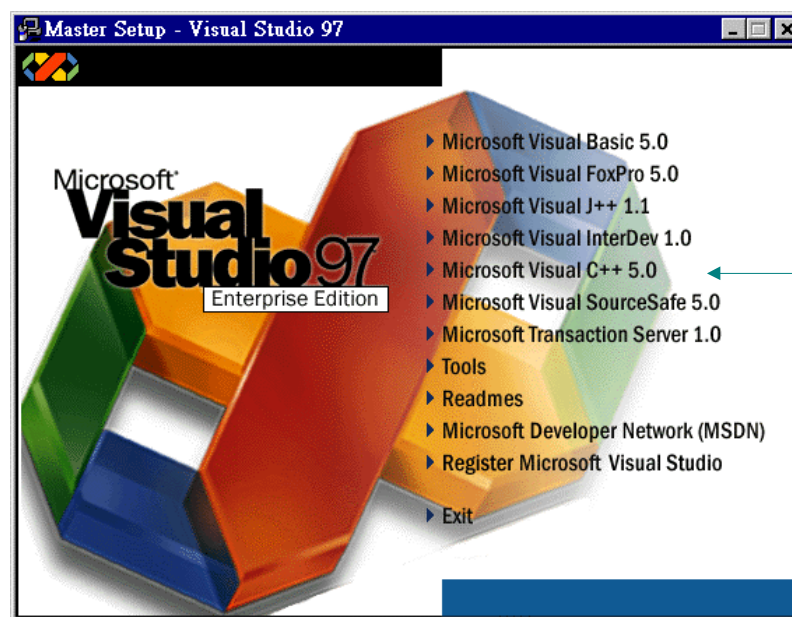


图4-1c Visual C++ 5.0 Setup 程序画面。请把鼠标移到右上角第五个项目 "Microsoft Visual C++ 5.0" 上面，并按下左键。

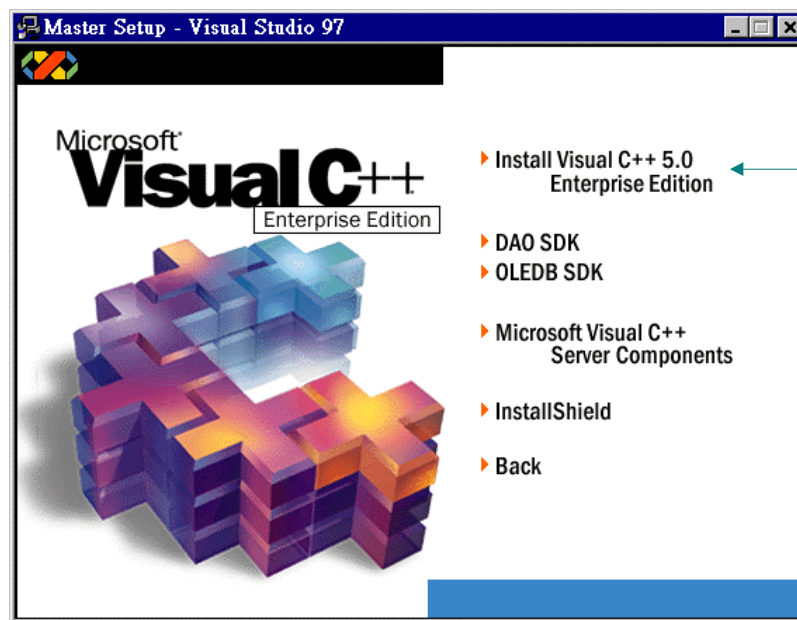


图4-1d 你可以安装Visual C++ 5.0 中的这些套件。其中InstallShield 是一套协助你制作安装软件的工具。

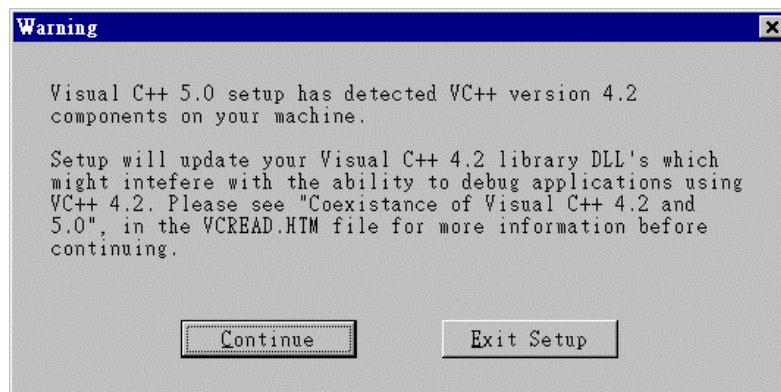


图4-1e Setup 程序侦测到我已经有了Visual C++ 4.2 , 于是提出警告。

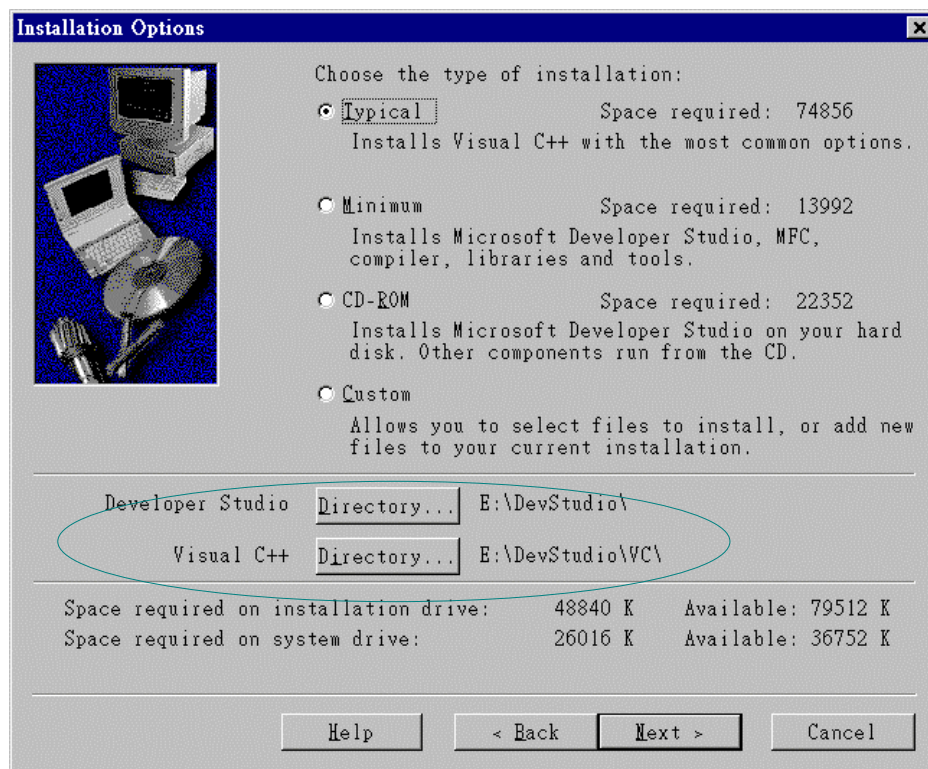


图4-1f Visual C++ 提供四种安装方式。中央偏下的【Directory...】钮允许我们设定安装目的地（硬盘目录）。

早期的Visual C++ 版本曾经要求你在AUTOEXEC.BAT 中加入这行命令：

```
SHARE /L:500 /F:5100
```

为的是让DOS 借着SHARE.EXE 的帮助支持「文件共享与锁定功能」。如今已不需要，因为Windows 95 及Windows NT 已内建此项能力。

这个整合环境并不要求你设定什么环境变量，它自己内部会在安装时记录该有的路径。如果你习惯以命令列的方式在DOS 环境（也就是Windows 95 或Windows NT 的DOS 窗口）下编译联结，那么你必须小心设定好PATH、LIB、INCLUDE 等环境变量。如果你有许多套开发工具，为每一个环境准备一个批次档是个不错的作法。下面是个例子：

```
rem file : envir.bat
cls
type c:\utility\envir.txt
```

其中envir.txt 的内容是：

```
(1) CWin95 & Visual C++ 1.5
(2) CWin95 & Visual C++ 2.0
(3) CWin95 & Visual C++ 4.0
(4) DDK
(5) CWin95 & Visual C++ 5.0
```

每当欲使用不同的工具环境，就执行envir.bat，然后再选择一个号码。举个例，3.BAT 的内容是：

```
rem 3.bat
rem Win95 & Visual C++ 4.0
@echo off
set TOOLROOTDIR=E:\MSDEV
rem
set PATH=E:\MSDEV\BIN;D:\WIN95;D:\WIN95\COMMAND
set INCLUDE=E:\MSDEV\INCLUDE;E:\MSDEV\MFC\INCLUDE
set LIB=E:\MSDEV\LIB;E:\MSDEV\MFC\LIB
set MSDevDir=E:\MSDEV
set
```

5. BAT 的内容是：

```
rem e:\devstudio\vc\bin\vcvars32.bat
@echo off
rem
rem e:\devstu~1 == e:\devstudio
set PATH=E:\DEVSTU~1\VC\BIN;E:\DEVSTU~1\SHARED~1\BIN;D:\WIN95;D:\WIN95\COMMAND
set INCLUDE=E:\DEVSTU~1\VC\INCLUDE;E:\DEVSTU~1\VC\MFC\INCLUDE;E:\DEVSTU~1\VC\ATL\INCLUDE
set LIB=E:\DEVSTU~1\VC\LIB;E:\DEVSTU~1\VC\MFC\LIB
set
```

其中大家比较陌生的可能是VC\ATL\INCLUDE 这个设定。ATL 全名是ActiveX

Template Library，用以协助我们开发ActiveX 控制组件。关于ActiveX 控制组件的开发设计，可参考ActiveX Control Inside Out（Adam Denning/Microsoft Press）一书（ActiveX 控制元件彻底研究 / 侯俊杰译/ 松岗出版）。至于ActiveX controls 的应用，可参考本书第16 章。

上述那些那些环境变量的设定，其实VC++ 早已为我们准备好了，就放在
\\DEVSTUDIO\\VC\\BIN\\VCVARS32.BAT 中，只不过形式比较复杂一些。

如果你也喜欢（或有必要）保留多套开发环境于硬盘中，请注意出现在DOS 提示号下的编译器和联结器版本号码，以确定你叫用的的确是你所要的工具。图4-2 是Microsoft 软件开发工具的版本号码。

VC++	编译器	联结器	NMAKE	RC.EXE	MFC
Microsoft C/C++ 7.0	7.00	S5.30	1.20	3.10	1.0
Visual C++ 1.0	8.00	S5.50	1.30	3.11	2.0
Visual C++ 1.5x	8.00c	S5.60	1.40	3.11	2.5
Visual C++ 2.0	9.00	I2.50	1.50	3.50	3.0
Visual C++ 4.0	10.00	I3.00	1.60	4.00	4.0
Visual C++ 4.2	10.20	I4.20	1.61	4.00	4.2
Visual C++ 5.0	11.00	I5.00	1.62	5.00	4.21

* 联结器S: Segmented Executable Linker
I: Incremental Linker

图4-2 Microsoft 编译器平台的演化

Visual C++ 提供三种版本：学习版，专业版和企业版。三者都提供C/C++ 编译器、MFC、以及整合开发环境，可以协助建立并除错各类型应用软件：

- MFC-based EXE
- MFC-based DLL
- Win32 Application (EXE)
- Win32 Dynamic Link Library (DLL)
- Win32 Console Applications
- MFC ActiveX Controls

- ATL COM (ActiveX Template Library Component Object Model)
- ISAPI (Internet Server API) Extension Application
- Win32 Static Library

图4-3 是VC++ 5.0 专业版安装完成后的程序群组，打开Win95 的【开始/程序集】便可看到。

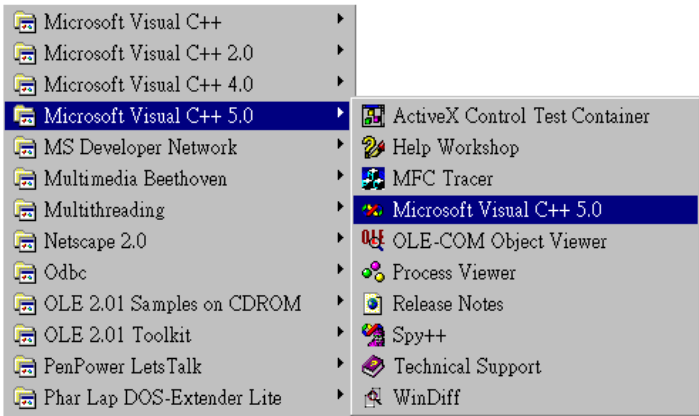


图4-3 VC++ 5.0 专业版安装完成后的程序群组 (group)

VC++ 5.0 安装完成后重要的文件分布如下。可能有些在你的硬盘，有些在光盘片上，因不同的安装方式而异：

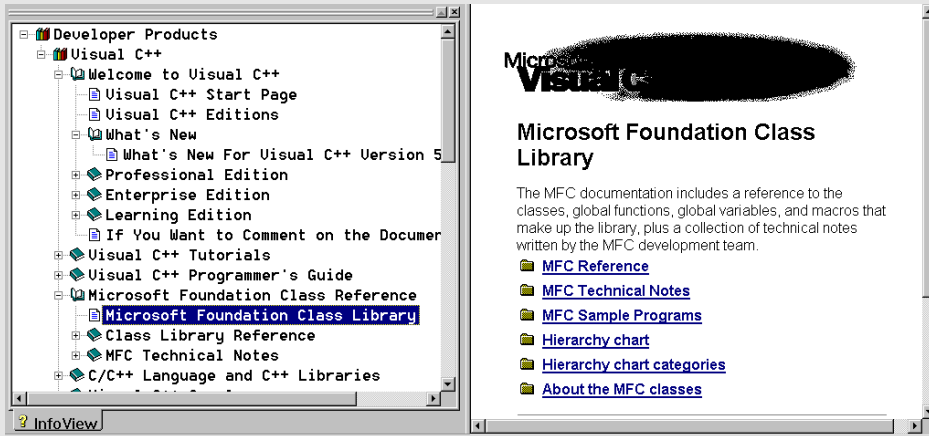
```
MSDEV <DIR>
  BIN <DIR>          各种EXE、BAT、DLL。
  DEBUG <DIR>        除错版本（各种DLLs）。
  HELP <DIR>         各种Help 文件。
  CRT <DIR>
  ATL <DIR>          ActiveX Template Library 函数库的源代码。
    INCLUDE <DIR>    ATL 的包含档（头文件）
    SRC <DIR>        ATL 的源代码
  REDIST <DIR>       这是可以自由（免费）传播的文件，包括你的应用程序售出后，
                    执行时期所需的任何DLLs，如MFC42.DLL、ODBC DLLs、
                    DAO DLLs。还包括微软公司附赠的一些OCXs。

  SAMPLES <DIR>      丰富的范例程序（请看附录 C）
    APPWIZ <DIR>
    ATL <DIR>
    COM <DIR>
    ENT <DIR>
    MFC <DIR>
    SDK <DIR>

  INCLUDE <DIR>      各种.H 文件。包括C/C++ 函数头文件、WINDOWS.H 等等。
  LIB <DIR>           各种.LIB。包括C/C++ runtime、Windows DLLs import 函数库。
  MFC <DIR>
    INCLUDE <DIR>    以AFX 开头的.H 文件（MFC 的头文件）。
    LIB <DIR>        MFC 的静态函数库（static library）。
    SRC <DIR>        MFC 的源代码（.CPP 档）。
```

手册呢？C/C++ 加上SDK 再加上MFC 共二十来本厚薄不一的手册不可能塞到宽仅五公分的VC++ 5.0 包装盒中。所有的手册都已电子化到那片CD-ROM 去了。像我这种看书一定得拿支笔的人，没什么比这更悲哀的事。不是没有补救办法，再花个数千元就可得到VC++ 印刷手册，另一个数千元可再得到SDK 印刷手册。

MFC Tech Notes



The screenshot shows the Visual C++ 5.0 Developer Products tree on the left, with the 'Microsoft Foundation Class Library' item selected. The right pane displays the 'Microsoft Foundation Class Library' page, which includes a description of the MFC documentation and a list of links: MFC Reference, MFC Technical Notes, MFC Sample Programs, Hierarchy chart, Hierarchy chart categories, and About the MFC classes.

VC++ 5.0 的Online Help 中有一些好东西：为数69 篇的宝贵技术文件。以下是一份列表。

文件1 至17 是一般性主题，适用于MFC 1.0 和2.0；文件18 和19 专注在如何将MFC 1.0 程序移植到MFC 2.0；文件20 至36 适用于MFC 2.0（或更高版本）；文件37 适用于32 位版MFC；文件38 至48 适用于MFC 2.5（或更高版本）；文件49 至52 适用于MFC 3.0（或更高版本）；文件53 至69 适用于MFC 4.0（或更高版本）。某些号码跳掉是因为MFC 1.0 的老东西不值得再提。

1. Window Class Registration
2. Persistent Object Data Format
3. Mapping of Windows Handles to Objects
4. C++ Template Tool
6. Message Maps
7. Debugging Trace Options
8. MFC OLE Support
11. Using MFC as Part of a DLL
12. Using Windows 3.1 Robustness Features

14. Custom Controls
15. Windows for Pen
16. Using C++ Multiple Inheritance with MFC
17. Destroying Window Objects
18. Migrating OLE Applications From MFC 1.0 to MFC 2.0
19. Migrating MFC 1.0 Applications to MFC 2.0
20. ID Naming and Numbering Conventions
21. Command and Message Routing
22. Standard Commands Implementation
23. Standard MFC Resources
24. MFC-Defined Messages and Resources
25. Document, View, and Frame Creation
26. DDX and DDV Routines
27. Emulation Support for Visual Basic Custom Controls
28. Context-Sensitive Help Support
29. Splitter Windows
30. Print Preview
31. Control Bars
32. MFC Exception Mechanism
33. DLL Version of MFC
34. Writing a Windows 3.0 Compatible MFC Application
35. Using Multiple Resource Files and Header Files with App Studio
36. Using CFormView with AppWizard and ClassWizard
37. Multithreaded MFC 2.1 Applications (32-bit specific)
38. MFC/OLE IUnknown Implementation
39. MFC/OLE Automation Implementation
40. MFC/OLE In-Place Resizing and Zooming
41. MFC/OLE1 Migration to MFC/OLE2
42. ODBC Driver Developer Recommendations
43. RFX Routines
44. MFC support for DBCS
45. MFC/Database support for Long Varchar/Varbinary
46. Commenting Conventions for the MFC classes

47. Relaxing Database Transaction Requirements
48. Writing ODBC Setup and Administration Programs for MFC Database Applications
49. MFC/OLE MBCS to Unicode Translation Layer (MFCANS32)
50. MFC/OLE Common Dialogs (MFCUIx32)
51. Using CTL3D Now and in the Future
52. Writing Windows 95 Applications with MFC 3.1
53. Custom DFX Routings for DAO Database Classes
54. Calling DAO Directory while Using MFC DAO Classes
55. Migrating MFC ODBC Database Classes Application to MFC DAO Classes
56. Installation of MFC Components
57. Localization of MFC Components
58. MFC Module State Implementation
59. Using MFC MBCS/Unicode Conversion Macros
60. The New Windows Common Controls
61. ON_NOTIFY and WM_NOTIFY Messages
62. Message Reflection for Windows Controls
63. Debugging Internet Extension DLLs
64. Apartment-Model Threading in OLE Controls
65. Dual-Interface Support for OLE Automation Servers
66. Common MFC 3.x to 4.0 Porting Issues
67. Database Access from an ISAPI Server Extension
68. Performing Transactions with the Microsoft Access 7 ODBC Driver
69. Processing HTML Forms Using Internet Server Extension DLLs and Command Handlers

以下是MFC Tech Notes 的性质分类：

■ MFC and Windows

TN001: Window Class Registration

TN003: Mapping of Windows Handles to Objects

TN012: Using MFC with Windows 3.1 Robustness Features

TN015: Windows for Pen

TN017: Destroying Window Objects

TN034: Writing a Windows 3.0 Compatible MFC Application

TN051: Using CTL3D Now and in the Future

TN052: Writing Windows 95 Applications with MFC3.1

■ MFC Architecture

TN002: Persistent Object Data Format

TN004: C++ Template Tool

TN006: Message Maps

TN016: Using C++ Multiple Inheritance with MFC

TN019: Updating Existing MFC Applications to MFC 3.0

TN021: Command and Message Routing

TN022: Standard Commands Implementation

TN025: Document, View, and Frame Creation

TN026: DDX and DDV Routines

TN029: Splitter Windows

TN030: Customizing Printing and Print Preview

TN031: Control Bars

TN032: MFC Exception Mechanism

TN037: Multithreaded MFC 2.1 Applications

TN044: MFC Support for DBCS

TN046: Commenting Conventions for the MFC Classes

TN058: MFC Module State Implementation

TN059: Using MFC MBCS/Unicode Conversion Macros

TN066: Common MFC 3.x to 4.0 Porting Issues

■ MFC Controls

TN014: Custom Controls

TN027: Emulation Support for Visual Basic Custom Controls

TN060: Windows Common Controls

TN061: ON_NOTIFY and WM_NOTIFY Messages

TN062: Message Reflection for Windows Controls

■ MFC Database

TN042: ODBC Driver Developer Recommendations
TN043: RFX Routines
TN045: MFC/Database Support for Long Varchar/Varbinary
TN047: Relaxing Database Transaction Requirements
TN048: Writing ODBC Setup and Administration Programs for MFC Database Applications
TN053: Custom DFX Routines for MFC DAO Classes
TN054: Calling DAO Directly While Using MFC DAO Classes
TN055: Migrating MFC ODBC Database Class Applications to MFC DAO Classes
TN068: Performing Transactions with the Microsoft Access 7 ODBC Driver

■ MFC Debugging

TN007: Debugging Trace Options

■ MFC DLLs

TN011: Using MFC as Part of a DLL
TN033: DLL Version of MFC
TN056: Installation of MFC Components
TN057: Localization of MFC Components

■ MFC OLE

TN008: MFC OLE Support
TN018: Migrating OLE Applications from MFC 1.0 to MFC 2.0
TN038: MFC/OLE IUnknown Implementation
TN039: MFC/OLE Automation Implementation
TN040: MFC/OLE In-Place Resizing and Zooming
TN041: MFC/OLE1 Migration to MFC/OLE2
TN049: MFC/OLE MBCS to Unicode Translation Layer (MFCANS32)
TN050: MFC/OLE Common Dialogs (MFCUIx32)
TN064: Apartment-Model Threading in OLE Controls
TN065: Dual-Interface Support for OLE Automation Servers

■ MFC Resources

TN020: ID Naming and Numbering Conventions

TN023: Standard MFC Resources

TN024: MFC-Defined Messages and Resources

TN028: Context-Sensitive Help Support

TN035: Using Multiple Resource Files and Header Files with Visual C++

TN036: Using CFormView with AppWizard and ClassWizard

■ MFC Internet

TN063: Debugging Internet Extension DLLs

TN067: Database Access from an ISAPI Server Extension

TN069: Processing HTML Forms Using Internet Server Extension DLLs and
Command Handlers

四个重要的工具

完全依赖整合环境，丢掉PE2（或其它什么老古董），这是我的良心建议。也许各个工具的学习过程会有些阵痛，但代价十分值得。我们先对最重要的四个工具作全盘性了解，再进去巡幽访胜一番。你总要先强记一下哪个工具做什么用，别把冯京当马凉，张飞战岳飞，往后的文字看起来才会顺畅。

图4-4 是MFC 程序的设计流程。

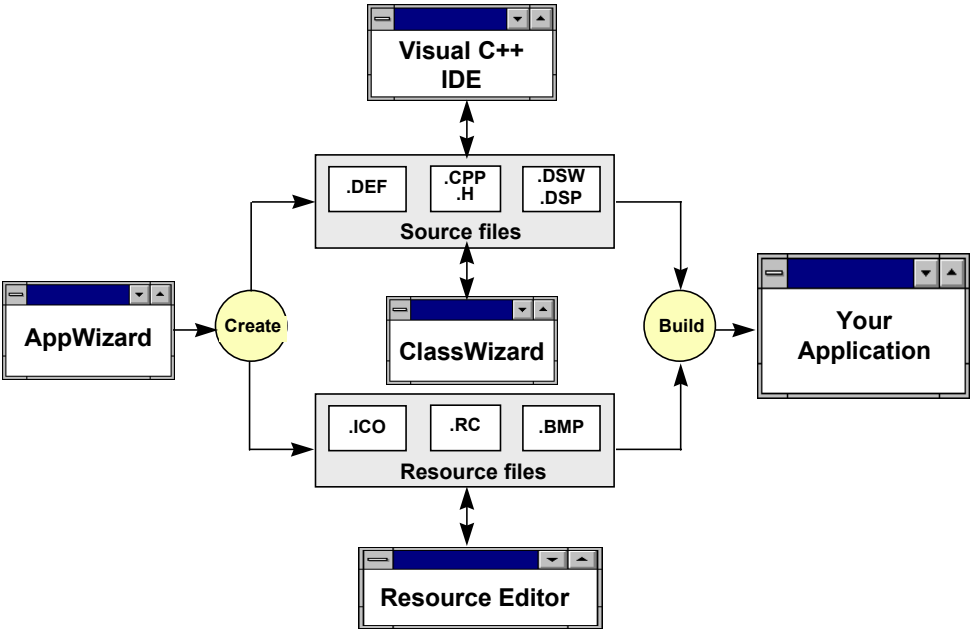


图4-4 MFC 程序的开发流程

Visual C++ 整合开发环境 (IDE)：你可以从中明显地或隐喻地激活其它工具如AppWizard 和ClassWizard；你可以设定各种工具、编译并联结程序、激活除错器、激活文字编辑器、浏览类别阶层...

AppWizard：这是一个程序代码产生器。基于application framework 的观念，相同类型（或说风格）的MFC 程序一定具备相同的程序骨干，AppWizard 让你挑选菜色（利用鼠标圈圈选选），也为你把菜炒出来（产生各种必要文件）。别忘记，化学反应是不能够还原的，菜炒好了可不能反悔（只能加油添醋），所以下手前需三思-- 每一个project 使用AppWizard 的机会只有一次。

Resource Editor：这是一个总合资源编辑器，RC 档内的各种资源它统统都有办法处理。Resource Editor 做出来的各类资源与你的程序代码之间如何维系关系？譬如说对话框中的一个控制组件被按下后程序该有什么反应？这就要靠ClassWizard 搭起鹊桥。

ClassWizard：AppWizard 制作出来的程序骨干是「起手无悔」的，接下来你只能够在程序代码中加油添醋（最重要的工作是加上自己的成员变量并改写虚拟函数），或搭起消息与程序代码之间的鹊桥（建立Message Map），这全得仰仗ClassWizard。以一般文字编辑器直接修改程序代码当然也可以，但你的思维必须非常缜密才不会挂一漏万。本书第四篇，当我们逐渐发展一个实用程序，你就会看到ClassWizard 的好处。

内务府总管：Visual C++ 整合开发环境

做为一个总管，要处理的大小事务很多。本章并不是Visual C++ 的完整使用手册，并不做细部操作解说（完整手册可参考Online Help 中的*Visual C++ User's Guide*）。基本上，如果你一边看这些文字说明一边实际玩玩这些工具，马上会有深刻的印象。

以功能菜单来分类，大致上Visual C++ 整合环境有以下功能：

File - 在此开启或储存文件。文字文件开启于一个文字编辑器中，这个编辑器对程序的撰写饶有助益，因为不同类型的关键词会以不同颜色标示。如果你新开启的是一个project，AppWizard 就会暗自激活（稍后再述）。文件的打印与印表机的设定也在此。

Edit - 这里有传统的剪贴簿（clipboard）功能。文字编辑器的Find 和Replace 功能也放在这里。

View - 对目前正在编辑之文件的各种设定动作。例如记号（bookmark）的设定寻找与清除，关键词颜色的设定与否、特定行号的搜寻...等等。ClassWizard 可在此菜单中被激活。

Insert - 可以在目前的project 中插入新的classes、resources、ATL objects...

Project - 可以在此操作project，例如加入文件、改变编译器和联结器选项等等。

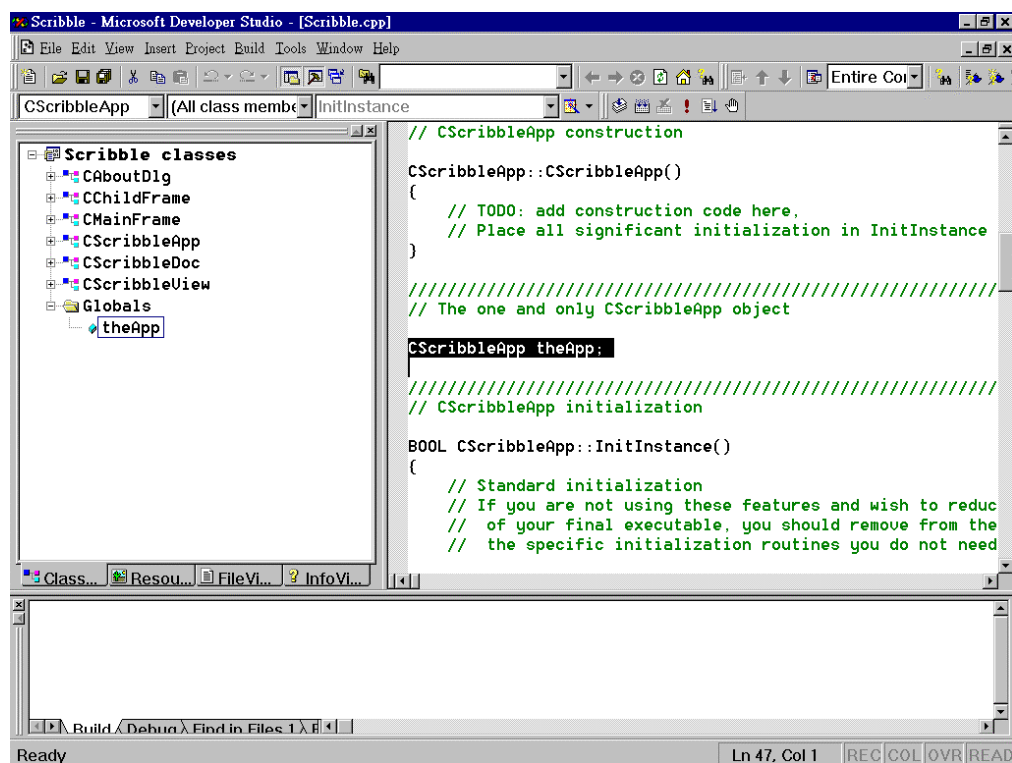
Build - 我们在这里制作出可执行档，也在这里除错。如果进入除错模式，Build 会变成Debug。

Tools - 可以激活Browser、MFC Tracer、SPY++ 以及其它工具。

Window - 整合环境（IDE）中各大大小小窗口可在此管理。

Help - 线上辅助说明，包括书籍、期刊、文章、范例。有一个不错的检索工具。

下面就是Visual C++ 整合环境（IDE）的画面：

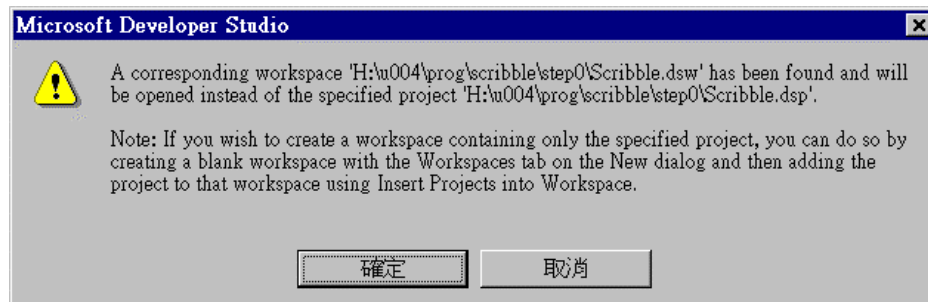


关于project

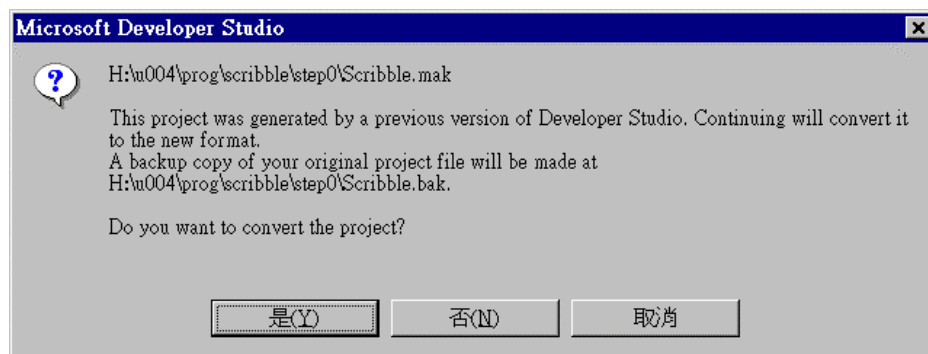
开发一个程序需要许多文件，这些文件以一个DSW 档和DSP 档（而不再是VC++ 4.x 时代的.MDP 档和.MAK 档）规范管理。一整组相关的文件就是一个project。只要你告诉Visual C++ 在哪个磁盘目录下开始一个新的project，它就会为你制作出一个DSW 档和一个DSP 档。假设我们的项目名称是"My"，那么就得到MY.DSP 和MY.DSW。下次你要继续工作时，在【File/Open】对话框中打开MY.DSW 就对了。

DSP 是Developer Studio Project 的缩写，DSW 是Developer Studio Workspace 的缩写。Workspace 是VC++ 整合环境（IDE）的一个维护档，可以把与该project 有关的IDE 环境设定都记录下来。所以，你应该在VC++ IDE 中选按【File/Open】后打开一个DSW 档（而不是DSP 档），以开启projects。如果你选择的是DSP 档，而同时存在着一个DSW

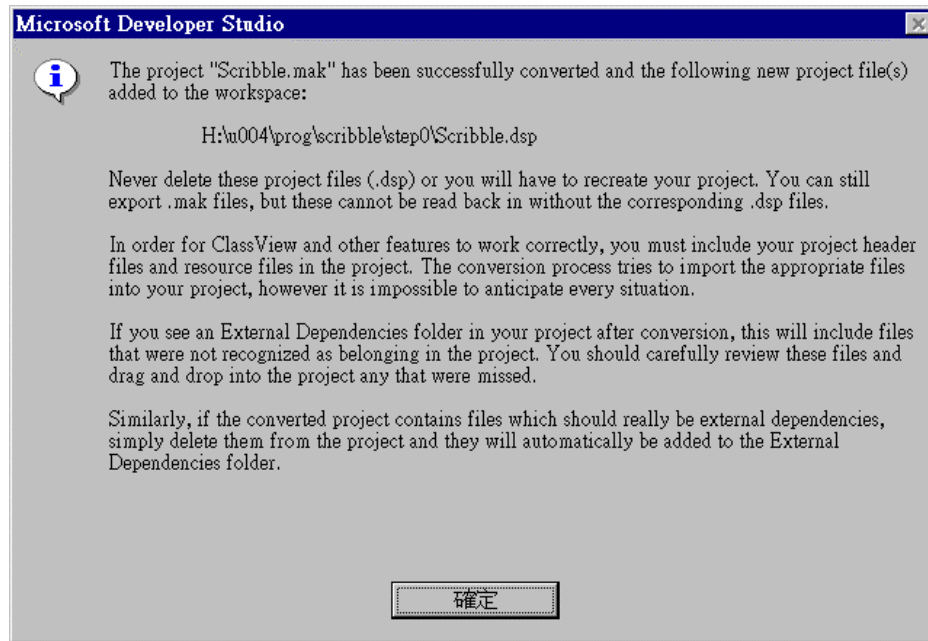
文件，你会获得这样的消息：



VC++ 4.x 的老用户们请注意，过去代表一个project 的所谓.MDP 档还存在吗？如果你是以VC++ 5.0 的wizards 来产生project，就不会再看到.MDP 档了，取而代之的是上述的.DSP 档和.DSW 档。如果你在VC++ 5.0 中开启过去在VC++ 4.x 时完成的project（.MDP 文件），会获得这样的消息：



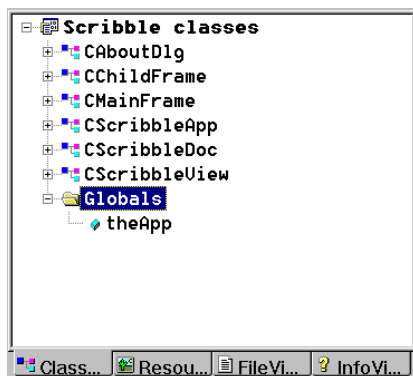
选择【是】之后，IDE 自动为你转换，并在完成之后给你这样的消息：



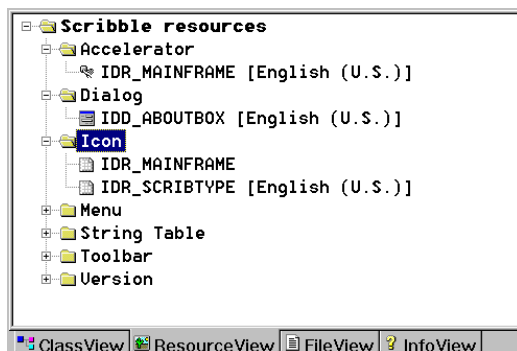
有趣的是，不论.MDP 档或.DSP 档或.DSW 档，我们的makefile 写作技巧势将逐渐萎缩。谁还会自己费心于那些!\$<<@# 等等诘屈聱牙的奇怪符号呢?! 这其实是件好事。

当你产生出一个project（利用AppWizard，稍后再提），整合环境提供了四个便利的管理窗口：

■ ClassView - 可以观察项目中所有的类别



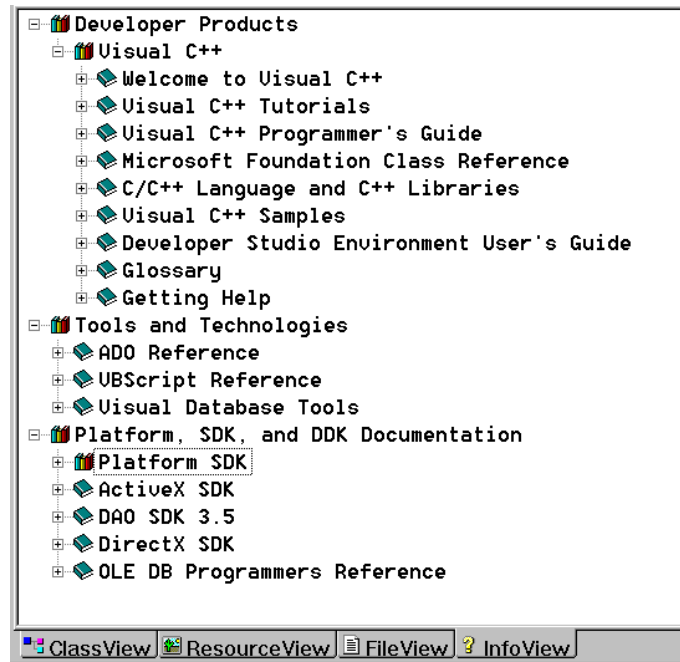
■ ResourceView - 可以观察项目中所有的资源



■ FileView - 可以观察项目中所有的文件



■ InfoView - Online Help 的总目录



关于工具设定

我们当然有机会设定编译器、联结器和RC 编译器的选项。图4-5 是两个设定画面。

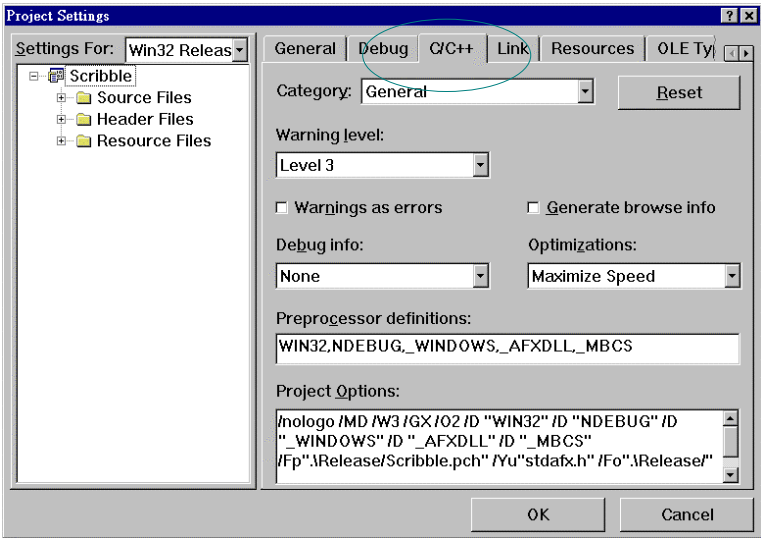


图4-5a 选择Visual C++ 的【Project/Setting...】，出现对话框。选择【C/C++】附页，于是可以设定编译器选项。

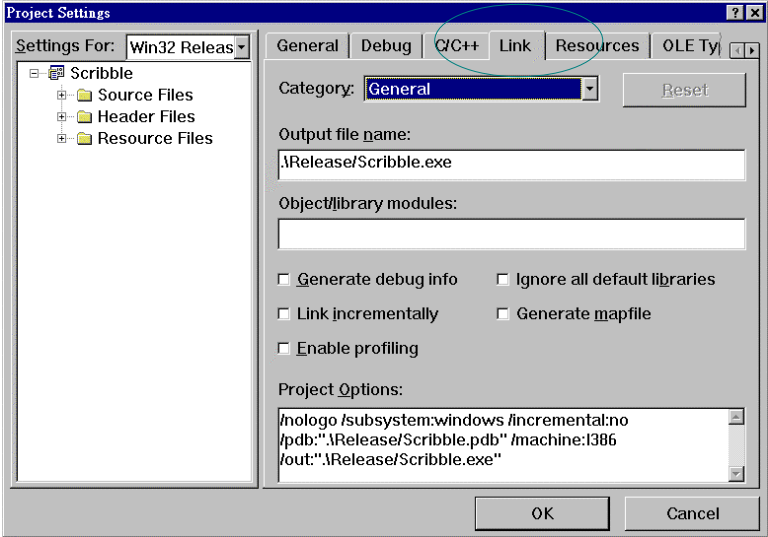


图4-5b 选择Visual C++ 的【Project/Setting...】，出现对话框。选择【Link】附页，于是可以设定联结器选项。

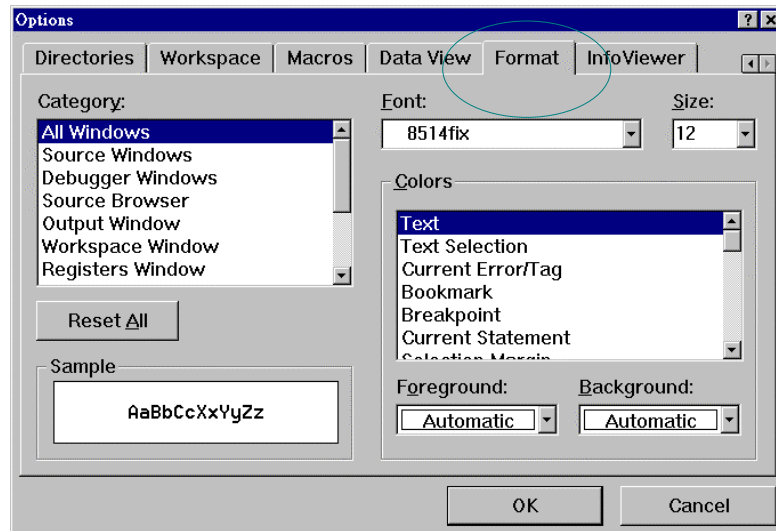
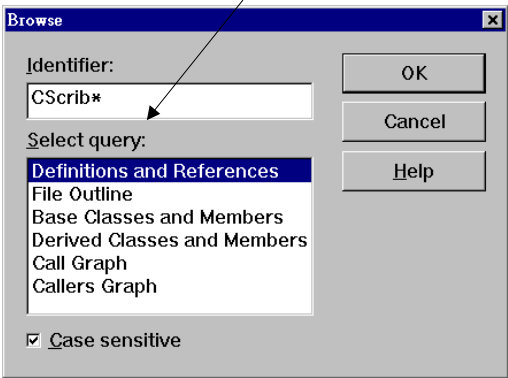


图4-5c 选择Visual C++ 的【Tools/Options...】，出现对话框。选择【Format】附页，于是可以设定程序代码编辑器的字型与大小....

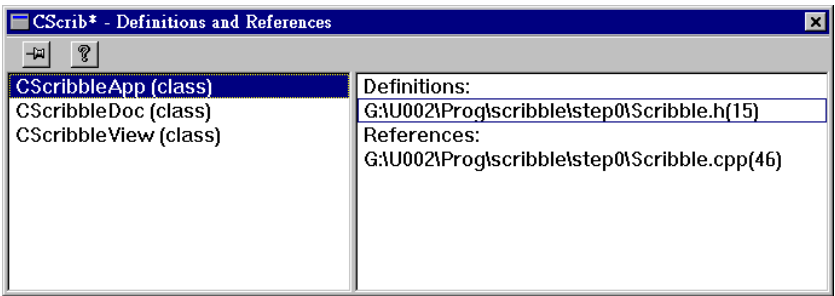
Source Browser

好的Browser（浏览器）是令人难以置信的一个有用工具，它把你快速带到任何你所指定的符号（symbol，包括类别、函数、变量、型别或宏）的出现地点。基本上Browser揭露两件事情：位置（places）和关系（relationship）。它可以显示某个符号「被定义」以及「被使用到」的任何位置。下面就显示名为CScrib*的所有类别：

选按Visual C++ 之【Tools/Source Browse...】菜单，出现以下对话框。在【Identifier】字段键入“CScrib*”，并在【Select Query】清单中选择【Definitions and References】：

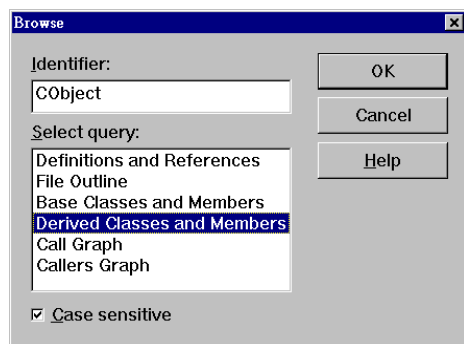


于是激活Browser，列出所有名为CScrib* 之类别。选择其中的CScribbleApp，右框之中就会填入所有它出现的位置（包括定义处以及被参考之处）。双击其中之一，你立刻置身其中，文字编辑器会跳出来，加载此档，准备为你服务。

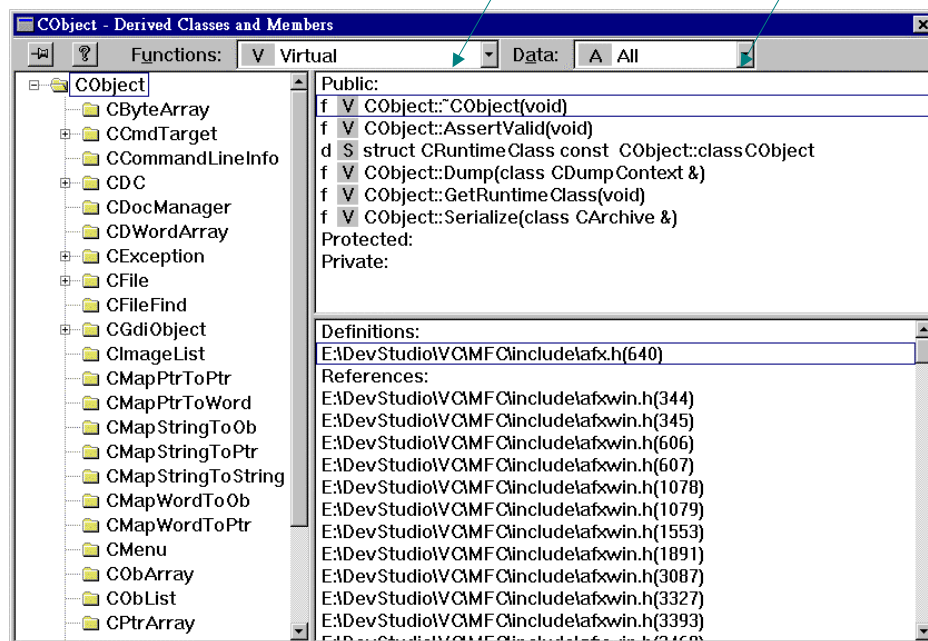


Browser 也揭露类别之间的关系以及类别与函数之间的关系。MFC 类别彼此叠床架屋，只以一般的文字编辑器（或如grep 之类的文字搜寻器）探索这些关系，就好象划一艘小船横渡太平洋到美利坚一样地缓慢而遥远。Browser 使我们在跋涉类别丛林时节省许多光阴。以下显示应用程序中所有衍生自CObject 的类别。

观察应用程序中所有衍生自CObject 的类别。请选按Visual C++ 之【Tools/Source Browse...】菜单，出现对话框。在【Identifier】字段键入“CObject”，请注意我选择的【Select Query】清单项目是【Derived Classes and Members】。

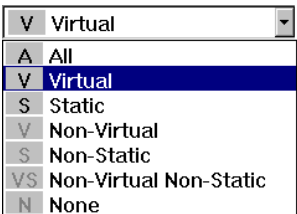


于是获得CObject 的所有衍生类别。请注意【Functions】栏是Virtual，【Data】栏是All。

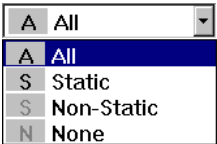


此时Browser 出现三个窗框，左边那个不论外观或行为都像文件总管里头的目录树，右边两个窗框显示你所选定之类别的详细信息。

Browser 提供这些【Functions】项目供观察：



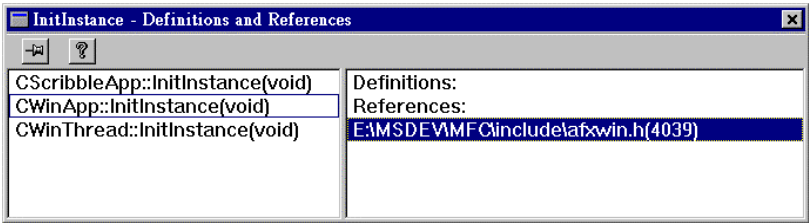
Browser 提供这些【Data】项目供观察：



Browser 系从一个特殊的数据库文件（.BSC）取得信息，此文件由Visual C++ 整合环境自动产生，非常巨大。如果暂时你不想要这个数据库，可以把图4-5a 中的【Generate browse Info】选项清除掉。而当你需要它时，选择【Tools/Source Browse...】，整合环境就会问你是否要建立.BSC 档。

提供给Browser 的资料（.BSC）很类似除错资料，两者都包含程序的符号信息。不同的是，除错资料附含在EXE 文件中，Browser 所需资料则独立于.BSC 档，不会增加EXE 文件大小（但会增加程序建造过程所需的时间）。

现在我打算观察InitInstance 函数。我在Browse 对话框中键入InitInstance 并选择【Definitions and References】，于是出现如下画面。双击其中的CWinApp::InitInstance，右框显示此函数原始定义于e:\msdev\mfc\include\afxwin.h #4039 行；再双击之，编辑器于是加载此档。以此方式观察MFC 源代码十分方便。



Online Help

我不是一个喜欢电子书的人，但是拿VC++ 这个Help 系统做快速查阅工作实在是不错。图4-6 是其使用画面与解说。

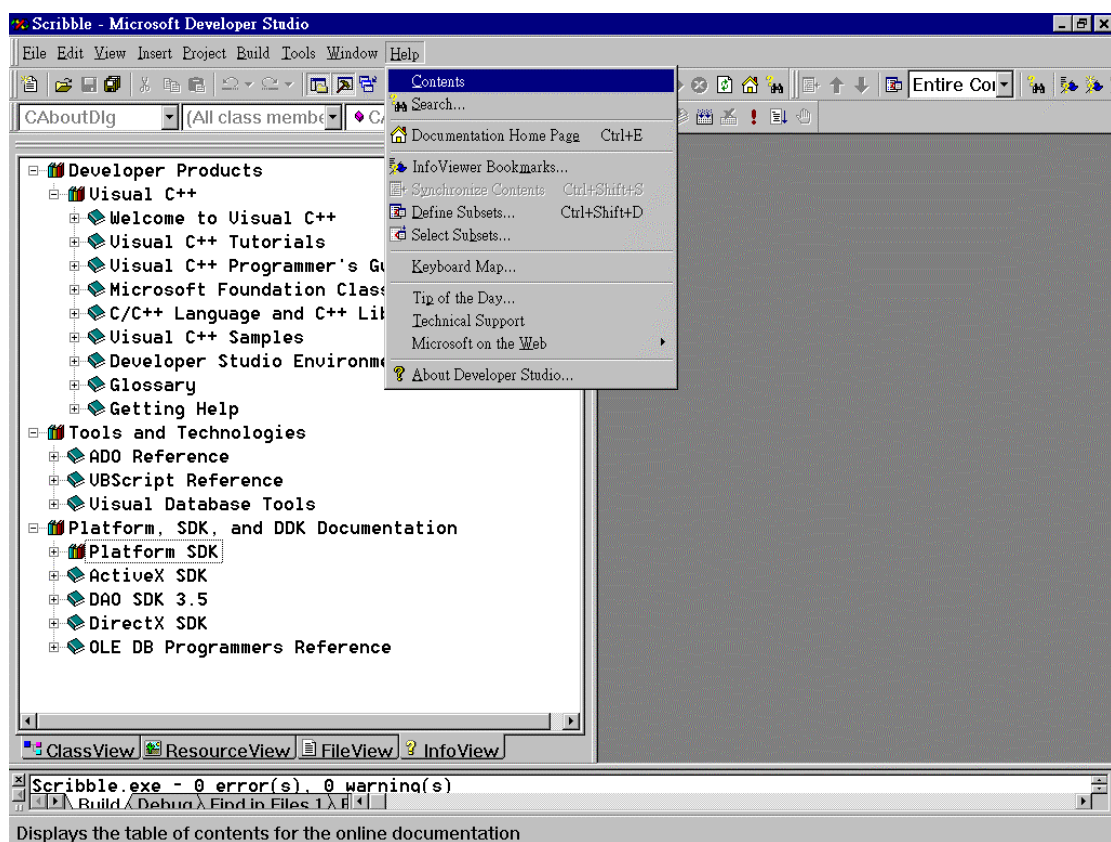


图4-6a VC++的Online Help 提供各种技术资料。按下【Help/Content】，就出现图左的数据清单，这也就是从Visual C++ 4.0 开始新增的所谓InfoView 窗口。Online Help 内容非常丰富。

让我们试试检索功能。选择【Help/Search】，出现对话框，键入CreateThread，出现数篇与此关键字有关的文章。选择某一篇文章，文章内容将出现在另一个窗口中。

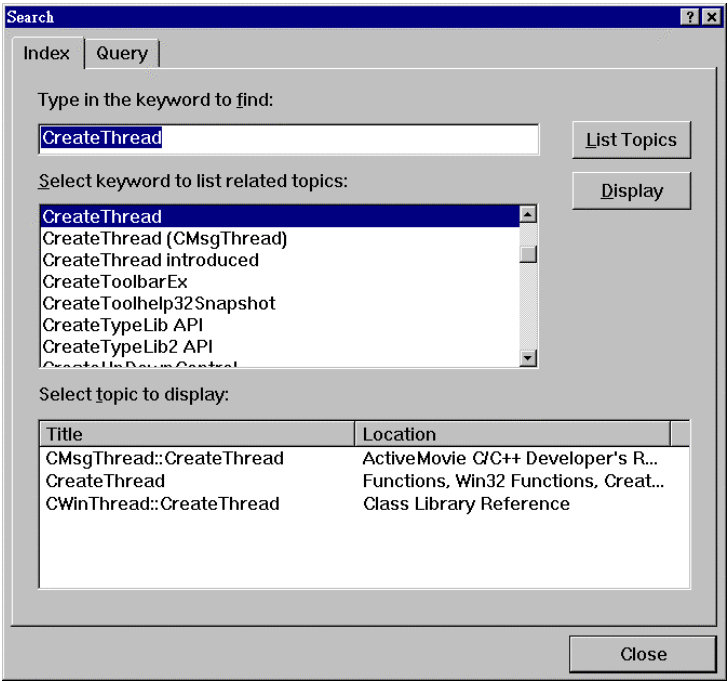


图4-6b 检索功能

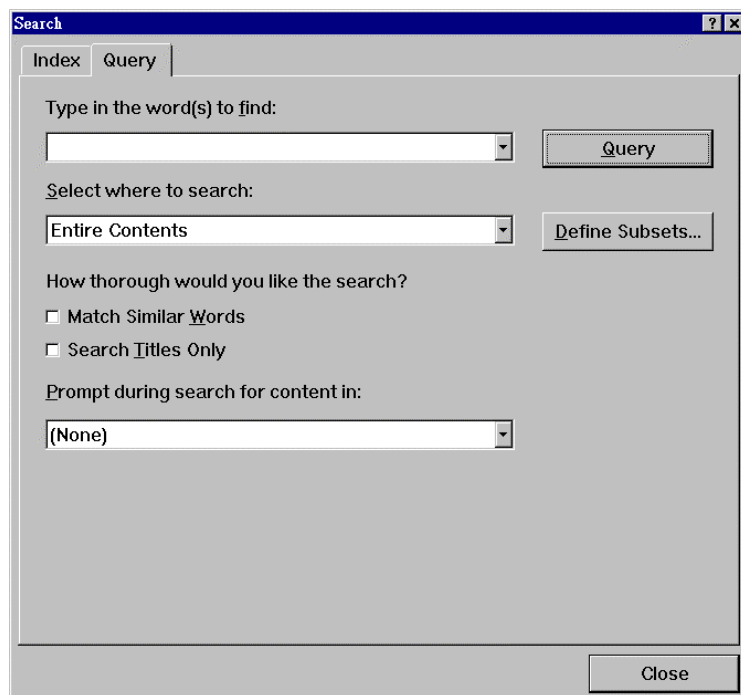


图4-6c 检索功能【Search】对话框的另一个附页。允许你做更多搜寻设定。

除错工具

每一位C程序员在DOS环境下都有使用「夹杀法」的除错经验：把可能错误的范围不断缩小，再缩小，最后以`printf`印出你心中的嫌疑犯，真象大白。

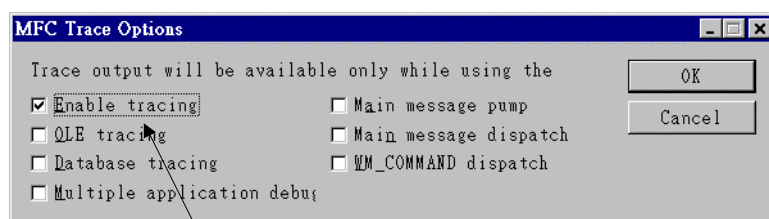
Windows程序员就没有方便的`printf`可用，唯`MessageBox`差可比拟。我曾经在Windows内存管理系统篇（旗标/1993）第0章介绍过一种以`MessageBox`和`NotePad.exe`合作仿真`printf`的方法，使用上堪称便利。

`MessageBox`会影响你的程序进行，自制`printf`又多费手脚。现在有了第三方案。你可以在程序的任何地方放置`TRACE`宏，例如：

```
TRACE("Hello World");
```

参数字符串将被输出到除错窗口去，不会影响你的程序进行。注意，*TRACE* 宏只对程序的除错版才有效，而且程序必须在 Visual C++ 的除错器中执行。

为了让 *TRACE* 生效，你还必须先另一个程序中做另一个动作。请选按【Tools / MFC Tracer】，得到这样的画面：



我们必须将【Enable Tracing】项目设立起来，然后除错窗口才能显示 *TRACE* 字符串。

旧版的 Visual C++ 中（v2.0 和 v1.5），*TRACE* 宏将字符串输出到一个名为 DBWin 的程序中。虽然应用程序必须以“Win32 debug”编译完成，但却不需要进入除错器就可以获得 *TRACE* 输出。从 Visual C++ 4.0 开始到 Visual C++ 5.0，不再附有 DBWin 程序，你无论如何需要大家伙（除错器）。如果你很怀念过去的好时光，请参考 Microsoft Systems Journal 上的三篇文章：1995/10 的 C++ Q/A，1996/01 的 C++ Q/A，以及 1997/04 的 C/C++ Q/A。这三篇文章都由 Paul Dilascia 执笔，教导读者如何自己动手做一个可接收 *TRACE* 巨集输出的 DBWIN 程序。我将在本书附录 D 中对 Paul Dilascia 的创意提供一些说明。

TRACE 很好用，美中不足的是它和 *MessageBox* 一样，只能输出字符串。这里有一个变通办法，把字符串和数值都送到 *afxDump* 变量去：

```
afxDump << "Hello World " << i << endl; // i 是整数变量
```

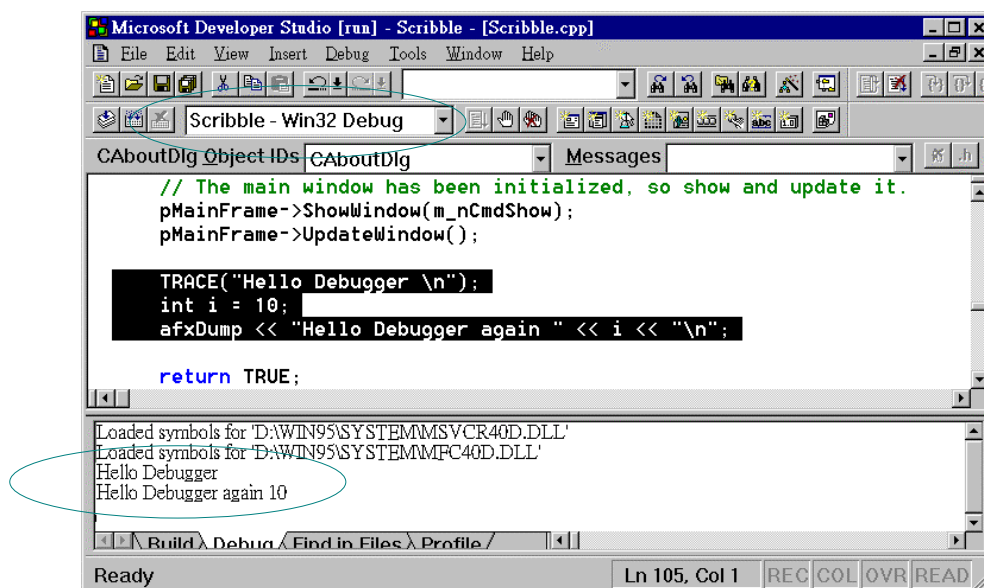
这是在 Visual C++ 中倾印（dump）一个对象内容的标准方法。它的输出也是流向除错窗口，所以你必须确定你的程序是除错版。

其实，要在应用程序中决定自己是不是除错版也很简单。若程序是以除错模式建造，`_DEBUG` 变量就成为`TRUE`，因此这样就可以判断一切了：

```
#ifdef _DEBUG

afxDump << "Hello World" << i << "\n"; // i 是整数变量

#endif
```



图上方的三进程序代码导致图下方除错窗口中的输出。

VC++ 除错器

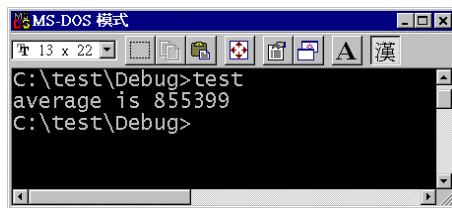
Visual C++ 整合环境内含一个精巧的除错器。这个除错器让我们很方便设定断点（程式执行至此会暂停）、观察变量内容、缓存器内容、并允许在除错过程中改变变量的值。我将以一个实际的猎虫行动示范如何使用除错器。

欲使用除错器，首先你的程序必须含有除错符号，也就是说，这必须是个除错版。很简单，只要在Visual C++ 整合环境上方选择【Win32 Debug】模式，然后再进行建造

(building) 工作，即可获得除错版本。现在假设我有一个程序，要计算五个学生的平均成绩：

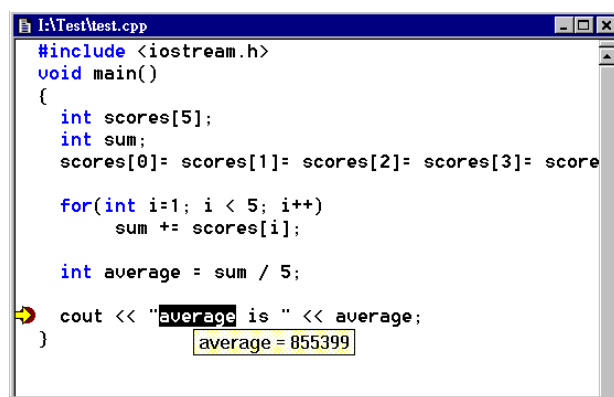
```
#0001 #include <iostream.h>
#0002 void main()
#0003 {
#0004     int scores[5];
#0005     int sum;
#0006     scores[0]= scores[1]= scores[2]= scores[3]= scores[4]= 60;
#0007
#0008     for(int i=1; i < 5; i++)
#0009         sum += scores[i];
#0010
#0011     int average = sum / 5;
#0012
#0013     cout << "average is " << average;
#0014 }
```

我预期的结果是`average` 等于60，而得到的结果却是：



为了把臭虫找出来，必须控制程序的进行，也就是设定断点 (breakpoint)；程序暂停之际，我就可以观察各个有嫌疑的变量。设定断点的方法是：把光标移到目的行，按下工具栏上的手形按钮 (或F9)，于是该行前面出现红点，表示断点设立。F9 是一个切换开关，在同一行再按一次F9 就能够清除断点。

为让断点生效，我必须以【Build/Debug/Go】(或F5) 执进程序，此时整合环境上的【Build】菜单变成了【Debug】。程序执行至断点即停下来，`average` 此刻的值应该是60。为证实此点，把光标放在`average` 上，出现一个小黄卷标：



```

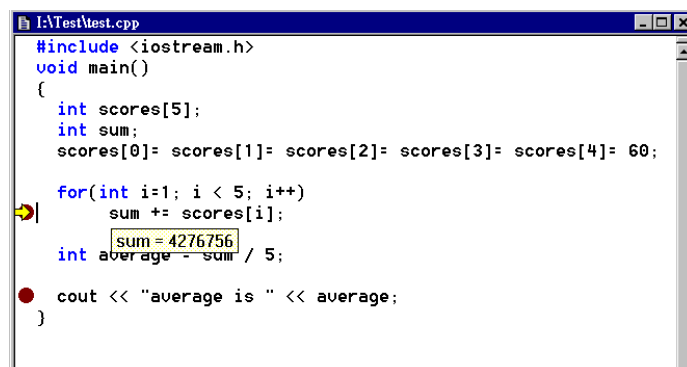
I:\Test\test.cpp
#include <iostream.h>
void main()
{
    int scores[5];
    int sum;
    scores[0]= scores[1]= scores[2]= scores[3]= score

    for(int i=1; i < 5; i++)
        sum += scores[i];

    int average = sum / 5;
    cout << "average is " << average;
}
    average = 855399

```

结果令人大吃一惊。显然我们有必要另设断点，观察其它变量。先以【Debug/Stop Debugging】结束除错状态，然后把断点设在`sum += scores[i]`这一行，重新"Go"下去。程序暂停时观察`sum`的值：



```

I:\Test\test.cpp
#include <iostream.h>
void main()
{
    int scores[5];
    int sum;
    scores[0]= scores[1]= scores[2]= scores[3]= scores[4]= 60;

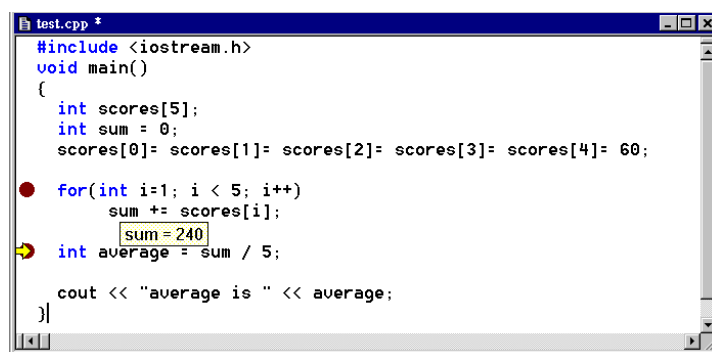
    for(int i=1; i < 5; i++)
        sum += scores[i];
    int average = sum / 5;
    cout << "average is " << average;
}
    sum = 4276756

```

此时此刻程序尚未执行任何一个加法，`sum`应该是0，但结果未符预期。显然，`sum`的初值没有设为0，我们抓到臭虫了。现在把程序代码第5行改为

```
int sum = 0;
```

重新建造，再"Go"一次。五次循环之后我们预期`sum`的值是300，结果却是240：



```

test.cpp *
#include <iostream.h>
void main()
{
    int scores[5];
    int sum = 0;
    scores[0]= scores[1]= scores[2]= scores[3]= scores[4]= 60;

    for(int i=1; i < 5; i++)
        sum += scores[i];
    int average = sum / 5;

    cout << "average is " << average;
}
    
```

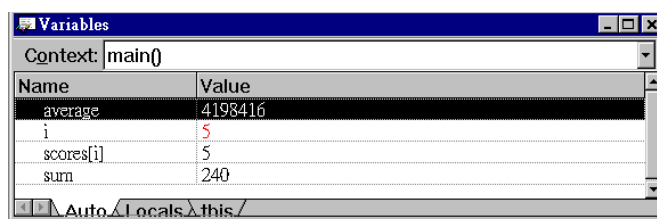
原来我竟把数组索引值*i* 从1 开始计算而不是从0 开始。

臭虫全部抓出来了，程序修正如下：

```

#0001  #include <iostream.h>
#0002  void main()
#0003  {
#0004      int scores[5];
#0005      int sum = 0;
#0006      scores[0]= scores[1]= scores[2]= scores[3]= scores[4]= 60;
#0007
#0008      for(int i=0; i < 5; i++)
#0009          sum += scores[i];
#0010
#0011      int average = sum / 5;
#0012
#0013      cout << "average is " << average;
#0014  }
    
```

除错过程中，你也可以选按【View/Variables】打开Variables 窗口，所有的变量值更能够一目了然：



Variables	
Context: main()	
Name	Value
average	4198416
i	5
scores[i]	5
sum	240

除了除错之外，我还常常以除错器追踪MFC 程序，以期深入了解MFC 各类别。你知道，数万行MFC 源代码光靠Step Into/Step Over/Call Stack 这几招，便能迅速切中「要害」。

小技巧：当你要找出与窗口painting 有关的臭虫时，尽量不要把欲除错之程序窗口与 Visual C++ IDE 窗口覆叠在一起，才不会互相影响。当然，最好你有一个17 寸屏幕和 1024*768 的分辨率。21 寸屏幕？呕，小心你的荷包。

Exception Handling

第2 章最后面我曾简介过 C++ exception handling。这里我要再举一个很容易练习的MFC exception handling 实例。

开档是一件可能产生许多exception 的动作。文件开启不成功，可能是因为文件找不到，或是磁盘空间不足，或是路径不对，或是违反文件共享原则（sharing violation），或是超出了可开档数目限制。你可以在任何一个程序中练习下面这一段码。不需要除错版，基本上exception handling 与除错模式并无瓜葛。下列程序代码中的*Output* 函数只是个代名，并不是真有这样的API 函数，你可以改用*MessageBox*、*TextOut*、*TRACE* 等任何有字符串输出能力的函数。

```
#001 CString str = "Hello World";
#002
#003 TRY {
#004     CFile file("a:hello.txt", CFile::modeCreate | CFile::modeWrite);
#005     file.Write(str, str.GetLength());
#006     file.Close();
#007 }
#008 CATCH(CFileException, e) {
#009     switch(e->m_cause) {
#010         case CFileException::accessDenied :
#011             Output("File Access Denied");
#012             break;
#013         case CFileException::badPath :
#014             Output("Invalid Path");
#015             break;
```

```
#016     case CFileException::diskFull :
#017         Output("Disk Full");
#018         break;
#019     case CFileException::fileNotFound :
#020         Output("File Not Found");
#021         break;
#022     case CFileException::hardIO :
#023         Output("Hardware Error");
#024         break;
#025     case CFileException::lockViolation :
#026         Output("Attemp to lock region already locked");
#027         break;
#028     case CFileException::sharingViolation :
#029         Output("Sharing Violation - load share.exe");
#030         break;
#031     case CFileException::tooManyOpenFiles :
#032         Output("Too Many Open Files");
#033         break;
#034     }
#035 }
```

让我简单地做一个说明。*TRY* 区块中的动作（本例为开档、写档、关档）如果在执行时期有任何exception 发生，就会跳到*CATCH* 区块中执行。*CATCH* 的第一个参数是 exception type：如果是文件方面的exception，就是*CFileException*，如果是内存方面的exception，那么就是*CMemoryException*。*CATCH* 的第二个参数是一个对象，经由其资料成员*m_cause*，我们可以获知exception 的发生原因。这些原因（如accessDenied, badPath, diskFull, FileNoteFound...）都定义于AFX.H 中。

程序代码产生器：AppWizard

有一个Generic 范例程序，号称为「Windows 程序之母」，恐怕大家都是从那里跨出 Windows 程序设计的第一步。过去，当我要开始一个新的project，我就把Generic 的所有文件拷贝到新的子目录下，然后改变文件名，然后把makefile 中所有的"GENERIC" 字符串改为新project 名称字符串。我还必须改变C 文件中的窗口类别名称、窗口标题、菜单名称、对话框名称；我必须改变RC 文件中的菜单、对话框等资源；我也得改变DEF 档中的模块名称和DESCRIPTION 叙述句。这些琐碎的事做完，我才开始在DEF、C、RC 档中添骨添肉。

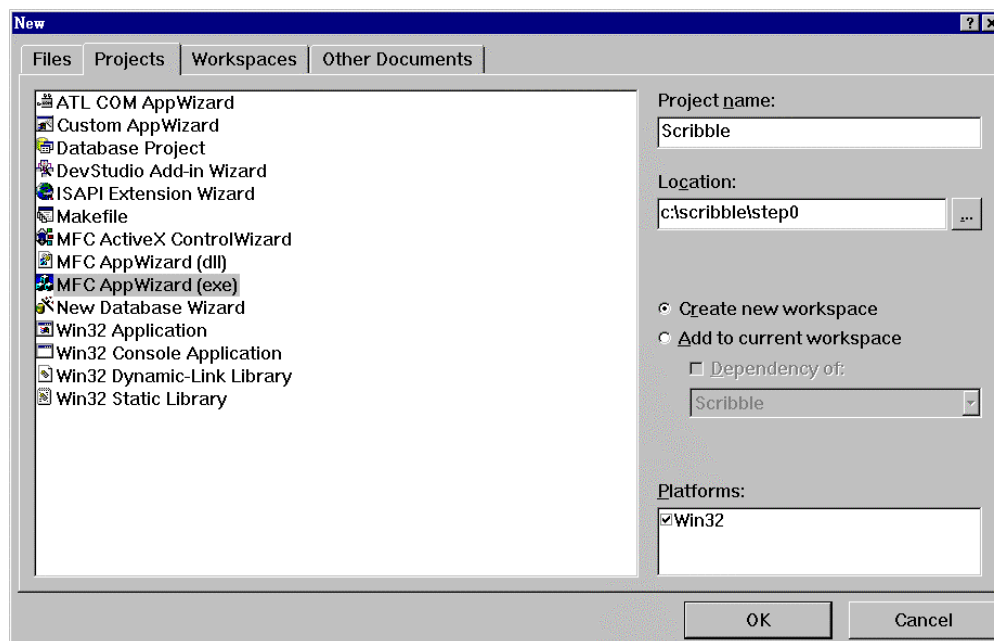
数以打计的小步骤要做!!

有了AppWizard，这些沉闷而令人生厌的琐碎工作都将自动化起来。不止如此，AppWizard 可以为我们做出一致化的骨干程序出来。以此种方式应付（我的意思是产生）标准接口十分合适。

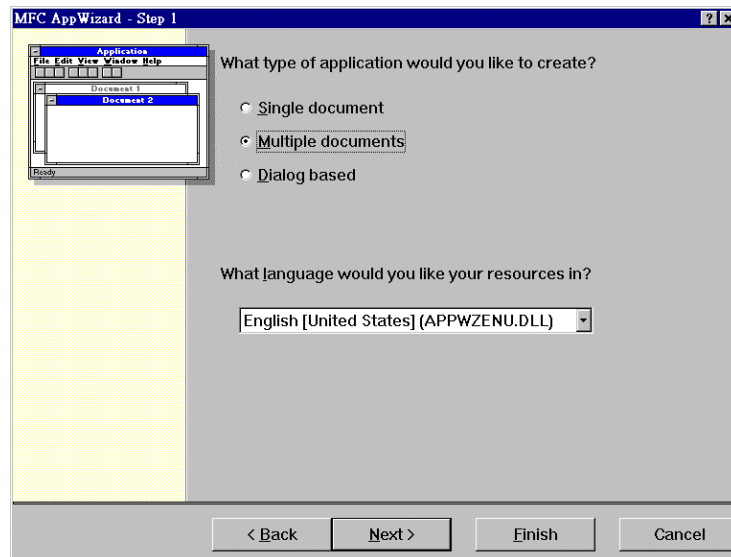
你可以从Visual C++ 整合环境中激活AppWizard。第一次使用时机是当你要展开一个新的project 之时。首先，为project 命名并为它找一个栖身场所（一个磁盘目录），然后选择你想要的程序风格（例如SDI 或MDI）。问答题作完，劈哩啪啦呼噜哗啦，AppWizard 很快为你产生一个骨干程序。这是一个完整的，不需增减任何一行码就可编译执行的程式，虽然它什么大事儿都没做，却保证令你印象深刻。外观（使用者接口）十分华丽，Win32 程序员穷数星期之心力也不见得做得出这样漂亮丰富的接口来。

东圈西点完成 MFC 程序骨干

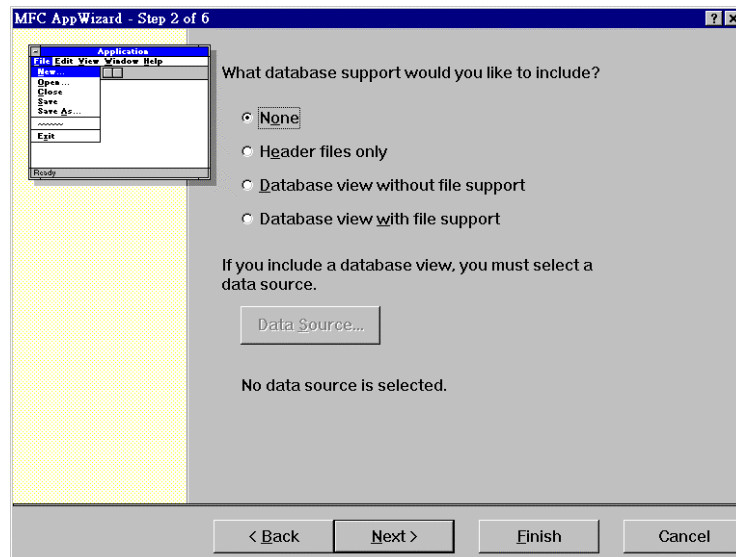
选按【File/New】，并在【New】对话框中选择【Project】附页。然后再在其中选择MFC Application (exe)，于是准备进入AppWizard 建立"Scribble" project。右边的磁盘目录和project 名称亦需填妥。



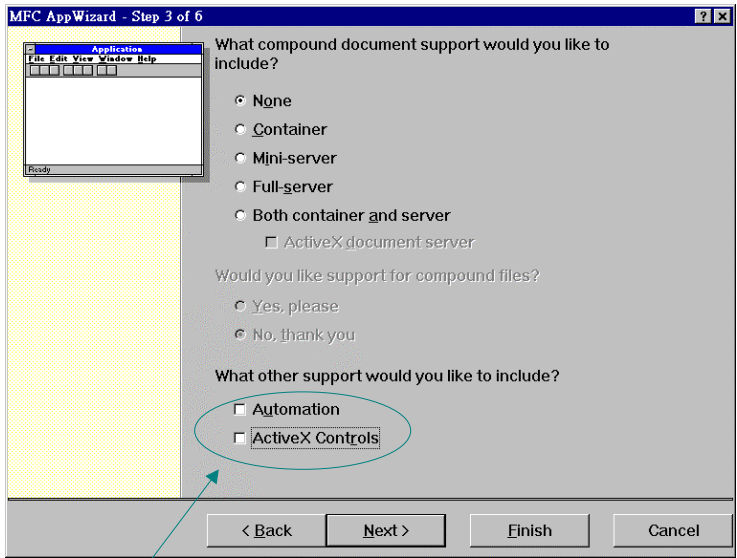
MFC AppWizard 步骤一，选择SDI 或MDI 或Dialog-based 程序风格。预设情况是MDI。



MFC AppWizard 步骤二，选择是否需要数据库支持。预设情况是None。

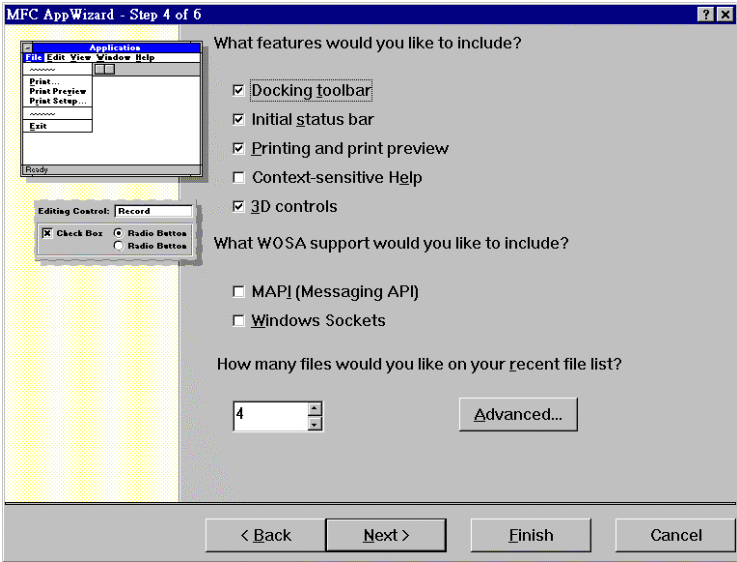


MFC AppWizard 步骤三，选择是否需要compound document 和ActiveX 支持。预设情况下支持ActiveX Controls，本例为求简化，将它关闭。

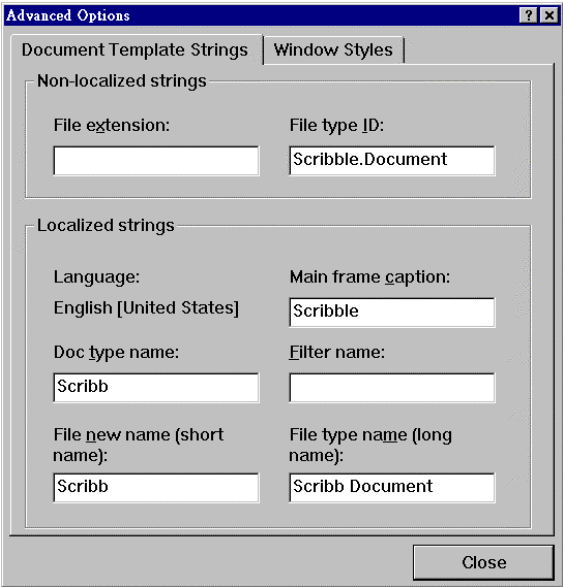


注意，在VC++ 4.x 版中此处为OLE Automation 和OLE controls。

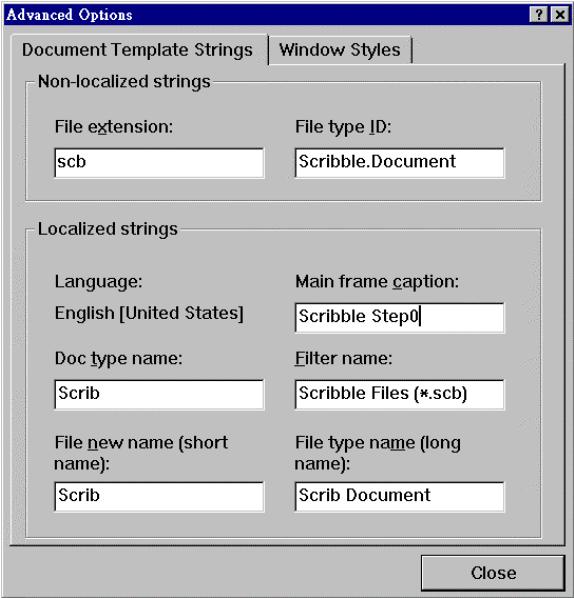
MFC AppWizard 步骤四，选择使用者接口。预设情况下【Context Sensitive Help】未设立。



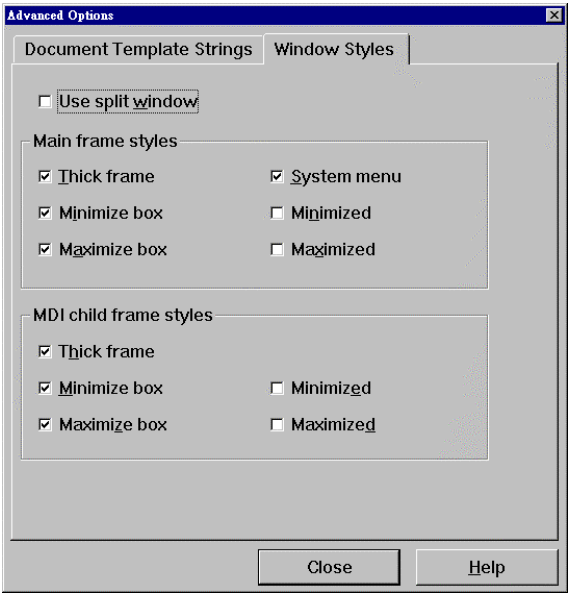
MFC AppWizard 步骤四的【Advanced】带出【Advanced Options】对话框：



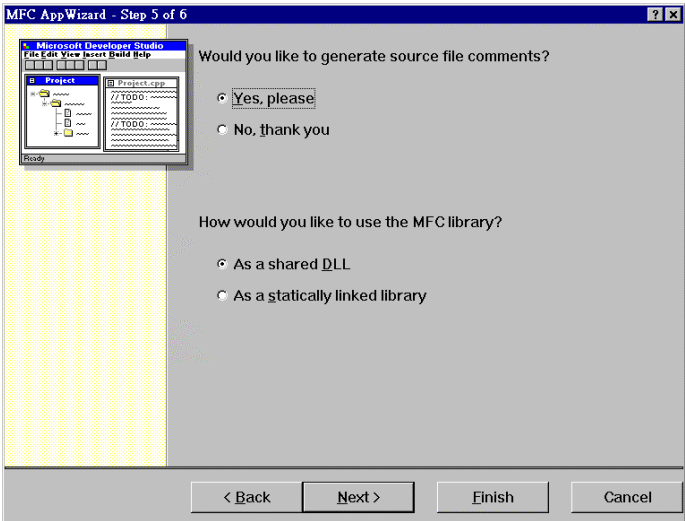
把上图修改为下面这个样子。这些修改对程序代码带来的变化，将在第 7 章中说明。



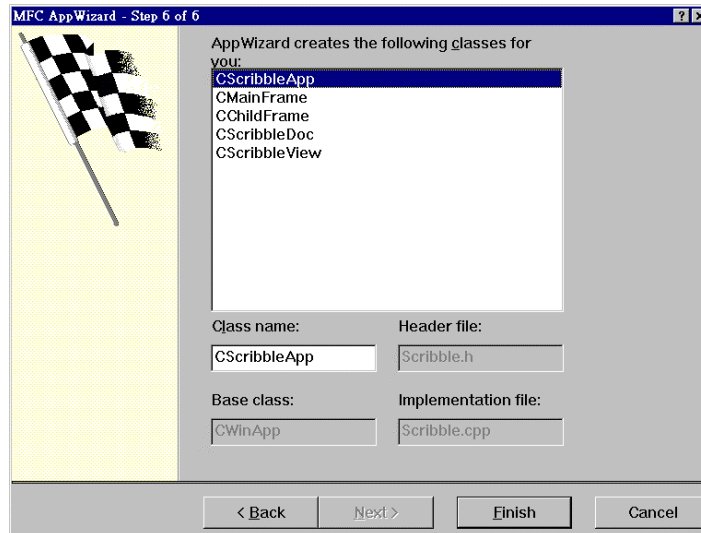
MFC AppWizard 【Advanced Options】的另一附页。其中最上面的一个核示钮【Use split window】预设是关闭状态。如果要制作分裂窗口（如本书第11 章），把它打开就是了。



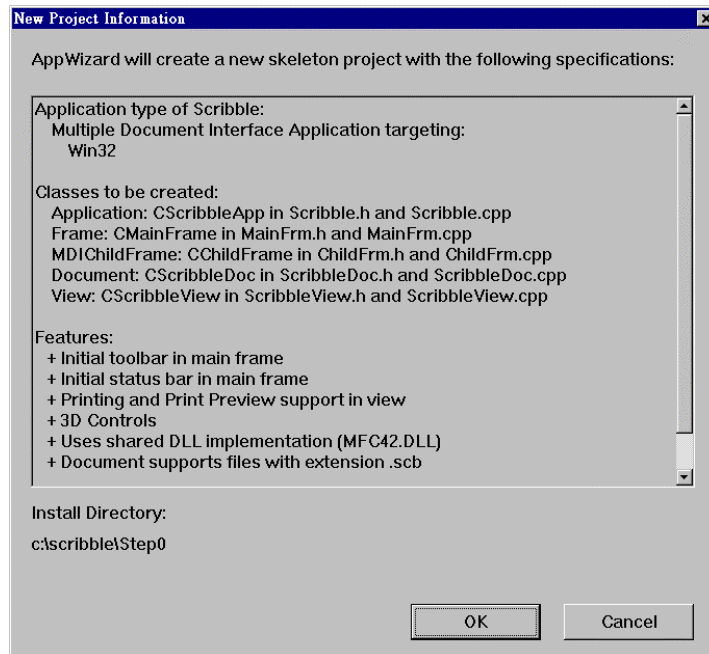
MFC AppWizard 步骤五，提供另一些选项，询问要不要为你的源代码产生一些说明文字。并询问你希望使用的MFC 版本（动态联结版或静态联结版）。



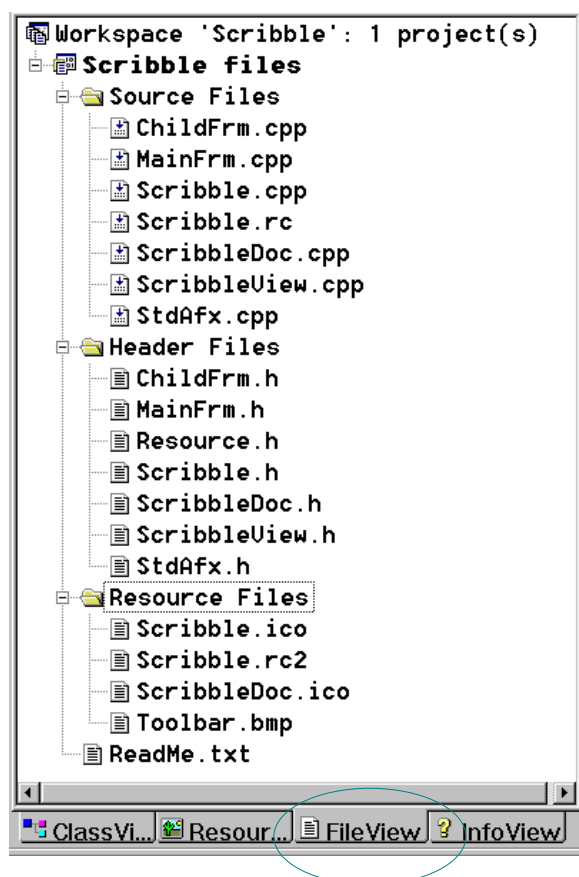
MFC AppWizard 步骤六（最后一步），允许你更改档名或类别名称。完成后按下【Finish】



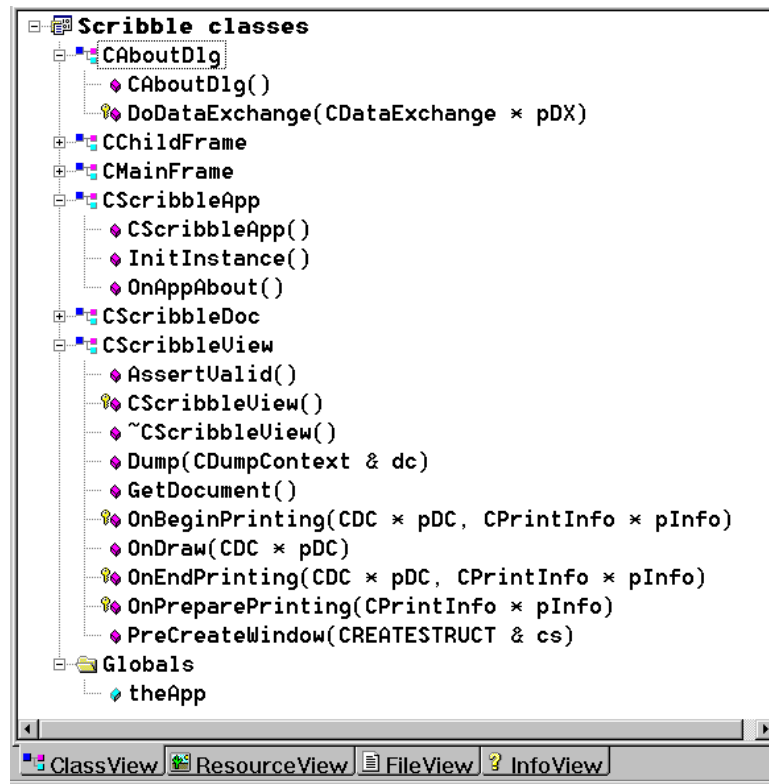
MFC AppWizard 获得的清单（包括文件和类别）：



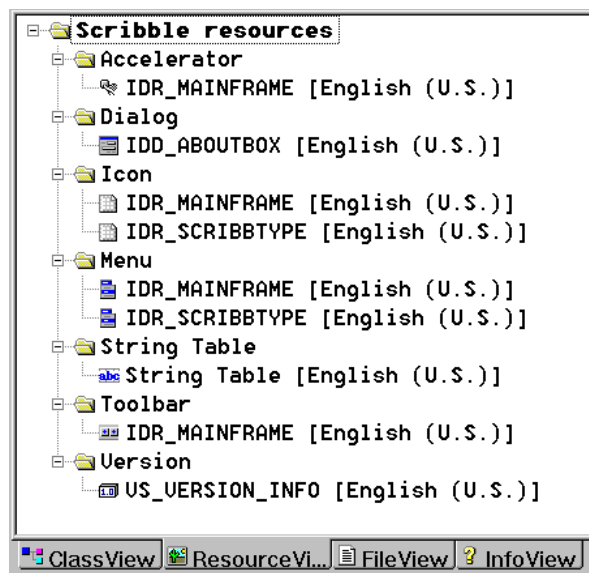
一行程序代码都还没写，就获得了这么多文件。你可以选择【Win32 Debug】或【Win32 Release】来建造（building）程序，获得的二进制文件将放在不同的子目录中。



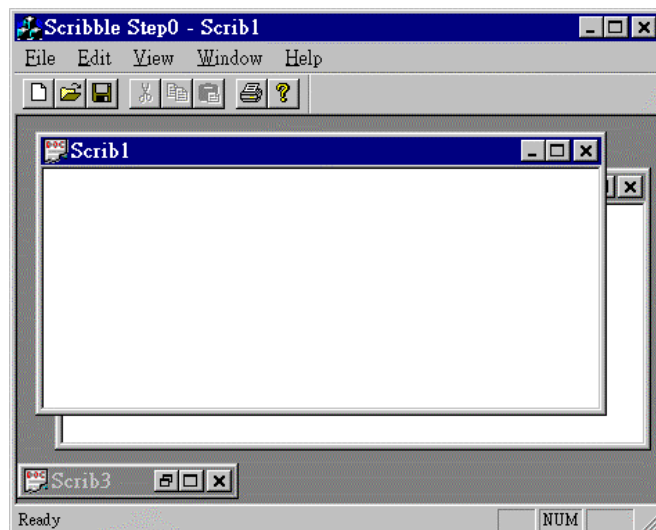
一行程序代码都还没写，就获得了这么多类别。



一行程序代码都还没写，就获得了这么多资源。



一行程序代码都没写，只是点点按按，我们就获得了一个令人惊艳的程序。基本功能一应俱全（文件对话框、打印机设定、Help、工具栏、状态列...），却什么也不能做（那是当然）。



AppWizard 总是为一般的应用程序产生五个类别。我所谓的「一般程序」是指non-OLE 以及non-ODBC 程序。针对上述的Scribble 程序，它产生的类别列于图4-7。

类别名称	基础类别	类别声明于	类别定义于
<i>CScribbleApp</i>	<i>CWinApp</i>	Scribble.h	Scribble.cpp
<i>CMainFrame</i>	<i>CMDIFrameWnd</i>	Mainfrm.h	Mainfrm.cpp
<i>CChildFrame</i>	<i>CMDIChildWnd</i>	Childfrm.h	Childfrm.cpp
<i>CScribbleDoc</i>	<i>CDocument</i>	ScribbleDoc.h	ScribbleDoc.cpp
<i>CScribbleView</i>	<i>CView</i>	ScribbleView.h	ScribbleView.cpp

图4-7 Scribble Step0 (骨干程序) 中，各个类别的相关资料。事实上Scribble 程序中用到了 9 个类别，不过只有上述 5 个类别需要改写 (override)。

你最好把哪一个类别衍生自哪一个MFC 类别弄清楚，并搞懂AppWizard 的命名规则。大致上命名规则是这样的：

```
'C' + ProjectName + Classtype = Class Name
```

所有的类别名称都由AppWizard 自动命名，如果你喜欢，也可以在AppWizard 的步骤六改变之。这些类别名称可以很长很长 (Windows 95 与Windows NT 均支持长档名)。每个类别都对应一个.H (类别声明) 和一个.CPP (类别定义)。

AppWizard 十分周到地为我们产生了一个README.TXT，对各个文件都有解释 (图4-8)。从激活AppWizard 到建立Scribble.exe，如果你是熟手，机器又不慢的话，不需要一分钟。拿着码表算时间其实不具意义 (就像计算程序行数多寡一样地不具意义)，我要说的是它的确便利。

```
=====
      MICROSOFT FOUNDATION CLASS LIBRARY : Scribble
=====

AppWizard has created this Scribble application for you. This application
not only demonstrates the basics of using the Microsoft Foundation classes
but is also a starting point for writing your application.

This file contains a summary of what you will find in each of the files that
make up your Scribble application.

Scribble.h
    This is the main header file for the application. It includes other
    project specific headers (including Resource.h) and declares the
    CScribbleApp application class.

Scribble.cpp
    This is the main application source file that contains the application
    class CScribbleApp.

Scribble.rc
    This is a listing of all of the Microsoft Windows resources that the
    program uses. It includes the icons, bitmaps, and cursors that are stored
    in the RES subdirectory. This file can be directly edited in Microsoft
    Developer Studio.

res\Scribble.ico
    This is an icon file, which is used as the application's icon. This
    icon is included by the main resource file Scribble.rc.

res\Scribble.rc2
    This file contains resources that are not edited by Microsoft
    Developer Studio. You should place all resources not
    editable by the resource editor in this file.

Scribble.clw
    This file contains information used by ClassWizard to edit existing
    classes or add new classes. ClassWizard also uses this file to store
    information needed to create and edit message maps and dialog data
    maps and to create prototype member functions.

////////////////////////////////////

For the main frame window:
```

```

MainFrm.h, MainFrm.cpp
    These files contain the frame class CMainFrame, which is derived from
    CMDIFrameWnd and controls all MDI frame features.

res\Toolbar.bmp
    This bitmap file is used to create tiled images for the toolbar.
    The initial toolbar and status bar are constructed in the
    CMainFrame class. Edit this toolbar bitmap along with the
    array in MainFrm.cpp to add more toolbar buttons.

////////////////////////////////////

AppWizard creates one document type and one view:

ScribbleDoc.h, ScribbleDoc.cpp - the document
    These files contain your CScribbleDoc class. Edit these files to
    add your special document data and to implement file saving and loading
    (via CScribbleDoc::Serialize).

ScribbleView.h, ScribbleView.cpp - the view of the document
    These files contain your CScribbleView class.
    CScribbleView objects are used to view CScribbleDoc objects.

res\ScribbleDoc.ico
    This is an icon file, which is used as the icon for MDI child windows
    for the CScribbleDoc class. This icon is included by the main
    resource file Scribble.rc.

////////////////////////////////////

Other standard files:

StdAfx.h, StdAfx.cpp
    These files are used to build a precompiled header (PCH) file
    named Scribble.pch and a precompiled types file named StdAfx.obj.

Resource.h
    This is the standard header file, which defines new resource IDs.
    Microsoft Developer Studio reads and updates this file.

////////////////////////////////////

Other notes:

AppWizard uses "TODO:" to indicate parts of the source code you
should add to or customize.

```


If your application uses MFC in a shared DLL, and your application is in a language other than the operating system's current language, you will need to copy the corresponding localized resources MFC40XXX.DLL from the Microsoft Visual C++ CD-ROM onto the system or system32 directory, and rename it to be MFCLOC.DLL. ("XXX" stands for the language abbreviation. For example, MFC40DEU.DLL contains resources translated to German.) If you don't do this, some of the UI elements of your application will remain in the language of the operating system.

图4-8 Scribble 程序的readme.txt 档。

别忘了，AppWizard 产生的是化学反应而不是物理反应，是不能够还原的。我们很容易犯的错误是像进入糖果店里的小孩一样，每样东西都想要。你应该约束自己，因为错一步已是百年身，不能稍后又回到AppWizard 要求去掉或改变某些选项，例如想把SDI 改为MDI 或是想增加OLE 支持等等，都不能够。欲变更程序，只有两条路可走：要不就令AppWizard 重新产生一组新的程序骨干，然后回到原程序中打捞点什么可以用的，以Copy/Paste 方式移植过来；要不就是直接进入原来程序修修补补。至于修补过程中到底会多么令人厌烦，那就不一而论了。所以，在开始你的程序撰写之前，小心做好系统分析的工作。

Scribble 是第四篇程序的起点。我将在第四篇以每章一个主题的方式，为它加上新的功能。下面是Scribble step0 的源代码。

SCRIBBLE.H

```
#0001 // Scribble.h : main header file for the SCRIBBLE application
#0002 //
#0003
#0004 #ifndef __AFXWIN_H__
#0005 #error include 'stdafx.h' before including this file for PCH
#0006 #endif
#0007
#0008 #include "resource.h" // main symbols
#0009
#0010 //////////////////////////////////////
#0011 // CScribbleApp:
#0012 // See Scribble.cpp for the implementation of this class
#0013 //
```

```

#0014
#0015 class CScribbleApp : public CWinApp
#0016 {
#0017 public:
#0018     CScribbleApp();
#0019
#0020 // Overrides
#0021 // ClassWizard generated virtual function overrides
#0022 //{{AFX_VIRTUAL(CScribbleApp)
#0023 public:
#0024     virtual BOOL InitInstance();
#0025     //}}AFX_VIRTUAL
#0026
#0027 // Implementation
#0028
#0029 //{{AFX_MSG(CScribbleApp)
#0030     afx_msg void OnAppAbout();
#0031         // NOTE - the ClassWizard will add and remove member functions here.
#0032         //      DO NOT EDIT what you see in these blocks of generated code !
#0033     //}}AFX_MSG
#0034     DECLARE_MESSAGE_MAP()
#0035 };

```

MAINFRM.H

```

#0001 // MainFrm.h : interface of the CMainFrame class
#0002 //
#0003 //////////////////////////////////////
#0004
#0005 class CMainFrame : public CMDIFrameWnd
#0006 {
#0007     DECLARE_DYNAMIC(CMainFrame)
#0008 public:
#0009     CMainFrame();
#0010
#0011 // Attributes
#0012 public:
#0013
#0014 // Operations
#0015 public:
#0016
#0017 // Overrides
#0018 // ClassWizard generated virtual function overrides
#0019 //{{AFX_VIRTUAL(CMainFrame)
#0020     virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
#0021     //}}AFX_VIRTUAL

```

```
#0022
#0023 // Implementation
#0024 public:
#0025     virtual ~CMainFrame();
#0026 #ifdef _DEBUG
#0027     virtual void AssertValid() const;
#0028     virtual void Dump(CDumpContext& dc) const;
#0029 #endif
#0030
#0031 protected: // control bar embedded members
#0032     CStatusBar  m_wndStatusBar;
#0033     CToolBar    m_wndToolBar;
#0034
#0035 // Generated message map functions
#0036 protected:
#0037     //{AFX_MSG(CMainFrame)
#0038     afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
#0039     // NOTE - the ClassWizard will add and remove member functions here.
#0040     //     DO NOT EDIT what you see in these blocks of generated code!
#0041     //}AFX_MSG
#0042     DECLARE_MESSAGE_MAP()
#0043 };
```

CHILDFRM.H

```
#0001 // ChildFrm.h : interface of the CChildFrame class
#0002 //
#0003 //////////////////////////////////////
#0004
#0005 class CChildFrame : public CMDIChildWnd
#0006 {
#0007     DECLARE_DYNCREATE(CChildFrame)
#0008 public:
#0009     CChildFrame();
#0010
#0011 // Attributes
#0012 public:
#0013
#0014 // Operations
#0015 public:
#0016
#0017 // Overrides
#0018 // ClassWizard generated virtual function overrides
#0019 //{AFX_VIRTUAL(CChildFrame)
#0020     virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
#0021 //}AFX_VIRTUAL
```

```

#0022
#0023 // Implementation
#0024 public:
#0025     virtual ~CChildFrame();
#0026 #ifdef _DEBUG
#0027     virtual void AssertValid() const;
#0028     virtual void Dump(CDumpContext& dc) const;
#0029 #endif
#0030
#0031 // Generated message map functions
#0032 protected:
#0033     //{AFX_MSG(CChildFrame)
#0034         // NOTE - the ClassWizard will add and remove member functions here.
#0035         //      DO NOT EDIT what you see in these blocks of generated code!
#0036     //}AFX_MSG
#0037     DECLARE_MESSAGE_MAP()
#0038 };

```

SCRIBBLEDOC.H

```

#0001 // ScribbleDoc.h : interface of the CScribbleDoc class
#0002 //
#0003 //////////////////////////////////////
#0004
#0005 class CScribbleDoc : public CDocument
#0006 {
#0007     protected: // create from serialization only
#0008         CScribbleDoc();
#0009         DECLARE_DYNCREATE(CScribbleDoc)
#0010
#0011     // Attributes
#0012     public:
#0013
#0014     // Operations
#0015     public:
#0016
#0017     // Overrides
#0018     // ClassWizard generated virtual function overrides
#0019     //{AFX_VIRTUAL(CScribbleDoc)
#0020     public:
#0021         virtual BOOL OnNewDocument();
#0022         virtual void Serialize(CArchive& ar);
#0023     //}AFX_VIRTUAL
#0024
#0025     // Implementation
#0026     public:

```

```
#0027 virtual ~CScribbleDoc();
#0028 #ifdef _DEBUG
#0029 virtual void AssertValid() const;
#0030 virtual void Dump(CDumpContext& dc) const;
#0031 #endif
#0032
#0033 protected:
#0034
#0035 // Generated message map functions
#0036 protected:
#0037 //{{AFX_MSG(CScribbleDoc)
#0038     // NOTE - the ClassWizard will add and remove member functions here.
#0039     //      DO NOT EDIT what you see in these blocks of generated code !
#0040 //}}AFX_MSG
#0041 DECLARE_MESSAGE_MAP()
#0042 };
```

SCRIBBLEVIEW.H

```
#0001 // ScribbleView.h : interface of the CScribbleView class
#0002 //
#0003 //////////////////////////////////////
#0004
#0005 class CScribbleView : public CView
#0006 {
#0007 protected: // create from serialization only
#0008     CScribbleView();
#0009     DECLARE_DYNCREATE(CScribbleView)
#0010
#0011 // Attributes
#0012 public:
#0013     CScribbleDoc* GetDocument();
#0014
#0015 // Operations
#0016 public:
#0017
#0018 // Overrides
#0019 // ClassWizard generated virtual function overrides
#0020 //{{AFX_VIRTUAL(CScribbleView)
#0021 public:
#0022     virtual void OnDraw(CDC* pDC); // overridden to draw this view
#0023     virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
#0024 protected:
#0025     virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
#0026     virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
#0027     virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
```

```

#0028  //}}AFX_VIRTUAL
#0029
#0030  // Implementation
#0031  public:
#0032  virtual ~CScribbleView();
#0033  #ifdef _DEBUG
#0034  virtual void AssertValid() const;
#0035  virtual void Dump(CDumpContext& dc) const;
#0036  #endif
#0037
#0038  protected:
#0039
#0040  // Generated message map functions
#0041  protected:
#0042  //{{AFX_MSG(CScribbleView)
#0043  // NOTE - the ClassWizard will add and remove member functions here.
#0044  //      DO NOT EDIT what you see in these blocks of generated code !
#0045  //}}AFX_MSG
#0046  DECLARE_MESSAGE_MAP()
#0047  };
#0048
#0049  #ifndef _DEBUG // debug version in ScribbleView.cpp
#0050  inline CScribbleDoc* CScribbleView::GetDocument()
#0051  { return (CScribbleDoc*)m_pDocument; }
#0052  #endif

```

STDAFX.H

```

#0001  // stdafx.h : include file for standard system include files,
#0002  // or project specific include files that are used frequently, but
#0003  //      are changed infrequently
#0004  //
#0005
#0006  #define VC_EXTRALEAN // Exclude rarely-used stuff from Windows headers
#0007
#0008  #include <afxwin.h> // MFC core and standard components
#0009  #include <afxext.h> // MFC extensions
#0010  #ifndef _AFX_NO_AFXCMN_SUPPORT
#0011  #include <afxcmn.h> // MFC support for Windows Common Controls
#0012  #endif // _AFX_NO_AFXCMN_SUPPORT

```

RESOURCE.H

```

#0001  //{{NO_DEPENDENCIES}}
#0002  // Microsoft Visual C++ generated include file.

```

```
#0003 // Used by SCRIBBLE.RC
#0004 //
#0005 #define IDR_MAINFRAME            128
#0006 #define IDR_SCRIBTYPE            129
#0007 #define IDD_ABOUTBOX             100
#0008
#0009 // Next default values for new objects
#0010 //
#0011 #ifdef APSTUDIO_INVOKED
#0012 #ifndef APSTUDIO_READONLY_SYMBOLS
#0013 #define _APS_3D_CONTROLS            1
#0014 #define _APS_NEXT_RESOURCE_VALUE    130
#0015 #define _APS_NEXT_CONTROL_VALUE     1000
#0016 #define _APS_NEXT_SYMED_VALUE       101
#0017 #define _APS_NEXT_COMMAND_VALUE     32771
#0018 #endif
#0019 #endif
```

STDAFX.CPP

```
#0001 // stdafx.cpp : source file that includes just the standard includes
#0002 //      Scribble.pch will be the pre-compiled header
#0003 //      stdafx.obj will contain the pre-compiled type information
#0004
#0005 #include "stdafx.h"
```

SCRIBBLE.CPP

```
#0001 // Scribble.cpp : Defines the class behaviors for the application.
#0002 //
#0003
#0004 #include "stdafx.h"
#0005 #include "Scribble.h"
#0006
#0007 #include "MainFrm.h"
#0008 #include "ChildFrm.h"
#0009 #include "ScribbleDoc.h"
#0010 #include "ScribbleView.h"
#0011
#0012 #ifdef _DEBUG
#0013 #define new DEBUG_NEW
#0014 #undef THIS_FILE
#0015 static char THIS_FILE[] = __FILE__;
#0016 #endif
#0017
```

```
#0018 ///////////////////////////////////////////////////
#0019 // CScribbleApp
#0020
#0021 BEGIN_MESSAGE_MAP(CScribbleApp, CWinApp)
#0022     //{AFX_MSG_MAP(CScribbleApp)
#0023     ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
#0024         // NOTE - the ClassWizard will add and remove mapping macros here.
#0025         //     DO NOT EDIT what you see in these blocks of generated code!
#0026     //}AFX_MSG_MAP
#0027     // Standard file based document commands
#0028     ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
#0029     ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
#0030     // Standard print setup command
#0031     ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
#0032 END_MESSAGE_MAP()
#0033
#0034 ///////////////////////////////////////////////////
#0035 // CScribbleApp construction
#0036
#0037 CScribbleApp::CScribbleApp()
#0038 {
#0039     // TODO: add construction code here,
#0040     // Place all significant initialization in InitInstance
#0041 }
#0042
#0043 ///////////////////////////////////////////////////
#0044 // The one and only CScribbleApp object
#0045
#0046 CScribbleApp theApp;
#0047
#0048 ///////////////////////////////////////////////////
#0049 // CScribbleApp initialization
#0050
#0051 BOOL CScribbleApp::InitInstance()
#0052 {
#0053     // Standard initialization
#0054     // If you are not using these features and wish to reduce the size
#0055     // of your final executable, you should remove from the following
#0056     // the specific initialization routines you do not need.
#0057
#0058     #ifdef _AFXDLL
#0059         Enable3dControls();      // Call this when using MFC in a shared DLL
#0060     #else
#0061         Enable3dControlsStatic(); // Call this when linking to MFC statically
#0062     #endif
#0063 }
```



```
#0064 // 侯俊杰注：0065~0068 为Visual C++ 5.0 新增
#0065 // Change the registry key under which our settings are stored.
#0066 // You should modify this string to be something appropriate
#0067 // such as the name of your company or organization.
#0068 SetRegistryKey(_T("Local AppWizard-Generated Applications"));
#0069
#0070 LoadStdProfileSettings(); // Load std INI file options (including MRU)
#0071
#0072 // Register the application's document templates. Document templates
#0073 // serve as the connection between documents, frame windows and views.
#0074
#0075 CMultiDocTemplate* pDocTemplate;
#0076 pDocTemplate = new CMultiDocTemplate(
#0077     IDR_SCRIBTYPE,
#0078     RUNTIME_CLASS(CScribbleDoc),
#0079     RUNTIME_CLASS(CChildFrame), // custom MDI child frame
#0080     RUNTIME_CLASS(CScribbleView));
#0081 AddDocTemplate(pDocTemplate);
#0082
#0083 // create main MDI Frame window
#0084 CMainFrame* pMainFrame = new CMainFrame;
#0085 if (!pMainFrame->LoadFrame(IDR_MAINFRAME))
#0086     return FALSE;
#0087 m_pMainWnd = pMainFrame;
#0088
#0089 // Enable drag/drop open
#0090 m_pMainWnd->DragAcceptFiles();
#0091
#0092 // Enable DDE Execute open
#0093 EnableShellOpen();
#0094 RegisterShellFileTypes(TRUE);
#0095
#0096 // Parse command line for standard shell commands, DDE, file open
#0097 CCommandLineInfo cmdInfo;
#0098 ParseCommandLine(cmdInfo);
#0099
#0100 // Dispatch commands specified on the command line
#0101 if (!ProcessShellCommand(cmdInfo))
#0102     return FALSE;
#0103
#0104 // The main window has been initialized, so show and update it.
#0105 pMainFrame->ShowWindow(m_nCmdShow);
#0106 pMainFrame->UpdateWindow();
#0107
#0108 return TRUE;
#0109 }
```

```
#0110
#0111 ///////////////////////////////////////////////////
#0112 // CAboutDlg dialog used for App About
#0113
#0114 class CAboutDlg : public CDialog
#0115 {
#0116 public:
#0117     CAboutDlg();
#0118
#0119 // Dialog Data
#0120 //{{AFX_DATA(CAboutDlg)
#0121 enum { IDD = IDD_ABOUTBOX };
#0122 //}}AFX_DATA
#0123
#0124 // ClassWizard generated virtual function overrides
#0125 //{{AFX_VIRTUAL(CAboutDlg)
#0126 protected:
#0127     virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
#0128 //}}AFX_VIRTUAL
#0129
#0130 // Implementation
#0131 protected:
#0132 //{{AFX_MSG(CAboutDlg)
#0133     // No message handlers
#0134 //}}AFX_MSG
#0135 DECLARE_MESSAGE_MAP()
#0136 };
#0137
#0138 CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
#0139 {
#0140     //{{AFX_DATA_INIT(CAboutDlg)
#0141     //}}AFX_DATA_INIT
#0142 }
#0143
#0144 void CAboutDlg::DoDataExchange(CDataExchange* pDX)
#0145 {
#0146     CDialog::DoDataExchange(pDX);
#0147 //{{AFX_DATA_MAP(CAboutDlg)
#0148 //}}AFX_DATA_MAP
#0149 }
#0150
#0151 BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
#0152 //{{AFX_MSG_MAP(CAboutDlg)
#0153     // No message handlers
#0154 //}}AFX_MSG_MAP
#0155 END_MESSAGE_MAP()
```

```
#0156
#0157 // App command to run the dialog
#0158 void CScribbleApp::OnAppAbout()
#0159 {
#0160     CAboutDlg aboutDlg;
#0161     aboutDlg.DoModal();
#0162 }
#0163
#0164 //////////////////////////////////////
#0165 // CScribbleApp commands
```

MAINFRM.CPP

```
#0001 // MainFrm.cpp : implementation of the CMainFrame class
#0002 //
#0003
#0004 #include "stdafx.h"
#0005 #include "Scribble.h"
#0006
#0007 #include "MainFrm.h"
#0008
#0009 #ifdef _DEBUG
#0010 #define new DEBUG_NEW
#0011 #undef THIS_FILE
#0012 static char THIS_FILE[] = __FILE__;
#0013 #endif
#0014
#0015 //////////////////////////////////////
#0016 // CMainFrame
#0017
#0018 IMPLEMENT_DYNAMIC(CMainFrame, CMDIFrameWnd)
#0019
#0020 BEGIN_MESSAGE_MAP(CMainFrame, CMDIFrameWnd)
#0021     //{AFX_MSG_MAP(CMainFrame)
#0022         // NOTE - the ClassWizard will add and remove mapping macros here.
#0023         //      DO NOT EDIT what you see in these blocks of generated code !
#0024     ON_WM_CREATE()
#0025     //}AFX_MSG_MAP
#0026 END_MESSAGE_MAP()
#0027
#0028 static UINT indicators[] =
#0029 {
#0030     ID_SEPARATOR,          // status line indicator
#0031     ID_INDICATOR_CAPS,
#0032     ID_INDICATOR_NUM,
#0033     ID_INDICATOR_SCRL,
```

```
#0034 };
#0035
#0036 //////////////////////////////////////
#0037 // CMainFrame construction/destruction
#0038
#0039 CMainFrame::CMainFrame()
#0040 {
#0041     // TODO: add member initialization code here
#0042
#0043 }
#0044
#0045 CMainFrame::~CMainFrame()
#0046 {
#0047 }
#0048
#0049 int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
#0050 {
#0051     if (CMDIFrameWnd::OnCreate(lpCreateStruct) == -1)
#0052         return -1;
#0053
#0054     if (!m_wndToolBar.Create(this) ||
#0055         !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
#0056     {
#0057         TRACE0("Failed to create toolbar\n");
#0058         return -1;    // fail to create
#0059     }
#0060
#0061     if (!m_wndStatusBar.Create(this) ||
#0062         !m_wndStatusBar.SetIndicators(indicators,
#0063         sizeof(indicators)/sizeof(UINT)))
#0064     {
#0065         TRACE0("Failed to create status bar\n");
#0066         return -1;    // fail to create
#0067     }
#0068
#0069     // TODO: Remove this if you don't want tool tips or a resizeable toolbar
#0070     m_wndToolBar.SetBarStyle(m_wndToolBar.GetBarStyle() |
#0071         CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC);
#0072
#0073     // TODO: Delete these three lines if you don't want the toolbar to
#0074     // be dockable
#0075     m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
#0076     EnableDocking(CBRS_ALIGN_ANY);
#0077     DockControlBar(&m_wndToolBar);
#0078
#0079     return 0;
```

```
#0080 }
#0081
#0082 BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
#0083 {
#0084     // TODO: Modify the Window class or styles here by modifying
#0085     // the CREATESTRUCT cs
#0086
#0087     return CMDIFrameWnd::PreCreateWindow(cs);
#0088 }
#0089
#0090 //////////////////////////////////////////////////
#0091 // CMainFrame diagnostics
#0092
#0093 #ifdef _DEBUG
#0094 void CMainFrame::AssertValid() const
#0095 {
#0096     CMDIFrameWnd::AssertValid();
#0097 }
#0098
#0099 void CMainFrame::Dump(CDumpContext& dc) const
#0100 {
#0101     CMDIFrameWnd::Dump(dc);
#0102 }
#0103
#0104 #endif // _DEBUG
#0105
#0106 //////////////////////////////////////////////////
#0107 // CMainFrame message handlers
```

CHILDFRM.CPP

```
#0001 // ChildFrm.cpp : implementation of the CChildFrame class
#0002 //
#0003
#0004 #include "stdafx.h"
#0005 #include "Scribble.h"
#0006
#0007 #include "ChildFrm.h"
#0008
#0009 #ifdef _DEBUG
#0010 #define new DEBUG_NEW
#0011 #undef THIS_FILE
#0012 static char THIS_FILE[] = __FILE__;
#0013 #endif
#0014
#0015 //////////////////////////////////////////////////
```

```
#0016 // CChildFrame
#0017
#0018 IMPLEMENT_DYNCREATE(CChildFrame, CMDIChildWnd)
#0019
#0020 BEGIN_MESSAGE_MAP(CChildFrame, CMDIChildWnd)
#0021     //{AFX_MSG_MAP(CChildFrame)
#0022         // NOTE - the ClassWizard will add and remove mapping macros here.
#0023         // DO NOT EDIT what you see in these blocks of generated code !
#0024     //}AFX_MSG_MAP
#0025 END_MESSAGE_MAP()
#0026
#0027 //////////////////////////////////////
#0028 // CChildFrame construction/destruction
#0029
#0030 CChildFrame::CChildFrame()
#0031 {
#0032     // TODO: add member initialization code here
#0033 }
#0034 }
#0035
#0036 CChildFrame::~CChildFrame()
#0037 {
#0038 }
#0039
#0040 BOOL CChildFrame::PreCreateWindow(CREATESTRUCT& cs)
#0041 {
#0042     // TODO: Modify the Window class or styles here by modifying
#0043     // the CREATESTRUCT cs
#0044
#0045     return CMDIChildWnd::PreCreateWindow(cs);
#0046 }
#0047
#0048 //////////////////////////////////////
#0049 // CChildFrame diagnostics
#0050
#0051 #ifdef _DEBUG
#0052 void CChildFrame::AssertValid() const
#0053 {
#0054     CMDIChildWnd::AssertValid();
#0055 }
#0056
#0057 void CChildFrame::Dump(CDumpContext& dc) const
#0058 {
#0059     CMDIChildWnd::Dump(dc);
#0060 }
#0061
```

```
#0062 #endif // _DEBUG
#0063
#0064 //////////////////////////////////////
#0065 // CChildFrame message handlers
```

SCRIBBLEDOC.CPP

```
#0001 // ScribbleDoc.cpp : implementation of the CScribbleDoc class
#0002 //
#0003
#0004 #include "stdafx.h"
#0005 #include "Scribble.h"
#0006
#0007 #include "ScribbleDoc.h"
#0008
#0009 #ifdef _DEBUG
#0010 #define new DEBUG_NEW
#0011 #undef THIS_FILE
#0012 static char THIS_FILE[] = __FILE__;
#0013 #endif
#0014
#0015 //////////////////////////////////////
#0016 // CScribbleDoc
#0017
#0018 IMPLEMENT_DYNCREATE(CScribbleDoc, CDocument)
#0019
#0020 BEGIN_MESSAGE_MAP(CScribbleDoc, CDocument)
#0021     //{AFX_MSG_MAP(CScribbleDoc)
#0022         // NOTE - the ClassWizard will add and remove mapping macros here.
#0023         //      DO NOT EDIT what you see in these blocks of generated code!
#0024     //}AFX_MSG_MAP
#0025 END_MESSAGE_MAP()
#0026
#0027 //////////////////////////////////////
#0028 // CScribbleDoc construction/destruction
#0029
#0030 CScribbleDoc::CScribbleDoc()
#0031 {
#0032     // TODO: add one-time construction code here
#0033
#0034 }
#0035
#0036 CScribbleDoc::~CScribbleDoc()
#0037 {
#0038 }
#0039
```

```
#0040 BOOL CScribbleDoc::OnNewDocument()  
#0041 {  
#0042     if (!CDocument::OnNewDocument())  
#0043         return FALSE;  
#0044  
#0045     // TODO: add reinitialization code here  
#0046     // (SDI documents will reuse this document)  
#0047  
#0048     return TRUE;  
#0049 }  
#0050  
#0051 ///////////////////////////////////////////////////  
#0052 // CScribbleDoc serialization  
#0053  
#0054 void CScribbleDoc::Serialize(CArchive& ar)  
#0055 {  
#0056     if (ar.IsStoring())  
#0057     {  
#0058         // TODO: add storing code here  
#0059     }  
#0060     else  
#0061     {  
#0062         // TODO: add loading code here  
#0063     }  
#0064 }  
#0065  
#0066 ///////////////////////////////////////////////////  
#0067 // CScribbleDoc diagnostics  
#0068  
#0069 #ifdef _DEBUG  
#0070 void CScribbleDoc::AssertValid() const  
#0071 {  
#0072     CDocument::AssertValid();  
#0073 }  
#0074  
#0075 void CScribbleDoc::Dump(CDumpContext& dc) const  
#0076 {  
#0077     CDocument::Dump(dc);  
#0078 }  
#0079 #endif //_DEBUG  
#0080  
#0081 ///////////////////////////////////////////////////  
#0082 // CScribbleDoc commands
```


SCRIBBLEVIEW.CPP

```
#0001 // ScribbleView.cpp : implementation of the CScribbleView class
#0002 //
#0003
#0004 #include "stdafx.h"
#0005 #include "Scribble.h"
#0006
#0007 #include "ScribbleDoc.h"
#0008 #include "ScribbleView.h"
#0009
#0010 #ifdef _DEBUG
#0011 #define new DEBUG_NEW
#0012 #undef THIS_FILE
#0013 static char THIS_FILE[] = __FILE__;
#0014 #endif
#0015
#0016 //////////////////////////////////////
#0017 // CScribbleView
#0018
#0019 IMPLEMENT_DYNCREATE(CScribbleView, CView)
#0020
#0021 BEGIN_MESSAGE_MAP(CScribbleView, CView)
#0022     //{AFX_MSG_MAP(CScribbleView)
#0023         // NOTE - the ClassWizard will add and remove mapping macros here.
#0024         //      DO NOT EDIT what you see in these blocks of generated code!
#0025     //}AFX_MSG_MAP
#0026     // Standard printing commands
#0027     ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
#0028     ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
#0029     ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
#0030 END_MESSAGE_MAP()
#0031
#0032 //////////////////////////////////////
#0033 // CScribbleView construction/destruction
#0034
#0035 CScribbleView::CScribbleView()
#0036 {
#0037     // TODO: add construction code here
#0038 }
#0039
#0040
#0041 CScribbleView::~CScribbleView()
#0042 {
#0043 }
#0044
```

```
#0045 BOOL CScribbleView::PreCreateWindow(CREATESTRUCT& cs)
#0046 {
#0047     // TODO: Modify the Window class or styles here by modifying
#0048     // the CREATESTRUCT cs
#0049
#0050     return CView::PreCreateWindow(cs);
#0051 }
#0052
#0053 //////////////////////////////////////////////////
#0054 // CScribbleView drawing
#0055
#0056 void CScribbleView::OnDraw(CDC* pDC)
#0057 {
#0058     CScribbleDoc* pDoc = GetDocument();
#0059     ASSERT_VALID(pDoc);
#0060
#0061     // TODO: add draw code for native data here
#0062 }
#0063
#0064 //////////////////////////////////////////////////
#0065 // CScribbleView printing
#0066
#0067 BOOL CScribbleView::OnPreparePrinting(CPrintInfo* pInfo)
#0068 {
#0069     // default preparation
#0070     return DoPreparePrinting(pInfo);
#0071 }
#0072
#0073 void CScribbleView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
#0074 {
#0075     // TODO: add extra initialization before printing
#0076 }
#0077
#0078 void CScribbleView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
#0079 {
#0080     // TODO: add cleanup after printing
#0081 }
#0082
#0083 //////////////////////////////////////////////////
#0084 // CScribbleView diagnostics
#0085
#0086 #ifdef _DEBUG
#0087 void CScribbleView::AssertValid() const
#0088 {
#0089     CView::AssertValid();
#0090 }
```

```
#0091
#0092 void CScribbleView::Dump(CDumpContext& dc) const
#0093 {
#0094     CView::Dump(dc);
#0095 }
#0096
#0097 CScribbleDoc* CScribbleView::GetDocument() // non-debug version is
#0098 {                                           // inline
#0099     ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CScribbleDoc)));
#0100     return (CScribbleDoc*)m_pDocument;
#0101 }
#0102 #endif //_DEBUG
#0103
#0104 //////////////////////////////////////
#0105 // CScribbleView message handlers
```

SCRIBBLE.RC (以下之码已经修剪,列出的主要目的是让你了解共有多少资源)

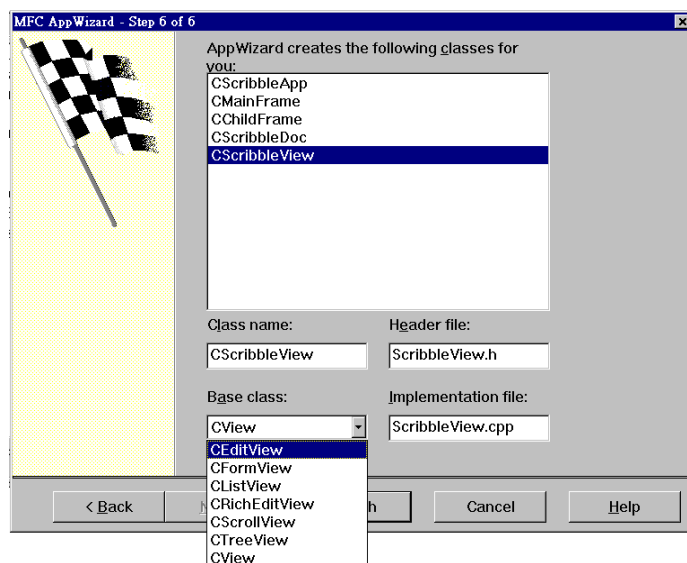
```
#0001 //Microsoft Visual C++ generated resource script.
#0002 //
#0003
#0004 #include "resource.h"
#0005 #include "afxres.h"
#0006
#0007 IDR_MAINFRAME          ICON      DISCARDABLE    "res\\Scribble.ico"
#0008 IDR_SCRIBTYPE          ICON      DISCARDABLE    "res\\ScribbleDoc.ico"
#0009
#0010 IDR_MAINFRAME          BITMAP     MOVEABLE PURE   "res\\Toolbar.bmp"
#0011
#0012 IDR_MAINFRAME TOOLBAR DISCARDABLE 16, 15
#0013 BEGIN
#0014     BUTTON      ID_FILE_NEW
#0015     BUTTON      ID_FILE_OPEN
#0016     BUTTON      ID_FILE_SAVE
#0017     SEPARATOR
#0018     BUTTON      ID_EDIT_CUT
#0019     BUTTON      ID_EDIT_COPY
#0020     BUTTON      ID_EDIT_PASTE
#0021     SEPARATOR
#0022     BUTTON      ID_FILE_PRINT
#0023     BUTTON      ID_APP_ABOUT
#0024 END
#0025
#0026 IDR_MAINFRAME MENU PRELOAD DISCARDABLE
#0027 BEGIN
#0028     POPUP "&File"
```

```
#0029      BEGIN
#0030      ...
#0031      END
#0032      POPUP "&View"
#0033      BEGIN
#0034      ...
#0035      END
#0036      POPUP "&Help"
#0037      BEGIN
#0038      ...
#0039      END
#0040  END
#0041
#0042  IDR_SCRIBTYPE MENU PRELOAD DISCARDABLE
#0043  BEGIN
#0044      POPUP "&File"
#0045      BEGIN
#0046      ...
#0047      END
#0048      POPUP "&Edit"
#0049      BEGIN
#0050      ...
#0051      END
#0052      POPUP "&View"
#0053      BEGIN
#0054      ...
#0055      END
#0056      POPUP "&Window"
#0057      BEGIN
#0058      ...
#0059      END
#0060      POPUP "&Help"
#0061      BEGIN
#0062      ...
#0063      END
#0064  END
#0065
#0066  IDR_MAINFRAME ACCELERATORS PRELOAD MOVEABLE PURE
#0067  BEGIN
#0068      ...
#0069  END
#0070
#0071  IDD_ABOUTBOX DIALOG DISCARDABLE 0, 0, 217, 55
#0072  CAPTION "About Scribble"
#0073  STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
#0074  FONT 8, "MS Sans Serif"
```

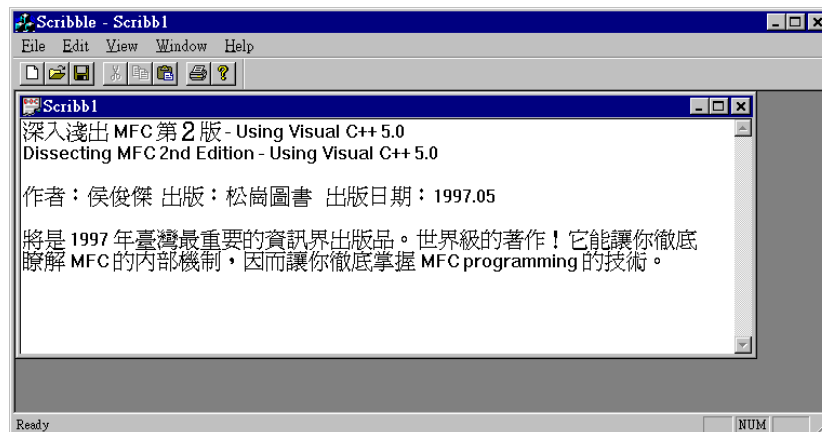
```
#0075 BEGIN
#0076      ...
#0077 END
#0078
#0079 VS_VERSION_INFO      VERSIONINFO
#0080     FILEVERSION      1,0,0,1
#0081     PRODUCTVERSION   1,0,0,1
#0082     FILEFLAGSMASK 0x3fL
#0083     #ifdef _DEBUG
#0084     FILEFLAGS 0x1L
#0085     #else
#0086     FILEFLAGS 0x0L
#0087     #endif
#0088     FILEOS 0x4L
#0089     FILETYPE 0x1L
#0090     FILESUBTYPE 0x0L
#0091 BEGIN
#0092     BLOCK "StringFileInfo"
#0093     BEGIN
#0094         BLOCK "040904B0"
#0095         BEGIN
#0096             VALUE "CompanyName",      "\0"
#0097             VALUE "FileDescription", "Scribble MFC Application\0"
#0098             VALUE "FileVersion",      "1, 0, 0, 1\0"
#0099             VALUE "InternalName",      "Scribble\0"
#0100             VALUE "LegalCopyright",    "Copyright (C) 1997\0"
#0101             VALUE "LegalTrademarks",   "\0"
#0102             VALUE "OriginalFilename",  "Scribble.EXE\0"
#0103             VALUE "ProductName",        "Scribble Application\0"
#0104             VALUE "ProductVersion",    "1, 0, 0, 1\0"
#0105         END
#0106     END
#0107     BLOCK "VarFileInfo"
#0108     BEGIN
#0109         VALUE "Translation", 0x409, 1200
#0110     END
#0111 END
#0112
#0113 //////////////////////////////////////
#0114 // String Table
#0115
#0116 STRINGTABLE PRELOAD DISCARDABLE
#0117 BEGIN
#0118     IDR_MAINFRAME "Scribble"
#0119     IDR_SCRIBTYPE "\nScrib\nScrib\nScribb Files
(*.scb)\n.scb\nScribble.Document\nScrib Document"
```

```
#0120 END
#0121
#0122 STRINGTABLE PRELOAD DISCARDABLE
#0123 BEGIN
#0124     AFX_IDS_APP_TITLE      "Scribble"
#0125     AFX_IDS_IDLEMESSAGE   "Ready"
#0126 END
#0127
#0128 STRINGTABLE DISCARDABLE
#0129 BEGIN
#0130     ID_INDICATOR_EXT       "EXT"
#0131     ID_INDICATOR_CAPS     "CAP"
#0132     ID_INDICATOR_NUM      "NUM"
#0133     ID_INDICATOR_SCRL     "SCRL"
#0134     ID_INDICATOR_OVR      "OVR"
#0135     ID_INDICATOR_REC      "REC"
#0136 END
#0137
#0138 STRINGTABLE DISCARDABLE
#0139 BEGIN
#0140     ID_FILE_NEW           "Create a new document\nNew"
#0141     ID_FILE_OPEN          "Open an existing document\nOpen"
#0142     ID_FILE_CLOSE         "Close the active document\nClose"
#0143     ID_FILE_SAVE          "Save the active document\nSave"
#0144     ...
#0145 END
```

好，我曾经说过，这个程序漂亮归漂亮，可什么也没做。我知道MFC 中有一个`CEditView`类别，具有文字编辑功能，我打算从那里继承我的View（现在的你还不了解什么是View，没关系）。于是我重来一次，一切都相同，只在AppWizard 的步骤六中设定`CScribbleView`的【Base class:】为`CEditView`：



这次我获得这样一个程序：



天啊，它不但有文字编辑功能，更有令人匪夷所思的打印功能和预览功能，也可以读写文字文件。

体会惊人的生产力了吗？

注意：在MFC AppWizard 的步骤6 中把 *CScribbleView* 的基础类别由 *CView* 改为 *CEditView*，会造成源代码如下的变化（粗体部份）：

```
// in ScribbleView.h
class CScribbleView : public CEditView
{
    ...
}

// in ScribbleView.cpp
IMPLEMENT_DYNCREATE(CScribbleView, CEditView)

BEGIN_MESSAGE_MAP(CScribbleView, CEditView)
    ...
    ON_COMMAND(ID_FILE_PRINT, CEditView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_DIRECT, CEditView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_PREVIEW, CEditView::OnFilePrintPreview)
END_MESSAGE_MAP()
// ScribbleView.cpp 中所有原先为CView 的地方，都被更改为CEditView

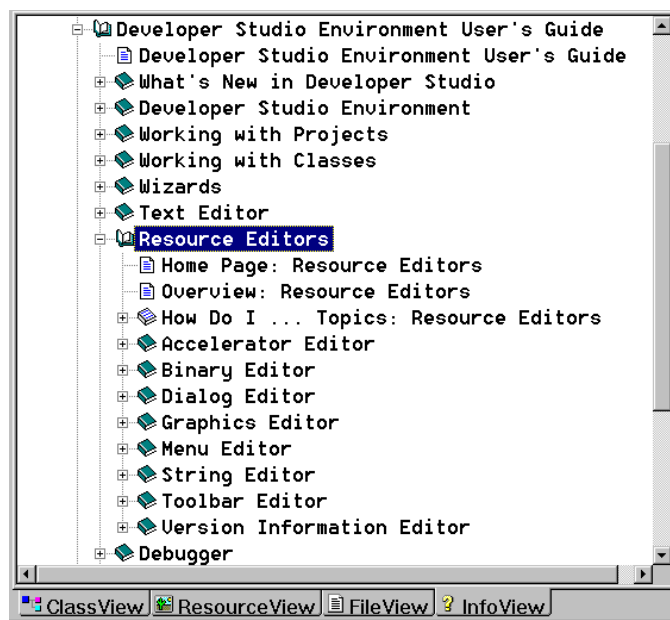
// in ScribbleDoc.cpp
void CScribbleDoc::Serialize(CArchive& ar)
{
    // CEditView contains an edit control which handles all serialization
    ((CEditView*)m_viewList.GetHead())->SerializeRaw(ar);
}
```


威力强大的资源编辑器

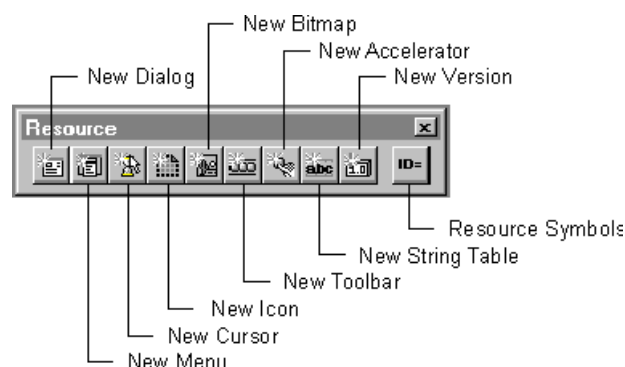
AppWizard 做出来的骨干程序带给我们Windows 程序的标准UI 接口。为了个人需求，你当然会另外加上一些资源，这时候你得准备启用资源编辑工具了。如果你曾经是 Visual C++ 的使用者，当记得曾有一个名为AppStudio 的多效合一资源编辑工具。是了，但现在不再有AppStudio，不再有独立的资源编辑工具，而是与Visual C++ 整合环境做了更密切的结合。

我将对这个工具提供的各种资源编辑功能逐一简介，并以实例展示如何在应用程序中加入新的资源项目。

资源的编辑，虽然与「正统」程序设计扯不上关系，但资源在Windows 程序所占的份量，众所周知。运用这些工具，仍然是你工作中重要的一环。VC++ 的Online 手册上有颇为完整的介绍；本章不能取代它们的地位，只是企图给你一个整体概观。以下是出现在InfoView 窗口中的Developer Studio Environment User's Guide 目录：



打开一个项目后，你可以从其ResourceView 窗口中看到所有的资源。想要编辑哪一个资源，就以鼠标双击之。如果要产生新的资源，整合环境的工具栏上有一整排的按钮等着你按。这个「资源工具栏」是选择性的，你可以按下整合环境的【Tools/Customize】菜单项目，再选择【Toolbar】附页（或是直接在工具栏区域中按下鼠标右键），从中决定要看到或不看到哪些工具栏。



选按其中任何一个钮，立刻会有一个适当的编辑器跳出来向你说哈！

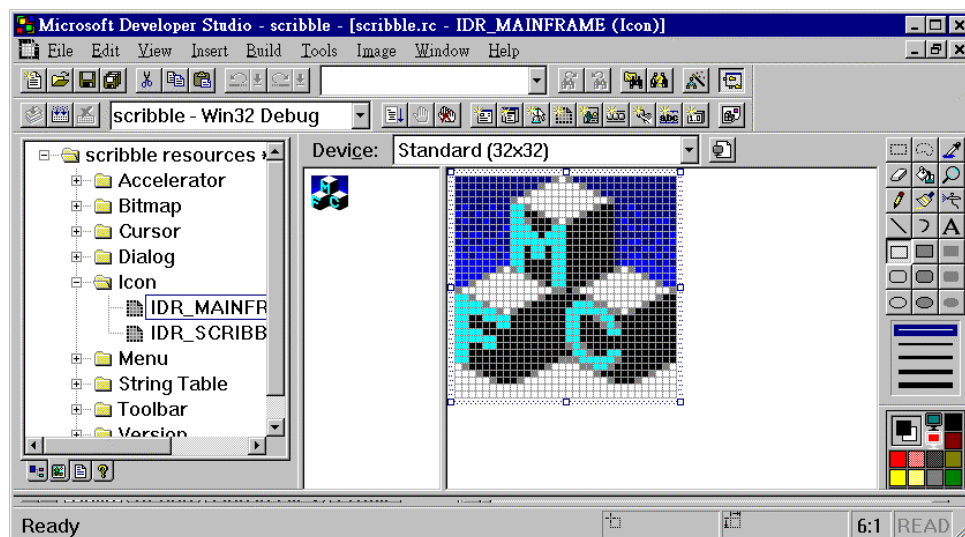
当然你可以用PE2 老古董直接编辑RC 档，但整合环境的好处是它会自动处理ID 号码，避免重复的情况发生，新的ID 并会自动放到你的RESOURCE.H 档中。总之就如我说过的，这些工具的目的在于使你专注于最主要的工作上，至于各文件间的关联工作，枝枝节节的琐碎事情，都由工具来完成。这，才叫作「整合性」工具环境嘛！

Icon 编辑器

Icon、Cursor、Bitmap 和Toolbar 编辑器使用同一个心脏：它们架构在同一个图形编辑器上，操作大同小异。过去这个心脏曾经遗漏两项重要功能，一是256 色图形支持，一是「敲入文字就出现对应之Bitmap」工具（这种工具允许使用者将文字直接键入一张bitmap 中，而不是一次一个图素慢慢地描）。自从Visual C++ 4.0 之后这两项重要功能就已经完全补齐了。

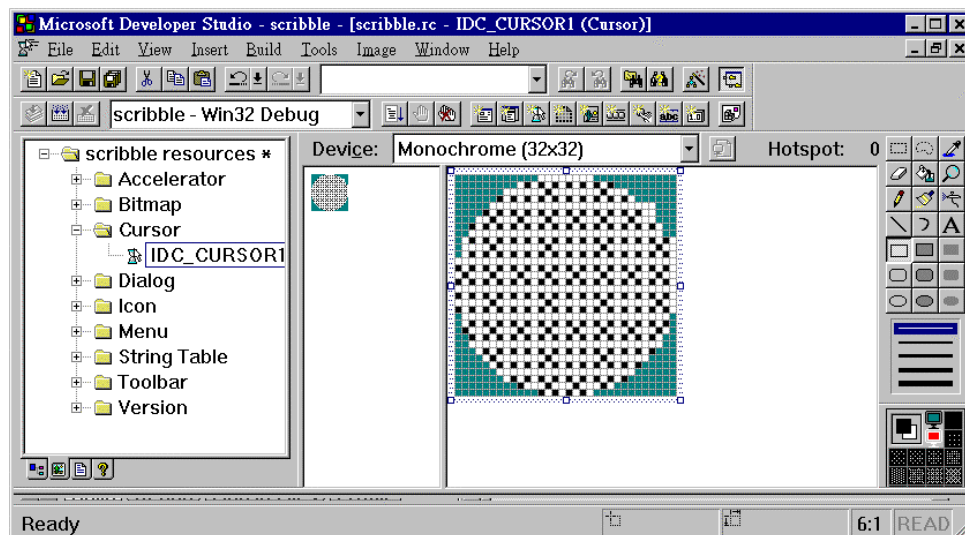
请注意工具箱（图最右侧）在不同的编辑器中稍有变化。

选按图左ResourceView 中的一个Icon，于是右侧出现Icon 编辑器。



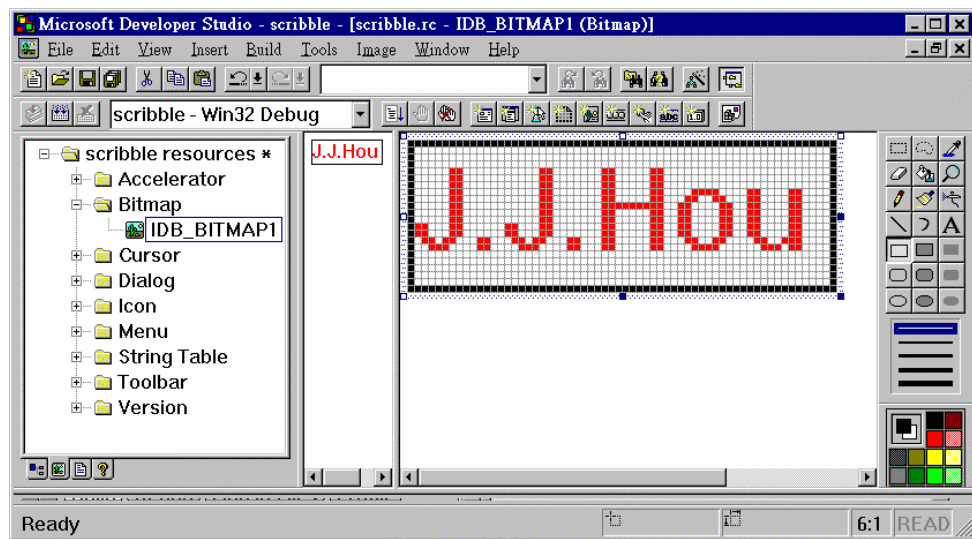
Cursor 编辑器

选按图左ResourceView 中的一个Cursor，于是右侧出现Cursor 编辑器。



Bitmap 编辑器

选按图左ResourceView 中的一张Bitmap，于是右侧出现Bitmap 编辑器。注意，本图的J.J.Hou 字样并非一点一点描绘而成，而是利用绘图工具箱（图最右）中的字形产生器（标有A 字形的那个图标）。它不但能够产生各种字形变化（视你安装的字形种类而定），在中文环境下更能够输入中文字！不过我还没有找到能够调整字形大小的功能。

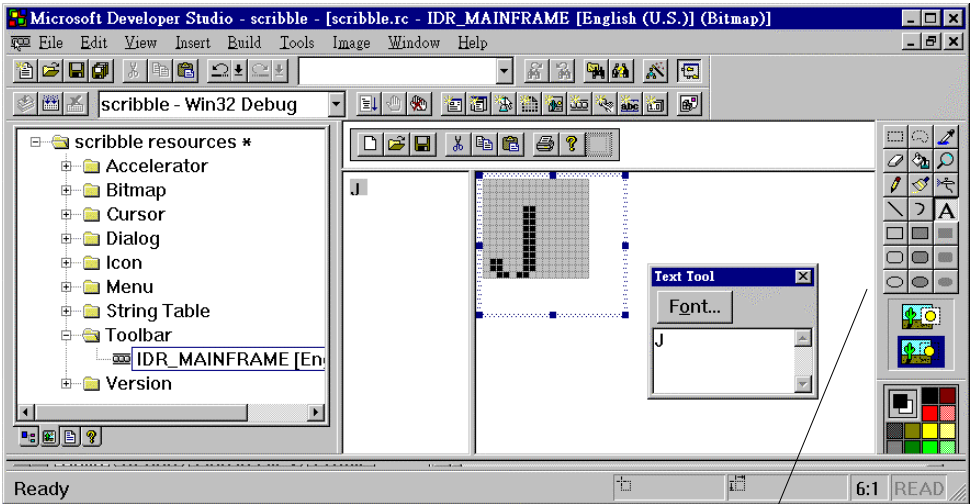


工具栏 (Toolbar) 编辑器

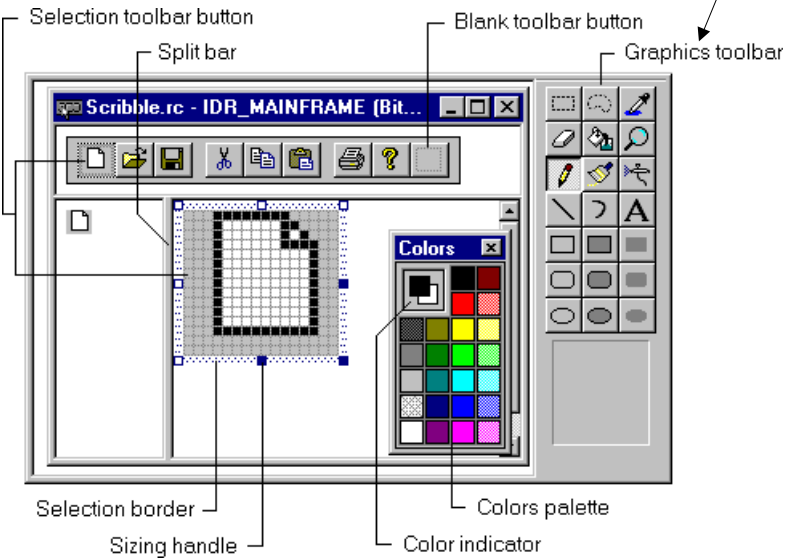
Visual C++ 早期版本没有这个编辑器，因为，工具栏原本不算是RC 档中的一份资源。而且，说穿了工具栏其实只是靠一张由固定大小之格状单元组成的一单张bitmap 构成，编辑工具栏其实就是编辑该张bitmap。但是那样一来，我们就得自己改写程序代码中有关于工具栏的设定部份，编辑程序显得不够一气呵成！

自从Visual C++ 4.0 开始，这中一切琐事就都由工具代劳了。我将在第7 章详细解释「工具列」资源如何在程序中发生效用。

选按图左ResourceView 中的一份Toolbar，于是右侧出现Toolbar 编辑器。



把上图局部放大来看：

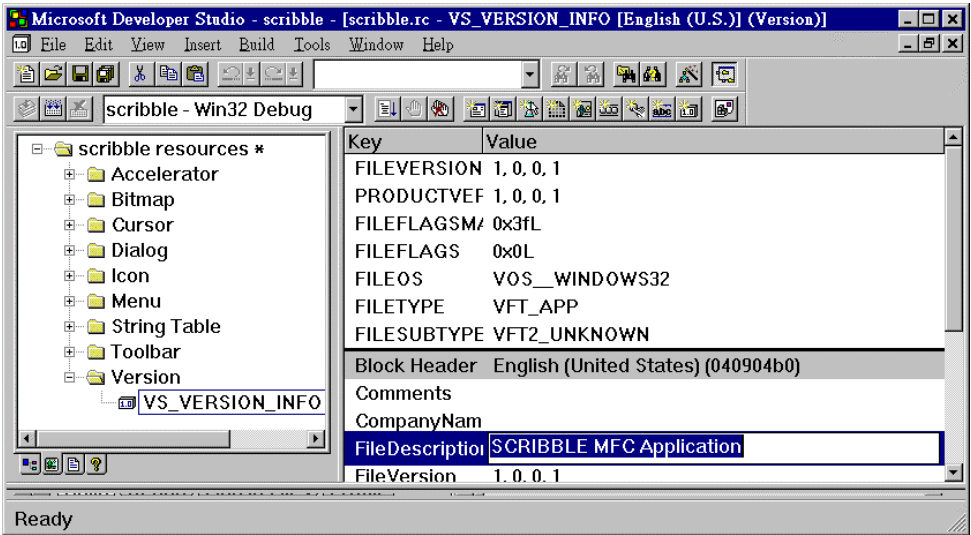


VERSIONINFO 资源编辑器

VERSIONINFO 可帮助程序判断存在于使用者系统中的文件版本号，如此一来就不会发生「以旧版本程序改写新格式之文件」的遗憾了。VERSIONINFO 资源也放在 RC 档，包含的资料可以识别版本、语言、操作系统、或含有资源之 DLL。AppWizard 会为你产生一份 VERSIONINFO 资源，但不强制你用它。下面是 Scribble.rc 档中有关于 VERSIONINFO 的内容：

```
#0001 VS_VERSION_INFO    VERSIONINFO
#0002     FILEVERSION      1,0,0,1
#0003     PRODUCTVERSION   1,0,0,1
#0004     FILEFLAGSMASK    0x3fL
#0005     #ifdef _DEBUG
#0006     FILEFLAGS 0x1L
#0007     #else
#0008     FILEFLAGS 0x0L
#0009     #endif
#0010     FILEOS 0x4L
#0011     FILETYPE 0x1L
#0012     FILESUBTYPE 0x0L
#0013 BEGIN
#0014     BLOCK "StringFileInfo"
#0015     BEGIN
#0016     BLOCK "040904B0"
#0017     BEGIN
#0018         VALUE "CompanyName",      "\0"
#0019         VALUE "FileDescription", "SCRIBBLE MFC Application\0"
#0020         VALUE "FileVersion",      "1, 0, 0, 1\0"
#0021         VALUE "InternalName",      "SCRIBBLE\0"
#0022         VALUE "LegalCopyright",    "Copyright \251 1996\0"
#0023         VALUE "LegalTrademarks",   "\0"
#0024         VALUE "OriginalFilename", "SCRIBBLE.EXE\0"
#0025         VALUE "ProductName",       "SCRIBBLE Application\0"
#0026         VALUE "ProductVersion",    "1, 0, 0, 1\0"
#0027     END
#0028     END
#0029     BLOCK "VarFileInfo"
#0030     BEGIN
#0031         VALUE "Translation", 0x409, 1200
#0032     END
#0033 END
```

选按图左ResourceView 中的一份VersionInfo，于是右侧出现VersionInfo 编辑器。你可以直接在每一个项目上修改字符串内容。



字符串表格（String Table）编辑器

字符串表格编辑器非常好用，允许你编辑RC 文件中的字符串资源（STRINGTABLE），这可增进国际化的脚步。怎么说？我们可以把程序中出现的所有字符串都集中在RC 文件的字符串表格，日后做中文版、日文版、法文版时只要改变RC 文件的字符串表格即可。噢当然，你还得选一套适当的Common Dialog DLL。

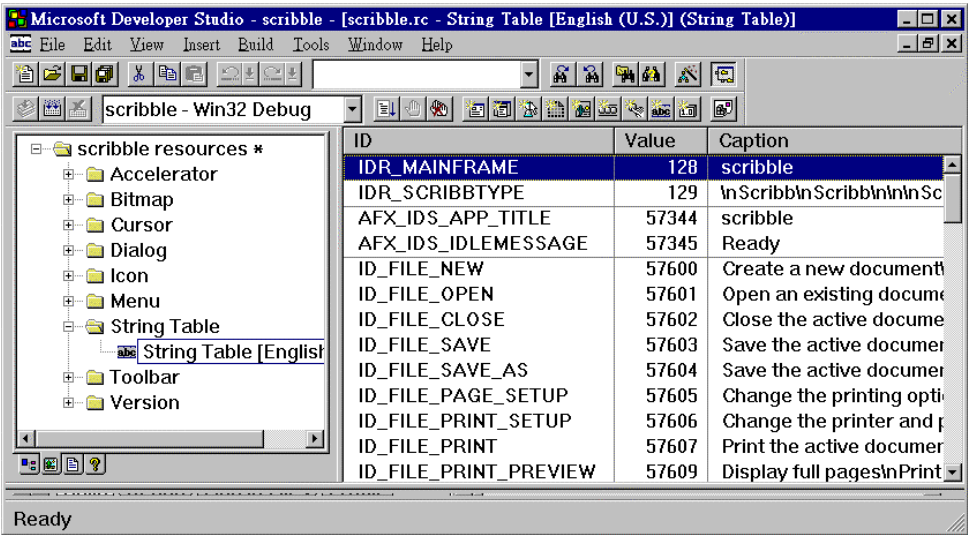
AppWizard 为我们制作骨干程序时不是加了一大套Menu 吗，对应于这些Menu，有数以打计的字符串资源，准备给状态列使用。下面是RC 文件字符串表格的一小部份：

```
STRINGTABLE DISCARDABLE
BEGIN
    ID_INDICATOR_EXT    "EXT"
    ID_INDICATOR_CAPS   "CAP"
    ID_INDICATOR_NUM     "NUM"
    ID_INDICATOR_SCRL   "SCRL"
    ID_INDICATOR_OVR    "OVR"
    ID_INDICATOR_REC    "REC"
END
```

```
STRINGTABLE DISCARDABLE
BEGIN
    ID_FILE_NEW          "Create a new document\nNew"
    ID_FILE_OPEN         "Open an existing document\nOpen"
    ID_FILE_CLOSE        "Close the active document\nClose"
    ID_FILE_SAVE         "Save the active document\nSave"
    ID_FILE_SAVE_AS      "Save the active document with a new name\nSave As"
...

```

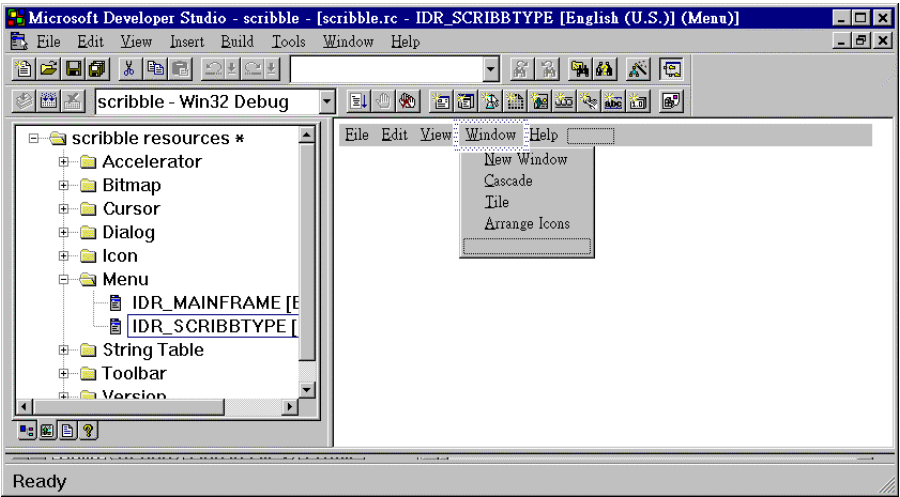
选按图左ResourceView 中的一个String Table , 于是右侧出现String Table 编辑器。你可以直接在每一个字符串上修改内容。



菜单 (Menu) 编辑器

菜单编辑器很好用。你可以一边看到正在建立的菜单，一边直接在适当位置键入菜单项目名称，窗体编辑器会把菜单项目的ID 值（当然是它自动为你产生的）放到RESOURCE.H 的#define 叙述中，就像字符串表格编辑器所做的那样。重新安排菜单项目的位置也很容易，因为所有动作都可以鼠标拖拉方式完成。

选按图左ResourceView 中的一套Menu，于是右侧出现Menu 编辑器。



假设我在菜单上添加一份popup 菜单，内有“JJHou” 和“MJChen” 两个项目。不但RC 档的MENU 资源有了变化：

```
IDR_MYTYPE MENU PRELOAD DISCARDABLE
BEGIN
    ...
    POPUP "MyFamily"
        BEGIN
            MENUITEM "JJHou",    ID_MYFAMILY_JJHOU
            MENUITEM "MJChen",   ID_MYFAMILY_MJCHEN
        END
    END
END
```

STRINGTABLE 也多了两个字符串定义，作为状态列消息：

```
STRINGTABLE DISCARDABLE
BEGIN
    ID_MYFAMILY_JJHOU    "J.J.Hou is a Good man"
    ID_MYFAMILY_MJCHEN   "M.J.Chen is a Good woman"
END
```

此外，RESOURCE.H 也多了两个常数定义：

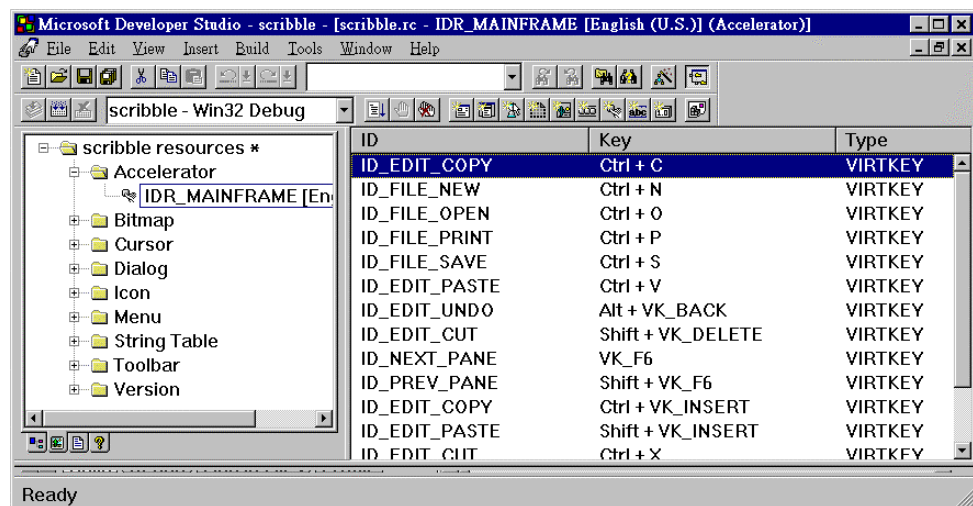
```
#define ID_MYFAMILY_JJHOU    32771
#define ID_MYFAMILY_MJCHEN   32772
```

此外也造成.CLW 档的变化，好让ClassWizard 知悉。ClassWizard 将在稍后介绍。

加速键 (Accelerator) 编辑器

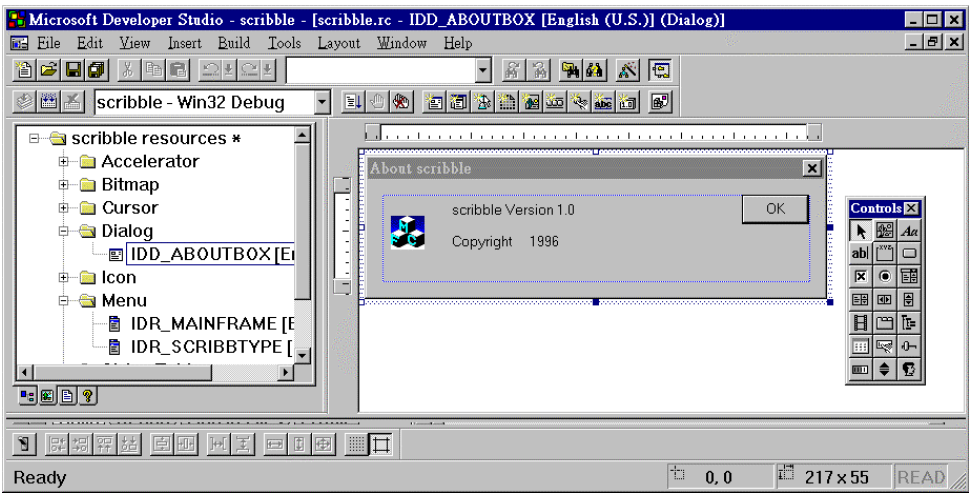
AppWizard 已经为骨干程序中的许多标准菜单项目设计了加速键。通常加速键是两个按键的组合（例如Alt + N），用以取代鼠标在层层菜单中的拉下、选按动作。所有的加速键设定都集中在RC 文件的加速键表格中，双击其中任何一个，就会出现加速键编辑器为你服务。你可以利用它改变加速键的按键组合。

选按图左ResourceView 中的一个Accelerator，于是右侧出现Accelerator 编辑器。你可以直接在每一个项目上修改内容。

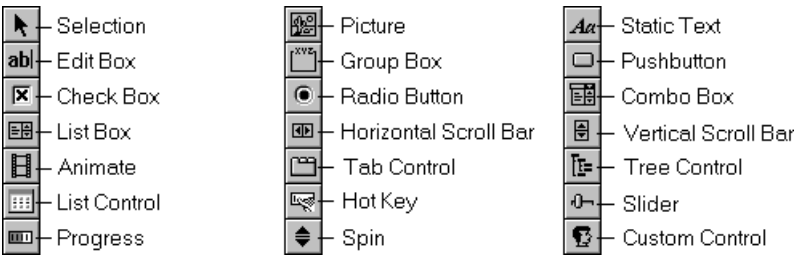


对话框（Dialog）编辑器

任何一个由AppWizard 产生出来的骨干程序，都有一个很简单朴素的"About" 对话框：
选按图左ResourceView 中的IDD_ABOUTBOX，右侧出现Dialog 编辑器并将About 对话框加载。



图右方有一个工具箱，内有许多控制组件（control）：



你可以在编辑器中任意改变对话框及控制组件的大小和位置，也可以任意拖拉工具箱内的组件放入对话框中。这些动作最后组成RC 文件中的对话框模板（Dialog template），也就是对话框外貌的文字描述，像这样：

```

IDD_ABOUTBOX DIALOG DISCARDABLE 0, 0, 217, 55
CAPTION "About Scribble"
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
FONT 8, "MS Sans Serif"
BEGIN
    ICON IDR_MAINFRAME, IDC_STATIC, 11, 17, 20, 20
    LTEXT "Scribble Version 1.0", IDC_STATIC, 40, 10, 119, 8, SS_NOPREFIX
    LTEXT "Copyright \251 1996", IDC_STATIC, 40, 25, 119, 8
    DEFPUSHBUTTON "OK", IDOK, 178, 7, 32, 14, WS_GROUP
END

```

Console 程序的项目管理

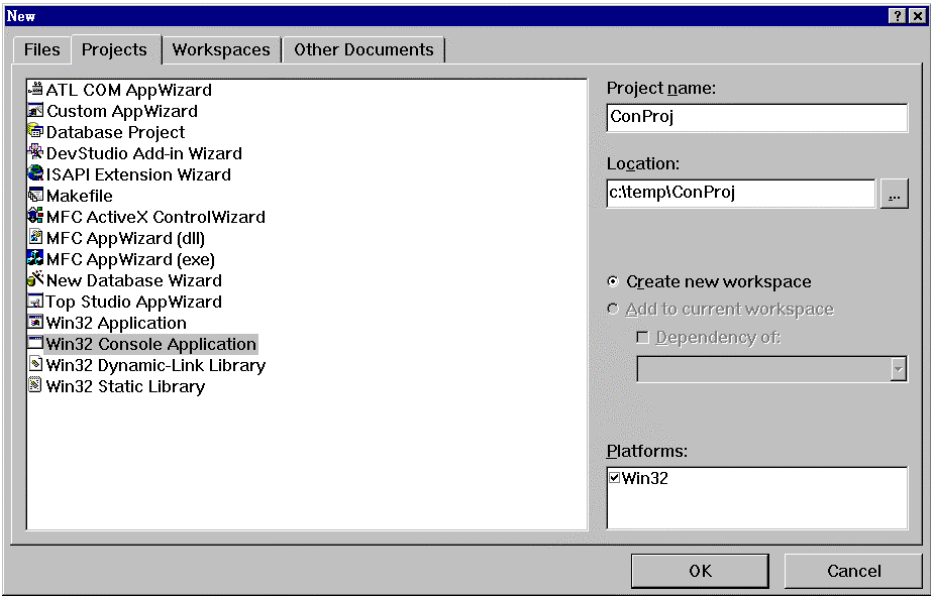
MFC AppWizard 会自动帮我们做出一个骨干程序的所有必须文件，建立起一个项目。但如果你想写一个「血统单纯」的纯粹C++ 程序呢？第1章曾经介绍过所谓的console 程式。第3章的所有范例程序也都是console 程序。

架构单纯的程序，如果文件只有一两个，直接使用命令列就可以了：

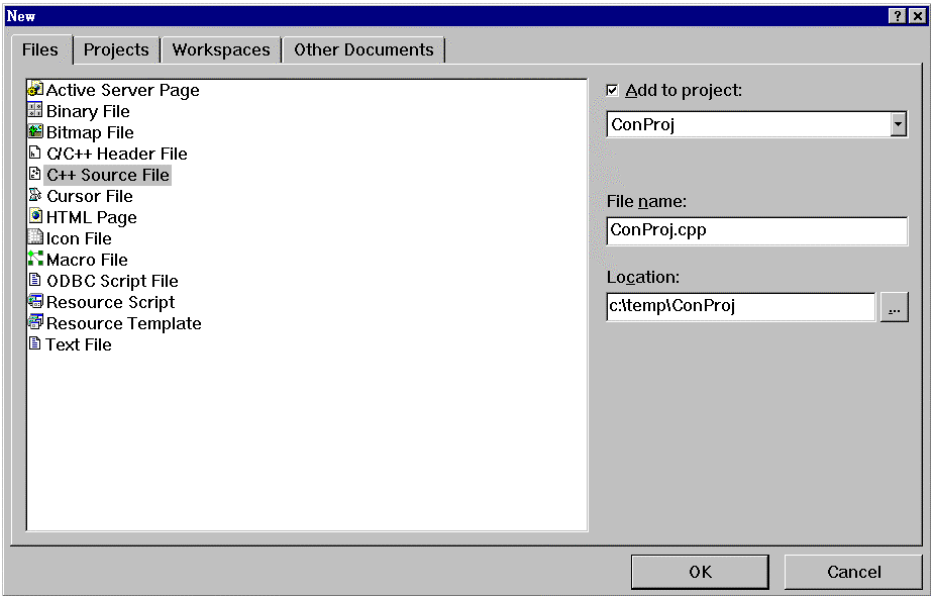
```
CL xxx.CPP <Enter>
```

如果组织架构比较复杂一点，文件有好几个，可以寻求项目管理员的协助。在Visual C++ 整合环境中建立一个conole 程序项目的步骤如下：

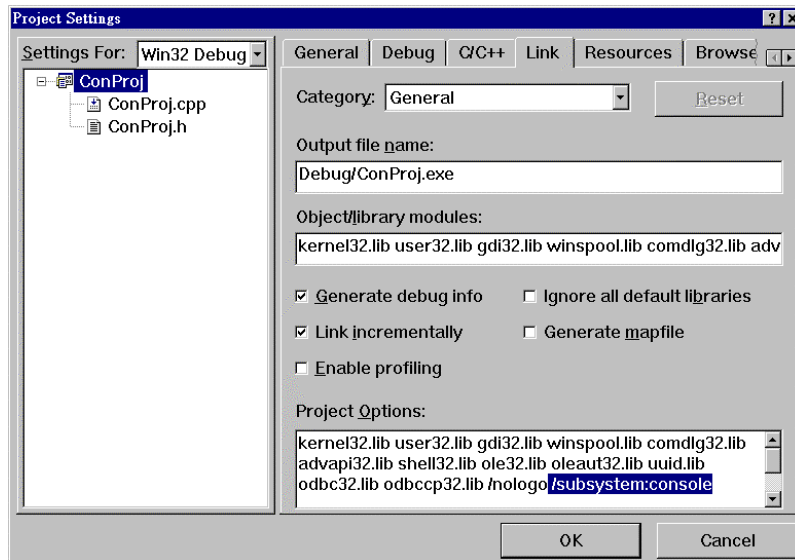
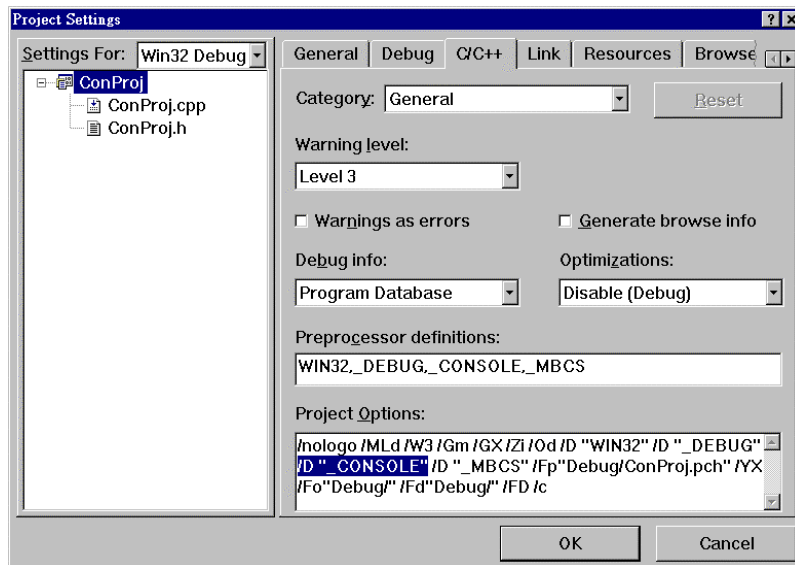
1. 选按整合环境的【File/New】，然后选择【Projects】附页，选按"Win32 Console Application"，并填写画面右端的项目名称和位置：



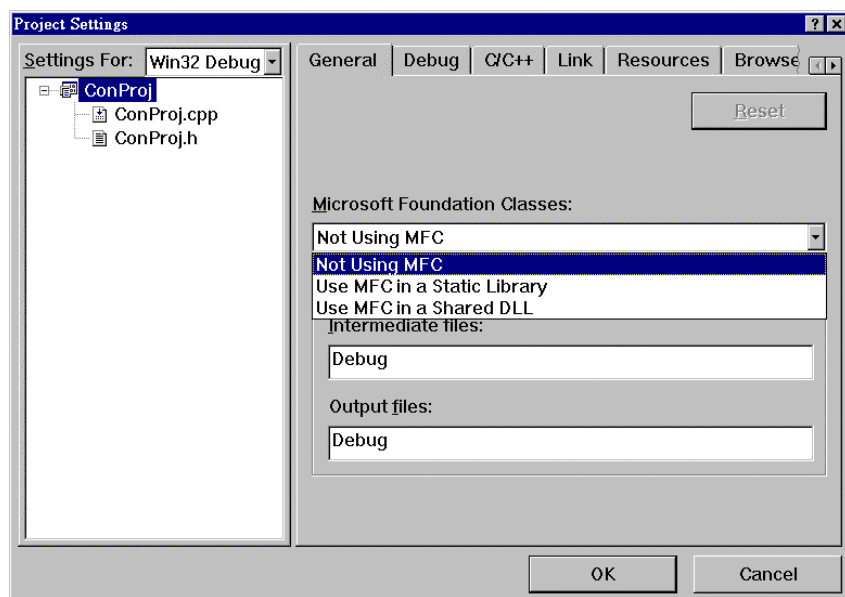
2. 按下【OK】钮，回到整合环境主画面，你可以选按【File/New】并选择【Files】附页，然后选按"C/C++ Header File" 或"C++ Source File" 以开启文件并撰写程序代码。开启的文件会自动加入此项目中。



3. 你可以选按整合环境的【Project/Setting】菜单项目，从中获得并修改整个项目的环境设定。我曾经在第 1 章提过，console 程序必须在编译时指定 `/D_CONSOLE` 常数，并在联结时指定 `subsystem:console`，这在以下两个画面中都可以看到（那是项目管理员自动为我们设定好的）：



第 1 章讨论console 程序时，我曾经说过，程序使用MFC 与否，关系到C runtime library 的单线程版或多线程版。是的，这项设定放在【General】附页之中：



你在这里所做的设定，会自动影响项目所联结的C runtime library 的版本。