第一部分

Visual C++基础

返回总目录

目　　录

第 1 章　Visual C++引论

只有真正的程序员才使用 Visual C++。反过来，则并不见得如此。最近似乎有许多专业的程序员放弃了使用这一语言而改用了其它产品提供的 RAD（快速应用开发）环境。许多人认为 Visual C++太古老、太烦琐、学起来太困难了。

说句实在话，Visual C++的学习周期确实比其它语言要长。可以很轻易地引入需要花上数小时才能排除的微妙错误这一事实一点也无助于提高程序员的开发效率。

然而，让我们先把这些负面缺陷暂时放到一边，来看一看 Visual C++具备的长处。使用 Visual C++主要的原因之一就在于它的灵活性。你可以彻底地控制整个开发环境。其它语言则更多地趋向于庇护程序员，当要做一些基本的东西时，它们会做得很好。不幸的是，当你需要使用像 C++这样的语言删除编程中的繁文缛节，并把任务完成时，这种保护作用就变成了开发工作的障碍。

长期以来，Visual C++一直拥有能够创建短小高效程序的美誉。使用这种语言编写的程序几乎可以与用汇编语言编写的程序达到相同的运行速度，并且避免了汇编语言存在的各种问题。C++实际上是介于汇编语言中寄存器编程的神秘莫测和像 Pascal 这种保护型编程环境方便性之中间难度的语言。

C++是编写诸如操作系统、设备驱动程序以及动态链接库（DLL）的强大语言，这一点不会不引起你的注意，这三种领域代码的开发依然是 Visual C++的主战场。Visual C++生成的短小、快速的代码在操作系统类对时间要求很高的

系统中获得了极高的赞誉。

到现在为止，我告诉你的一切你可能已经在其它书籍中看过无数遍了。下面就让我讲一些这个版本的 Visual C++中独有的内容。Microsoft 添加到这个版本的 Visual C++中的最新特性之一是更佳的原型能力，这一点通过增强的向导来实现。现在，这个特性还不能把 Visual C++提升到像 Visual Basic 向导相同的水平上，但它确实减少了开发应用程序的入门时间——这是个深受欢迎的变化。另外，增强的向导使得这个版本的 Visual C++比以前版本更为友好（本章的后续部分我将全部用于介绍 Visual C++的其它新特性）。

Visual C++也是编写 ActiveX 控件以及像 ISAPI（Internet 服务器应用程序接口）扩展和 ISAPI 过滤器这样针对 IIS 的专用代码的理想编程环境。即使对属于快速应用程序开发环境中的应用程序编程来说，也没有人愿意花时间从 Internet 上下载一个巨型控件，也同样没有人愿意让慢速的过滤器加重 Web 服务器的负担。Visual C++可以生成人们真正想要的简短可执行文件。另外，Visual C++提供的额外灵活性也使得编写这些类型的应用程序更加容易些。本书中我们将花些时间来研究 ActiveX 控件，因此这里我就不再多讲了。我还为使用 Web 服务器的人们准备了几个 ISAPI 扩展和 ISAPI 过滤器的示例。

Visual C++擅长的另一领域是数据库编程。我并不是说要使用 Visual C++编写一个功能完备的仓储控制系统——那样确实要花去很长的时间。然而，Internet 用户也确实需要访问数据库管理者那里的数据。Visual C++能够在不比 RAD 语言多多少编程的情况下提供快速的数据库访问。

1.1 为什么要用 Visual C++

我们已经研究了作为编程语言为什么要使用 C++的一般性问题。然而，对你想生成 Windows 应用程序时为什么要使用 Visual C++却谈论得不多。最基本的原因很好理解：由于 Microsoft 是 Windows 的生产厂商，那里的人们当然最清楚地知道该操作系统内部的东西是如何运行的。在 MFC（Microsoft 基础类库）的 Windows API 钩子中你会找到这些内幕知识。

Visual C++还提供了许多在其它产品中或许找不到的特性。尽管你可以通过构建自己的工具来克服其它产品存在的任何缺点，但为什么非要从头造车轮呢？本书中我们讨论最多的两个特性是数据库编程和 ActiveX 编程，两者都包括控件和文档。我们还将讲述与 Internet 相关的主题——比如 ISAPI——以及新的应用类型，比如基于 HTML 的应用。Visual C++让你能够在一个产品中访问所有这些技术，而无须购买第三方的产品。

持久性是使用 Visual C++的另一个原因。与我聊过天的一些程序员关心计算机界某些公司的未来。他们为这些公司以后不能对其产品提供支持而深感担忧。开发人员不仅今天需要、而且未来也需要得到强有力的支持。由于 C++的每种实现都存在一些细微的差别，特别是在你使用厂商提供的特有特性的情况下，你的应用程序确实与开发工具产品的命运联系在一起了。我们可以负责地说，在不远的将来 Microsoft 不会跑到哪儿去——使用它的产品就意味着你总可以升级到更好的产品，当然要在这些产品发行的时候。

1.2 本版 Visual C++的新特性

近来软件开发界流传的一个词是 Internet。你现在见到的每个产品都具备某些与 Internet 相关的小玩意儿，Visual C++也不例外。在本书中我们将考察与 Internet 相关的新特性和升级特性。表 1.1 列出了 Visual C++新特性的完整列表，并显示了哪些版本包含这些新特性（显然，企业版包含的特性最完整）。在随后的段落中你会找到 Visual C++ Internet 特有特性的概览，从而决定 Visual C++如何最佳地来满足你的 Internet、桌面以及数据库编程的需要。

表 1.1 Visual C++版本之间特性比较

特性	学习版（标准版）	专业版	企业版
活动模板库（ATL）	X[1]	X	X
应用分发许可	X	X	X
AppWizard 和数据源	X	X	X
通过 OLE-DB 进行的 ASA400 数据库访问			X
AutoCompletion（自动完成语句）	X	X	X
客户/服务器应用开发	\[2]	\	X

续表

代码优化和剖析器		X	X
创建定制化应用向导		X	X
数据绑定控件（RemoteData）		X	X
扩展的存储过程向导			X
InstallShield		X	X
Internet 信息服务器（IIS）4.0			X
MFC 数据绑定		X	X
MFC 数据库类（DAO，ODBC 以及文件 I/O）	X	X	X
MFC 数据库类（OLE-DB 和 ADO）；也称做 OLE-DB 模板		X	X
Microsoft Transaction Server（MTS）			X
代理服务器			X
远程自动化部件			X
远程数据对象（RDO）		X	X
SNA 服务器			X
SQL 数据库工具		X	X
SQL 编辑、调试以及存储过程调试			X

续表

SQL Server 6.5 Developer 版和服务包 3			X
静态链接 MFC 库		X	X
系统管理服务器（SMS）			X
TSQL 调试			X
可视化数据库工具		\	X
可视化建模器			X
Visual SourceSafe 版本控制			X
Visual Studio Enterprise Edition（VSEF）解决方案教程			X

[1]X　完全兼容

[2]\　部分兼容

/GZ 编译器选项（在调试模式下捕获发行版本中的错误）

你曾经遇到过在你的发行版中碰到了运行错误，但在调试模式错误又消失得无影无踪的情况吗？/GZ 编译器选项正是用于帮助你排除这类错误的法宝。该编译选项允许你使用/O1，/O2，/Ox 或/Og 优化开关，从而找出发行版错误的根源。然而，这时 Visual C++依然对代码中任何以#pragma 开始的语句进行优化，这就是说，有可能不会找出所有的优化缺陷。除此之外，/GZ 编译器选项还将完成下述三项任务：

自动初始化局部变量　Visual C++将自动把局部变量初始化为值 OxCC，这样你就会更快地找出未初始化的变量。该选项还会以 C4700

，这样你就会更快地找出未初始化的变量。该选项还会以 C4700 或 C4701 警告信息让你找到未正常引用的变量。

函数指针调用栈有效性检查 Visual C++检查每个函数，确保函数正常地使用堆栈。函数退出时检查 ESP 寄存器的值，以确保与函数进入时的值相同。这种检查让你能够发现，在使用函数指针调用函数时被调用函数期待的清理堆栈方式与调用函数提供的清理堆栈方式之间的不匹配情况。

调用堆栈有效性检查 Visual C++在每个调用结束时检查堆栈指针（ESP 寄存器），以保证该指针没有发生变化。这种检查让你能够发现内联汇编例程或函数调用中不匹配的错误。

_ _强制内联关键字

强制内联关键字在 Visual C++的前一个版本中已经提供，在这个版本中功能又进一步得到了增强。通过把被调用函数直接展开到调用函数内部的方法，内联函数避免了函数调用的消耗。使用内联函数可以加快应用程序的运行速度，然而不幸的是，内联代码也增加了可执行代码的长度，原因在于每当内联函数被调用时，都要嵌入到调用函数中。

正常情况下，Visual C++要对内联函数进行成本分析，以确定使用内联方式增加的代码长度是否能够得到相应的速度优势。在两种情况下，Visual C++按正常方式创建内联函数的独立副本并以函数方式而不是内联方式调用它：递归函数调用和在转换单元其它地方引用的函数。

_ _强制内联关键字通知 Visual C++不执行正常的成本分析并且总把内联函数编译成内联代码，换句话说，程序员负责考虑某些情况下（比如递归函数调用）可能产生极大规模代码的问题。因此，在使用这个新的关键字时要十分小心，确保代码的规模保持在合理的范围内。

即使使用了_ _强制内联关键字，在某些情况下编译程序依然不生成内联代码。例如，使用/Ob0 命令行开关（调试模式的一个开关）就不生成内联代码。不能生成内联代码的另一种情况是被调用函数使用了变长参数表。无论什么情况下，当编译器不能生成内联代码时，它都会向你发出第一级的警告信息（编号为 4714）。

ADO 数据绑定

Microsoft 不断地努力改进其编程语言产品的数据访问能力。ADO（ActiveX Data Object，ActiveX 数据对象）代表了提供数据访问的一条新的途径，它通过把数据绑定 ActiveX 控件与 ADODC（ADO Data Control，ADO 数据控件）结合起来来实现。ADODC 作为数据源定义了要显示信息的存储位置和数据访问需求。需要为 ADODC 提供六段信息：OLE-DB 提供者的名称（比如 SQL Server）、DSN（像在控制面板的 ODBC 设置中那样指定的数据源名称）、用户名、保密字、记录源（通常是个 SQL 查询）以及连接字符串。ActiveX 控件用于显示数据源的内容。

与以前的数据库访问方法相比，ADO 有数项优点。下面就为你介绍这些优点：

独立地创建对象　你不再需要必须遵循对象的层次。这一特性允许你只创建所需要的对象，从而减少了内存需求并加快了应用程序的运行速度。

成批更新　代替把每次更新都发送给服务器，你可以在本地内存中收集这些更新然后一次将所有更新都发送给服务器。这样我们就改进了应用程序的性能（原因在于可以在后台进行更新），并降低了网络的负载。

存储过程　存储过程作为数据库管理器的一部分驻留在服务器上，并且能够在指定数据集上完成特定的任务。ADO 支持使用带有输入/输出参数和返回值的存储过程。

多种游标类型　从本质上讲，游标指向当前要操作的数据。理论上说，你甚至可以使用特定的后台游标。

返回行数限制　你只得到能够满足用户实际数据需求的数据量。

多记录集对象　该功能支持操作存储过程或批处理过程返回的多个记录集。

自由线索化对象　增强了 Web 服务器的功能。

有两种数据绑定模型可用于 ActiveX 控件。第一个模型是简单的数据绑定，它允许使用像文本框这样的 ActiveX 控件显示一条记录的一个字段；第二个模型是复杂的数据绑定，它允许使用像网格这样的 ActiveX 控件同时显示多条记录的多个字段。复杂的数据绑定也要求 ActiveX 控件管理要显示哪些记录和哪些字段，这是简单数据绑定时 ADODC 通常要完成的任务。

Visual C++本身带了一些支持 ADO 的 ActiveX 控件，其中包括：

DataGrid
DataCombo
DataList
Hierarchical Flex Grid
Date and Time Picker

ATL 复合控件

这一特性让你能够按自己的方式使用其它控件创建出新的 ATL（Active Template Library，活动模板库）控件。定制这样的控件时要打开一个对话框，然后在对话框中填入窗口和 ActiveX 控件。该对话框占据了新控件的整个客户区。使用这两个向导之一能够创建 ATL 复合控件：ATL COM AppWizard 或 ATL Object Wizard。

绝大多数程序员使用这项技术的原因是它能够创建一个作为单一 ActiveX 控件显示的窗体。例如，如果你现在正要求用户下载一个有许多控件的窗体来显示 Web 页面，那么使用 ATL 复合控件可以减低下载任务的复杂性。当你修改了 ATL 复合控件时，用户只需要下载一个控件也就可以了。这种下载方式不仅加快了控件的下载速度，而且也降低了下载失败的可能性（要么下载整个控件，要么不下载——而不会下载部分窗体）。

AutoCompletion

AutoCompletion 特性帮助你完成 C++头文件和源代码文件中的语句。应用该功能时，你会看到一个带有完成代码行可能选项的列表框。例如，考虑下面的代码：

```
CString  oMyString;

oMyString.
```

当你在 oMyString 后输入圆点时，你就会看到一个包含了 CString 对象的各种方法和属性的列表框，此时，你可以向下滚动列表框，找到所需的属性或方法后单击该属性或方法，相应的属性或方法就会填充到原点后面。当然也可以输入你要使用的方法或属性的前几个字母，列表框会随着输入过程的进行而朝希望的名称滚动。一旦你选择了某个方法或属性，Visual C++将自动键入你要使用的属性或方法的名称。这样就大大减少了文字输入的时间，从而有更多的时间编写代码。

技巧 如果你的环境中丢失了 AutoCompletion 列表框，那么在该列表框先前出现过的地方右击，然后从上下文菜单中选择 List Members 菜单项，这个列表框就会重新显示在你的面前。

现在假定你正在查看 AutoCompletion 列表框，但还不能肯定 CString 对象的 AnsiToOem()方法存在的限制。如果你加亮这个列表项并把鼠标指针放置到这个列表项上，就会看到一个气球帮助，它告诉你该方法最多可处理 255 个字符。

这种气球型帮助正是 Visual C++的另一个新特性，称做文档注释（Doc Comment）。

文档注释并不仅仅是 Visual C++软件包的一部分，你也可以把这样的注释添加到你的函数和变量上。方法很简单，你所要做的就是或者在函数或变量说明的前面加上一条注释，或者把注释放置到变量或函数说明的同一行上。利用这个特性你可以向其他人提供函数或变量的各种信息。显然，你不会希望把这个注释弄得很大，Microsoft 把文档注释的长度限制为 12 行，对绝大多数情况来说在这个限制下也足够使用了。

文档注释并不局限于在 AutoCompletion 列表框中使用。把鼠标指针放置到某个变量或函数上，你会看到在与 AutoCompletion 列表框中显示的相同的文档注释气球。正如你所看到的，文档注释是为你的应用程序提供连续文档的迅捷方法，并且也使得阅读你的代码的其他人更方便地理解代码。

改进的 ClassView 和 WizardBar

Microsoft 为 ClassView 和 WizardBar 都提供了一些新特性。下面的列表予以介绍：

删除成员函数 现在不需要你做许多工作就可以删除成员函数了。你所必须完成的所有任务就是把光标放置到成员函数内部，然后从 WizardBar Action 下拉列表框中选择 Delete 或在 ClassWizard 对话框的 Message Maps 选项卡中选择 Delete Function。无论使用哪种方法，Visual C++都从头文件中删除函数声明并在源代码中注解掉函数体。这种处理方式让你在需

要被删除掉的函数时，可以轻易地恢复该函数。

转移到对话框编辑器 这个功能只在创建对话框风格的应用程序时可以使用。选择包含所需对话框的类，然后在 WizardBar Action 下拉列表框中选择 Go to Dialog Editor 选项。

新窗体 向你的工程中添加新的窗体。你会看到一个 New Form 对话框，在这个对话框中选择窗体名、窗体的基类（最常用的基类包括CdaoRecordView，CformView，CDialog 以及 CRecordView）、以及资源编号。基类的选择部分依赖于你正创建的应用程序的类型和使用应用程序向导过程中所做的各种选择。某些情况下，你也将可以选择自动化方式并提供文档的模板信息。当某些选项当前不能使用时，Visual C++会把这些选项变成灰色方式显示。

跟踪空类 现在 WizardBar 可以跟踪所有类，即使那些没有成员函数的类也一样能够跟踪。如果你选择了某个没有成员函数的类，那么WizardBar的 Members 组合框中将显示“No Members - Create New Class”。缺省的 WizardBar 动作是向空类中添加一个新的成员。

注 使用 Insert | New Form 命令也可以打开 New Form 对话框。

命令行参数

Visual C++提供了一个崭新的 MSDEV 实用程序，利用这个实用程序，你可以直接从命令行来编译工程，而无须首先移出 MAKE 文件、然后再使用 NMAKE 实用程序来完成编译。MSDEV 实用程序的语法为：

MSDEV <Filename> [/MAKE "<Project Name> - <Configuration Name> |

ALL"] [/REBUILD /CLEAN /NORECURSE /OUT <Log File> /USEENV]

你必须提供 DSP（工程）或 DSW（工作区）文件名。下面的文字描述了可选的命令行开关：

/MAKE 该选项让指定编译时使用的工程和配置。工程名（Project Name）总是工程创建过程中你为工程起的名称。配置名（Configuration Name）既可以是像 Win32 Debug 这样的特定名称，也可以是所有可用的配置（使用 ALL 关键字）。Visual C++支持在一行上使用多个/MAKE 参数，因此，你可以创建任意多个配置。

/REBUILD 在开始编译前首先清除所有中间文件，然后建立新的中间文件。这种方式保证进行完整编译。

/CLEAN 在开始编译前首先清除所有中间文件，但并不重建中间文件。

/NORECURSE 编译当前工程，但并不编译与该工程相关的其他工程。

/OUT <Log File> 把屏幕输出重定向到一个记录文件中。利用该记录文件可以分析编译中发现的错误并进行相应校正。

/USEENV 忽略 DSW 或 DSP 文件中的 tools.options.directories 设置，并用当前环境中的设置进行替代。这样就可以不用重新打开 Visual C++而覆盖环境设置，但这种做法也意味着可能会产生错误。

全面改进的编译器

编译大型 PCH 文件的工程将花费更少的时间。虽然你会注意到对小型工程速度的改进，但对使用大型 PCH 文件的工程来说，这种改进更加明显。

延迟加载移入

正常情况下，有两种方式使用 DLL：可以让应用程序自动加载 DLL，也可以根据需要手工地加载和卸载 DLL。延迟加载特性为你提供了第三种选择，它允许应用程序在需要时自动加载 DLL。这就是说，如果用户未要求使用某种服务——比如说你的字处理应用程序中的拼写检查——那么系统就不会加载相应的 DLL。

对程序员来说，使用这一特性并不需要做多少工作。你要完成的所有任务就是在链接应用程序时加上/DELAYLOAD 命令行开关并把 DELAYIMP.LIB 添加到链接文件列表中。

动态 HTML

新的 MFC 类 CHTML 允许你把动态 HTML（DHTML）视图添加到你的应用程序中。这个类真正的功能就是在不需要多做多少编程的情况下可以把 Web 浏览器加入到应用程序中。显然，由于 Visual C++是微软的产品，你的应用程序中的 Web 浏览器将是 Internet Explorer 4 特性的翻版。这个特性让你能够以对用户近乎透明的方式把桌面应用和 Internet 应用混合在一起。

动态语法分析

随着向应用程序中键入新的类、函数以及变量信息，ClassView 现在能够自动进行更新。这样我们随时都可以看到应用程序中的变化而无须等待将它们保存到源文件中。这个特性也意味着像 WizardBar 和 ClassWizard 这样的部件总是最新的。

注释 这个特性现在还存在一些问题，微软许诺在该产品发行时这些问题将得到纠正。例如，某些输入错误将会导致 Visual C++不正确地猜测你的意图，结果造成了 IDE 的死循环。在所有情况下，重新启动 IDE 就会清除这样的问题。

编辑和继续特性

编辑和继续特性让你能够在调试应用程序时对应用程序做些小的改动，而无须从头重建工程。这个特性不仅让我们能够进行更快速的调试，而且也可以对应用程序执行“如果——那么”类型的分析。

使用编辑和继续特性时也存在一些限制。如果修改的代码超过了编辑和继续特性能够完成的范围，那么 Visual C++简单地提示你重建工程。下面的列表给出了不能使用编辑和继续特性的一般原则：

- 头文件
- 全局/静态数据
- 类定义和函数原型

- 引入新的变量类型
- 例外处理模块

编辑和继续特性在缺省情况下是打开的，因此使用该特性时不需要做任何特殊的工作。如果想关闭该特性，那么使用 Tools | Options 命令来显示 Options 对话框，选择 Debug 选项卡，不选中 Debug Commands Invoke Edit and Continue Debugging 选项。当打开或关闭该特性后，需要完整地重建工程。

扩展存储过程向导

存储过程让你能够以某种方式操作数据库中的数据并把结果输出给用户。Visual C++提供了一个新的向导帮助你创建 SQL Server 的存储过程。这些存储过程使用了带有名称为 Extended stored procedure(扩展存储过程)的函数的 COM 接口。当我们在 SQL Server 上注册了扩展存储过程 DLL 后，就可以访问使用标准 COM 接口检索出的数据了。

新的调试特性

Visual C++提供了丰富的新调试特性，这些特性让你能够更容易地找出应用程序中存在的缺陷。表 1.2 列出了所有这些新特性，并给出了每个特性的简单描述。

表 1.2　新调试特性概览

特性	描述
VARIANT 和 GUID 的自动扩展	AUTOEXP.DAT（自动扩展）文件包含了以更有意义方式显示 VARIANT 和 GUID 的新规则（AUTOEXP.DAT 文件放置在硬盘的 Program Files\Microsoft Visual Studio\Common\MSDev98\Bin 目录中）。例如，VARIANT 现在将以实际包含数据的数据类型方式显示；字符串就显示为字符串。另外，Visual C++同时显示 VARIANT 类型以及相应的值。GUID 并不一定要显示为一串数字，Visual C++将努力在注册表中查找 GUID 的名称并以名称进行显示
调试器格式化符号	Visual C++支持完整的新调试器符号： ,hr 像通用 COM 返回值那样显示 32 位结果或出错代码，比如 S_OK 或 E_NOTIMPL。如果它不能显示精确的出错值，Visual C++将试图把返回值转换成像“Not Enough memory”这样的出错信息。
	,mq 让你以四个四字方式显示内存，以此代替 64 位寄存器。
	,st 根据 AUTOEXP.DAT 文件中 Unicode Strings 设置值决定以 Unicode 还是 ANSI 格式来显示字符串。
	,wc 以解码的 Windows 类标志（WC_constants）来显示数值值。

续表

	,wm 以解码的 Windows 消息值（WM_constants）来显示数值值
反汇编输出	在反汇编输出中提供了更多的信息，其中包括可用时的符号信息
Load COFF & Exports	该选项放置在 Options 对话框的 Debug 选项卡中（使用 Tools | Options 命令可以打开该对话框）。这个选项让你加载附加的调试信息，这些信息可以被不是 Visual C++的其它编译器使用（Visual C++使用 CodeView 格式的调试信息）。例如，Visual Basic 在 MSVBVM50.DBG 文件中提供了通用对象文件格式（COFF）的调试信息。当没有其它调试信息（至少是 Visual C++能够识别的信息）可用时，移出选项就发挥作用了。Visual C++将加载每个 DLL 移出表的内容并将其转换成用于调试目的的符号信息
寄存器	几个新的寄存器将帮助我们找出应用程序中的更详细信息。Thread Information Blocks（TIBs，线程信息块）伪寄存器提供了当前线程的信息。ERR 伪寄存器显示应用程序中最近一次发生错误的错误代码。该寄存器与 GetLastError()函数调用完成相同的任务。对 ERR 寄存器加上,hr 修饰符后的功能将显示 32 位的出错代码，同时给出像“Not Enough memory”这样的出错信息。64 位 MMX 寄存器值现在显示在 Watch 和 Quick Watch 窗口中。在所有的 x86 机器上都显示这些寄存器，即使

	这些机器本身并不支持 MMX 指令
未修饰符号	简单地说，这个特性表示你不会再在函数名中看到隐藏在某些地方的无意义的一大串字符。另外，所有的函数名都将包含可用时的参数信息
V_Table 和函数指针显示	可能时，函数指针和虚表项像文本项（符号化的）那样显示，其中用参数来代替 16 进制地址

OLE-DB 提供者模板

使用 OLE-DB 提供者模板可以开发 OLE-DB 访问接口，特别是到远程数据库的连接。该模板并不完成什么神奇的功能，它所完成的任务也就是把访问 OLE-DB 技术的过程变得容易些。OLE-DB 是微软最新的数据库技术，它用于为包含 OLE-DB 提供者的数据库管理系统（DBMS）提供高性能的数据访问。

数据绑定控件的资源编辑器改进

资源编辑器（Resource Editor）让你可以使用所有最新的 ADO 和 OLE-DB 技术来创建与数据库的连接，你需要使用新型的 ADODC 数据源控件来创建连接。另外，还需要使用新型，与 ADO 兼容的数据绑定控件显示数据。这条规则的唯一例外情况是简单的绑定控件，我们可以使用这类控件在 ADODC 和 MSRDC（Microsoft Remote Data Control，微软远程数据控件）数据源控件之间内部交换数据。

对 DocObject 包容特性的向导支持

MFC 应用向导（AppWizard）在第三步中包含了一个新的选项，该选项只在选中了 Container 或 Both Container and Server 复合文档特性选项后才有效。

利用 DocObject 包容特性（同时打开 Active Document Container 选项），你可以把应用程序支持的所有类型的文档包含在一个框架中，而不需要对每类文档分别创建不同类型的框架。如果想准确地感受一下这种技术到底如何工作，那么就请看一看 Microsoft Office Binder 吧。正常情况下，OLE 允许用户与复合文档中的一个对象进行交互，而 DocObject 包容技术让用户能够在一个框架中激活整个文档，包括菜单、工具条以及其它所支持的特性。

WizardBar 性能增强

当资源编辑器中的 WizardBar 可见时，带大量对话框和控件的（其 CLW 文件很大）工程通常慢如蜗牛。Microsoft 已经采取措施改进了这项特性的性能，从而减少了 WizardBar 对 IDE 性能的冲击。

1.3 一些缺陷

正如当今市场上所有其它产品一样，Visual C++也存在一些并不能像所期待的那样（至少像你自己所期待的那样）完成工作的缺陷。我发现的这些缺陷（某些人将它们称做“臭虫”或遗憾的变化）中没有哪个缺陷十分恐怖，以至于阻

止了你的工作，但它们依然令人讨厌。Microsoft 或许正要发行 Visual C++ 6.0 的一个或多个补丁程序，因此，我希望这里谈论的一些缺陷在你读到本文时就已经修正过了。即使 Microsoft 不能提供解决问题的替代方案，你至少也应该清醒地意识到必须修改某些代码以适应新的特性。

注释 本节并不是要罗列 Visual C++ 所有缺陷的各个方面，而是指出一些明显的问题以及避开这些问题的方法。但愿 Microsoft 也会修正一些深层次、不明显、隐藏的缺陷——即使这些缺陷在这里我并没有提到。

你首先会注意到的一件事是 InfoViewer 已经被一个非集成、基于 MSDN（Microsoft Developer Network）的帮助所取代。虽然我们依然可以加亮某个命令后得到相应的帮助，但 MSDN 的非集成特性也就意味着由外部程序提供该帮助，这也说明了现在需要更多的内存来获取帮助信息。另外，MSDN 的查找能力与已经熟悉的 InfoViewer 相比简直不可同日而语。MSDN 缺少许多 InfoViewer 提供的选项，并且基于 MSDN 的帮助还存在其它许多问题，其中最主要的问题是它要求必须有良好的 Internet 连接。总而言之，你会发现 Visual C++ 6.0 中的帮助是一大倒退——这是 Microsoft 产品升级中最失败的决策之一。

另一个问题是并非所有复制到 Bin 文件夹中的程序都会作为 Microsoft Visual C++ 6.0 文件夹中的快捷方式显示在 Start（开始）菜单中。例如，在那个文件夹中存放有 Help Workshop 快捷方式，但却没有 Hotspot Editor 快捷方式。本书中我们将多次使用这些实用程序，因此，你至少应该知道这些程序放置在什么地方，之后你就可以把它们添加到 Windows 95、Windows 98 或 Windows NT 的

Start（开始）菜单中了。再说一句，虽然 Start（开始）菜单缺少某些快捷方式并不是什么了不得的问题，但我们依然奇怪为什么 Microsoft 不把这些事说在明处。

也许部分原因是由于 2000 年日期问题，现在我们不能再使用 COleDateTime 得到两位数字的日期了。虽然该类可以接收两位数字的日期，但随后对像 GetMonth 或 GetMinute 这样的成员函数的调用将导致失败。这样，对作为程序员的你，就不得不重新修改包括这些成员函数的老程序了。否则的话，尽管编写了正确无误的程序，你依然会遇到神秘的错误终止了程序的现象。

虽然界面本身并不存在自身的缺陷，但界面依然让使用多国语言的人们感到担心。Microsoft 正致力于将其所有语言工具产品的界面进行统一，也就是说除语言细节之外都是一致的，这样开发人员几乎不需要新的培训就能从一种环境迁移到另一种环境。但 Visual C++ 却是例外，它到现在还依然保持着一付老面孔。虽然对这样的界面你不需要重新学习，但对使用微软其它产品的开发者来说，他们在使用 Visual C++ 之前还必须首先熟悉该工具的集成开发环境（IDE）。幸运的是，微软计划在不久的将来改变这种状况（至少该公司中与我交流过的人们是这样说的）。

1.4 考察界面元素

在我们大量编程之前，我要花点时间讲述一下 Developer Studio 的界面，这样大家就统一口径了。你还将学习一些以前未曾用到过的新特性，例如，我们

要考察一下工具条并介绍其中的某些新特性。如果你已经是一位经验丰富的Visual C++ 程序员，并且自认为没有什么界面元素对你来说还是新东西，那就随意跳过这一节吧。

MFC Studio 窗口元素

当使用 Visual C++ 编辑程序时，你可以把 Visual C++ 的显示界面划分成三块功能区：工具条、视图以及编辑窗口。每个功能区都可以独立于其它功能区来操作，因此，可以在各功能区之间自由地切换。图 1.1 显示了典型编辑器的显示情况以及这三个功能区的位置。

注释 本节中介绍各种窗口、视图以及工具条。随后的各节中介绍它们的详细信息。

经常用到的窗口有两种类型：文本窗口和资源窗口。

图 1.1 显示了一个典型的文本窗口。无论什么时候，当你要修改代码或编辑文本文件时，看到的就是这类窗口。

Visual C++ 通过各种手段来简化编码工作。文本窗口中首先注意到的东西是代码的颜色。例如，关键字的缺省颜色为蓝色，而注释的颜色为绿色。这种方式的彩色代码让你能够轻易地分辨出代码的性质。

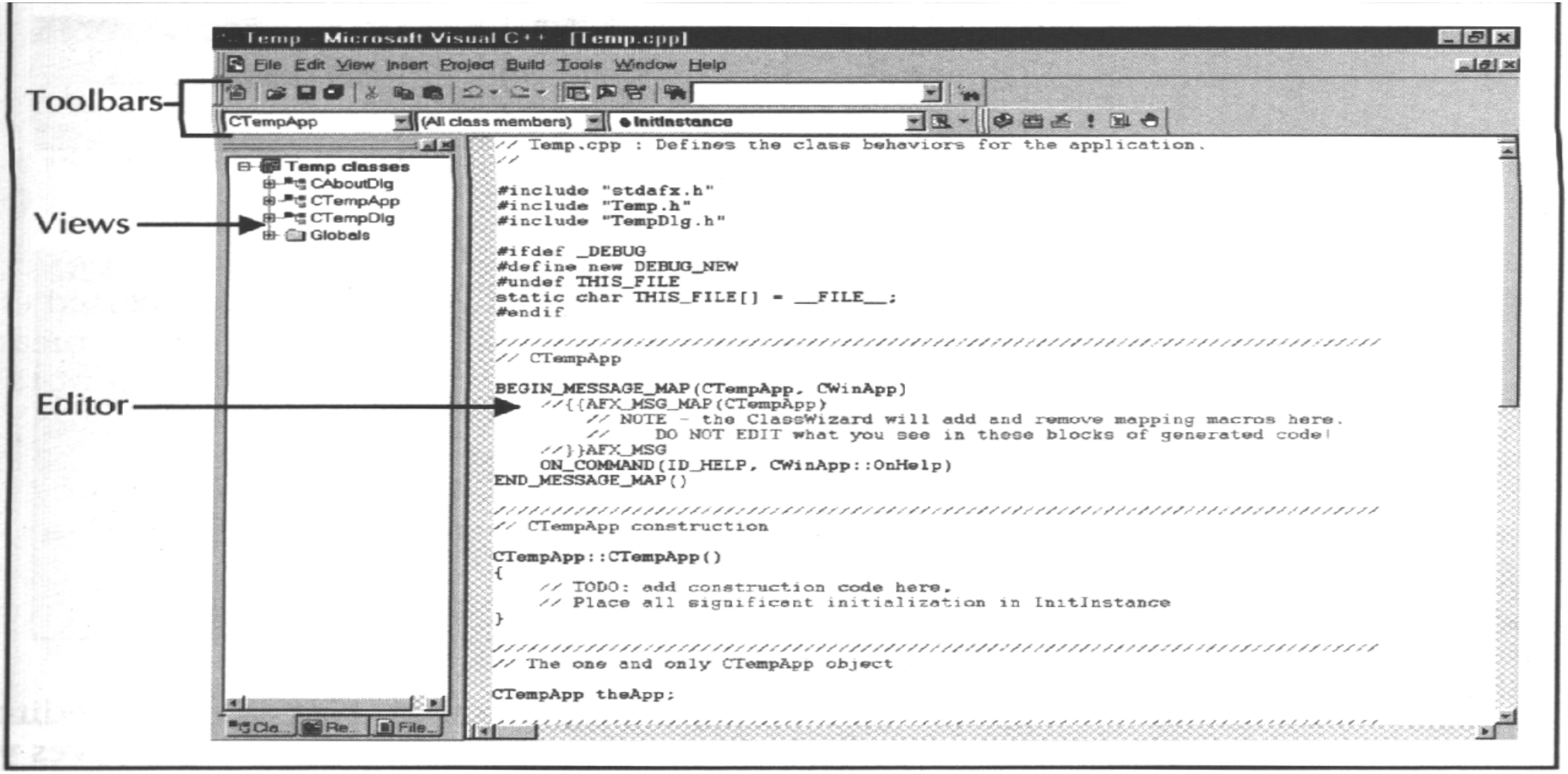


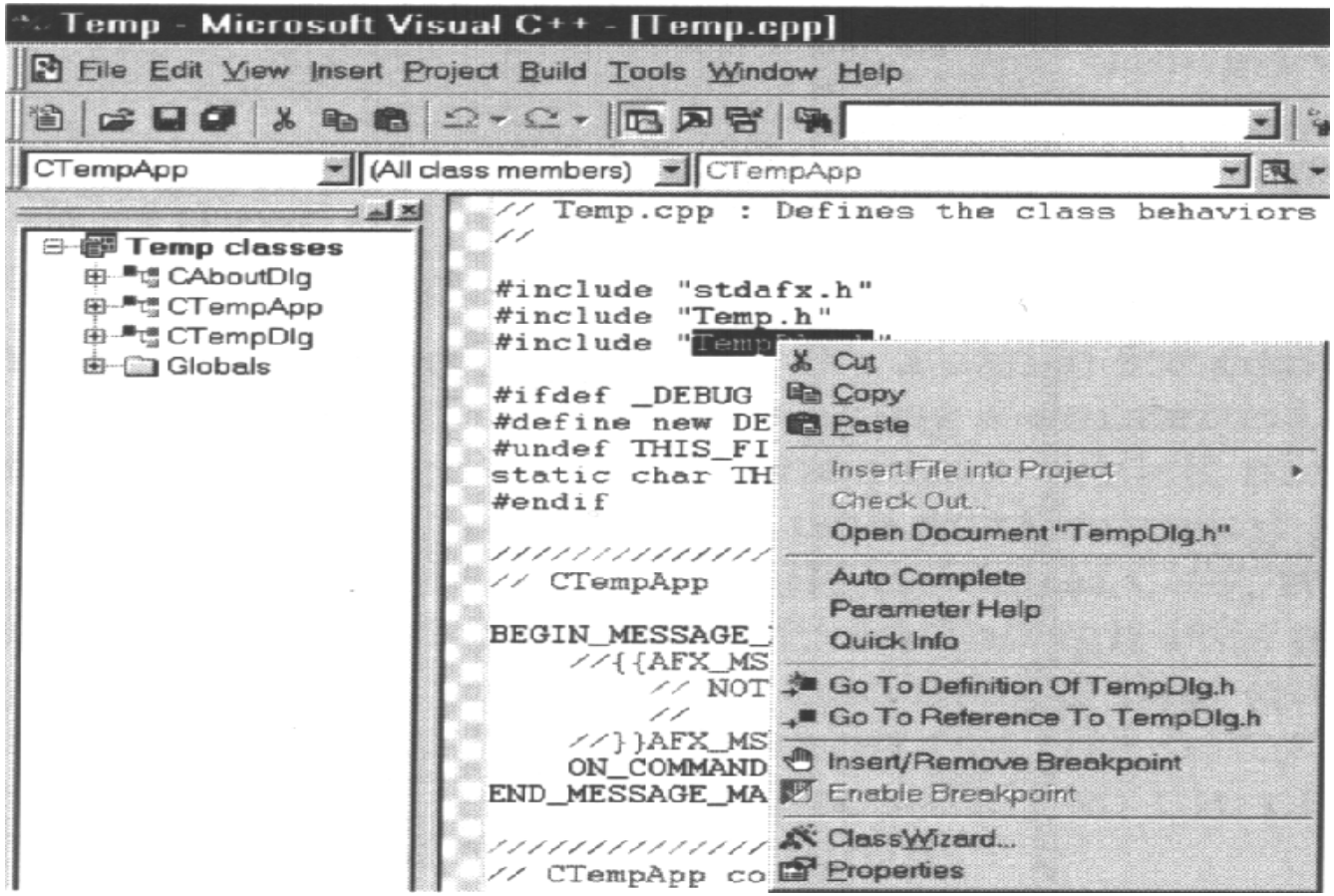
图 1.1 Developer Studio 的屏幕可以划分成三个功能区

本窗口的左边缘是一竖条，这是 Visual C++ 放置各种符号的地方，你甚至会看到它通过颜色来区分数据和代码控制区。例如，当我们在代码中设置了断

点后，Visual C++ 就在这个竖条上显示断点符号，当该断点可用时，断点符号为红色，否则，断点符号为白色。

文本框的各个区域中都可以通过右击来显示一个上下文相关菜单。上下文菜单中包含了可以进行基本操作的各种选项，与 Developer Studio 界面的其它地方不同，这些菜单不能直接进行修改。下图是右击文本编辑器时可能出现的上下文菜单的一个示例。

请注意，你可以像使用任何编辑器那样剪切、复制或粘贴文本。由于我已经加亮了#include 文件项，因此上下文菜单提供了一个打开该文件的机会。如果当前工程中还没有包含该文件，那么使用 Insert File into Project 菜单项就可以把该文件包含到工程中。Check Out 菜单项用于管理小组型工程，通过它可以获取对文件的控制，以便编辑文件。接下来的三个菜单项可以查找当前项更详细的信息。例如，使用 Parameter Help 菜单项可以得到函数调用的参数信息，Auto Complete 协助你完成函数调用的键入。后面的两个菜单项用于浏览工程。你可以找到某个特定项目在哪里引用、在哪里定义。由于这些项目依赖于建立工程时创建的 BSC 文件，因此，在使用这些项目前要确保最近有效地重建过工程。后面的两个菜单项——Insert/Remove Breakpoint 和 Enable Breakpoint 用于调试应用程序。最后，你可以打开 ClassWizard 来操作所选对象（本书将多次使用这种方法）或查看文档的属性。



我最后要介绍的窗口示例是你在编辑资源时看到的那种窗口，虽然资源编辑窗口比我们曾经谈过的其它窗口有更多的变化形式（无须着急，在本书的适当部分将介绍每一种资源编辑窗口），但图 1.2 是个典型的这种窗口。

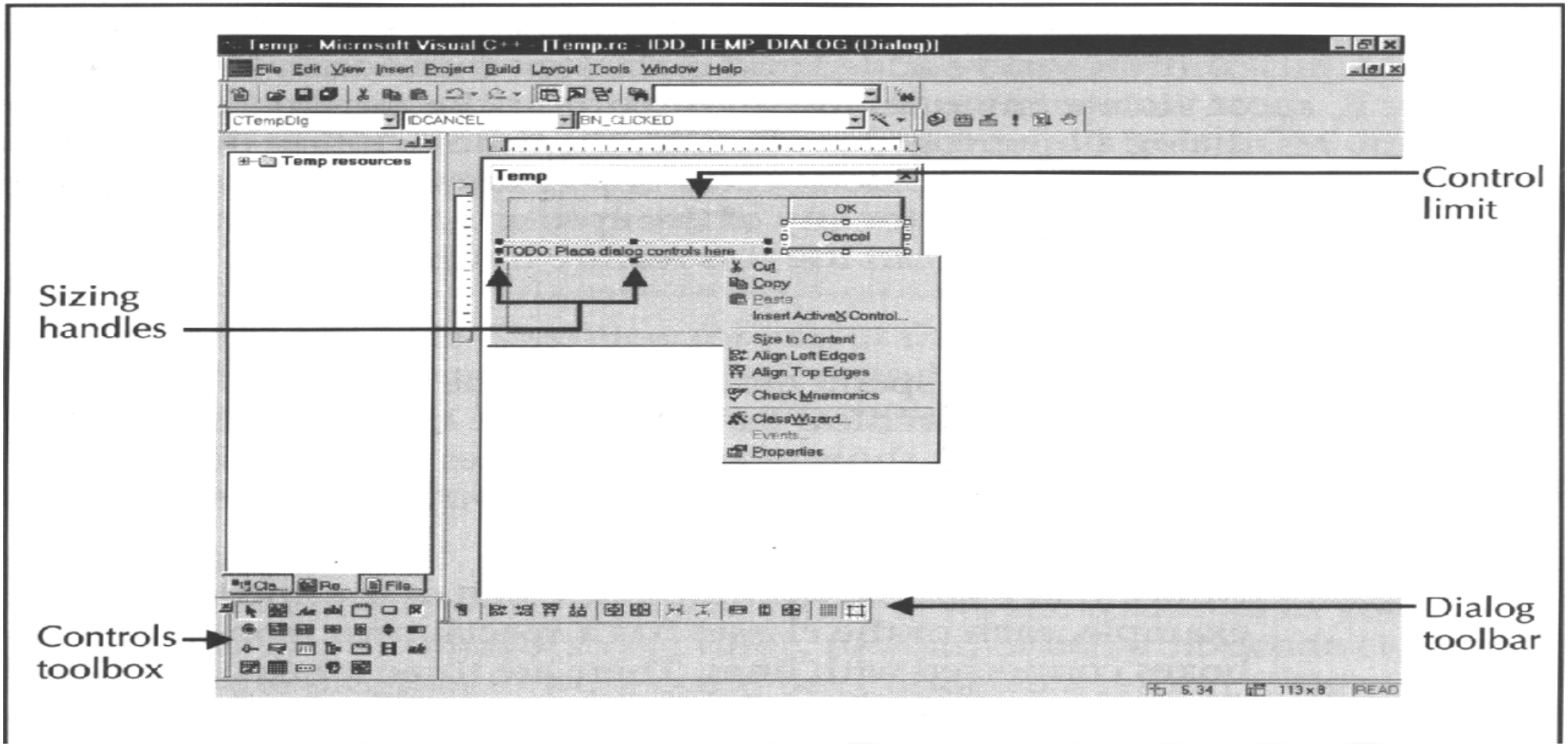


图 1.2　资源编辑窗口有许多不同的类型——这个资源编辑窗口显示了一个对话框

在上图的情况下，我们在查看对话框。这个对话框中有四个控件，其中两个控件被选中（两个选中控件的周围被改变大小的句柄所环绕）。在对话框边沿你还会看到一条暗淡的线条（缺省配置下线条的颜色为蓝色），这一线条确

定了对话框上放置控件的外边缘。

在图 1.2 的左下角，你会看到一个控件工具箱，它包含了可以放置到对话框中的各种控件，如果想在工具箱中添加新的控件，那么请使用 Project | Add to Project |Components and Controls 命令。工具箱的右边是对话框工具条，它提供了简化对话框布局排列的有用工具。例如，Make Same Size 按钮将选中的任意个数的控件与第一个选中控件等大。Test 按钮让你在对话框完成之前就可以看一看它的外观。

现在让我们讨论一下图 1.2 中显示的上下文菜单。由于前面我们已经讨论过这些菜单项的主要用途，因此这里只研究三个最重要的菜单项。使用 Insert ActiveX Control 菜单项可以把 ActiveX 控件添加到当前对话框中而无须将它实际添加到工程中。这样，你就可以在生成一大堆代码之前先看到该控件的工作效果。不过，如果你决定确实要使用该控件时，请记住以后把该控件添加到工程中。Check Mnemonics 菜单项告诉 Visual C++ 检查一下已经添加的所有控件，看是否存在同名错误或任何违反规则的情况。当做完了一个对话框后，这个检查相当重要。最后，Event 菜单项显示一个对话框，告诉我们所选对象都支持哪些事件。这个对话框还让你能够创建你想监控的任何事件的句柄。第 2 章中我们将讲述使用这一特性的方法。

类视图（ClassView）

或许类视图（ClassView）将花去你大部分的时间。该视图提供了工程中所有类的层次列表，通过扩展可以显示类中包含的细节。例如，类中将包含要编辑的成员函数等。图 1.3 显示了一个典型的类视图。

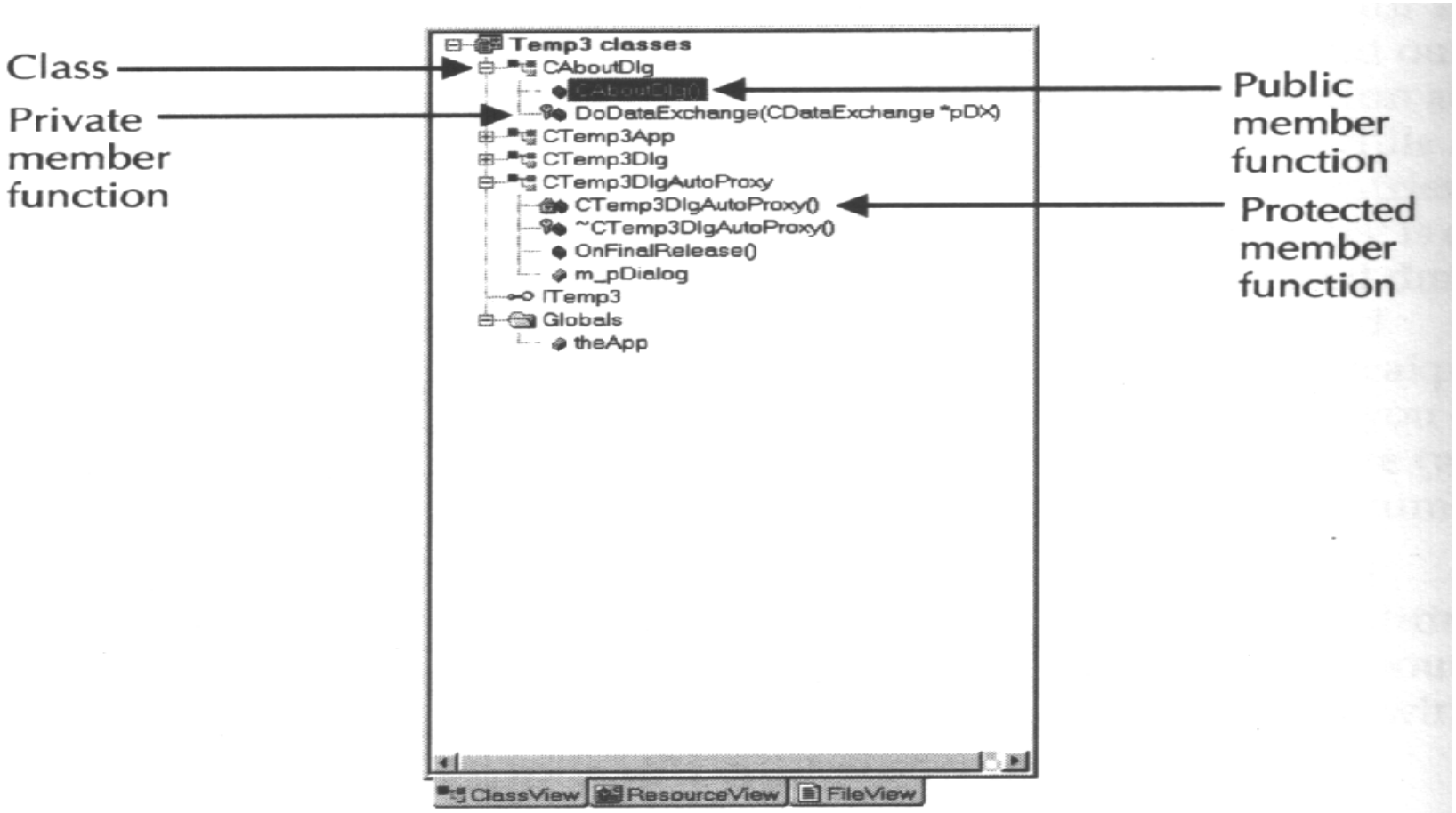


图 1.3　在 ClassView 中显示项目中所有的类

请注意，在层次列表的每个项目前面都有一个特殊的图标。例如，每个类的前面都有一个由线条连接起来的三个方框形图标。类视图中显示三种成员函数：第一种是公共成员函数，它的图标为一个紫色菱形框；第二种是私有成员函数，它的图标上也有个菱形框，不过菱形框的旁边还有一把钥匙；第三种是

保护成员函数，它的图标上有个菱形框和一个小锁。与此相似，变量也有三种类型，它们都使用青绿色图标。当看到一个绿色框时，你就知道是看到了某个COM对象的方法。类视图中还使用了其它几种类型的图标，但上面所述的六种图标是创建绝大多数工程时都需要知道的图标。

> **技巧** 在ClassView窗口中右击，然后从上下文菜单中选择Hide，可以隐藏ClassView窗口（或任何其它视图，只要在这些视图中按这一步骤操作）。使用View|Workspace命令可以重新显示ClassView窗口。

资源视图（ResourceView）

资源视图（ResourceView）在层次列表中列出了工程中用到的所有资源。任何图像、字符串值以及程序所需要的其它非编程部件都可以作为资源使用。图1.4显示了一个典型的资源视图窗口。

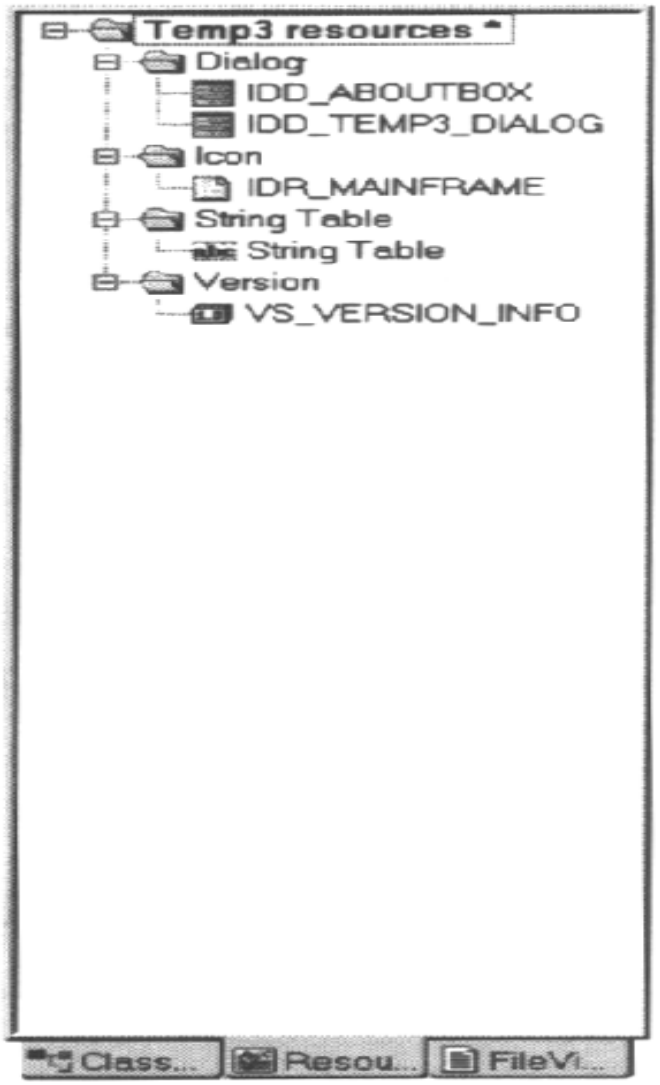


图 1.4　资源视图提供了应用程序所需资源的完整列表

Visual C++ 中可以创建的每一类资源在资源视图中都有自己的文件夹。如果你的工程中没有使用某种特定类型的资源，那么资源视图中就不会显示这种资源的文件夹。在每个文件夹中包含了工程中所用的该类资源。例如，对话框（Dialog）文件夹中包含了工程中所有的对话框，包括 About 对话框。每类资源也都使用自己的图标。

技巧 右击资源视图最顶端的文件夹，系统显示一个可以访问两类特殊对话框资源的菜单：Resource Includes 和 Resource Symbols。右击特定资源的文件夹，系统会显示添加该类新资源的上下文菜单。

文件视图（FileView）

文件视图（FileView）提供了工程中所有文件的完整列表，无论这些文件是否包含代码。图 1.5 显示了文件视图窗口的一个典型示例。请注意，列表中包含了从 ReadMe.txt 到 Visual C++ 自动生成的各种文件。

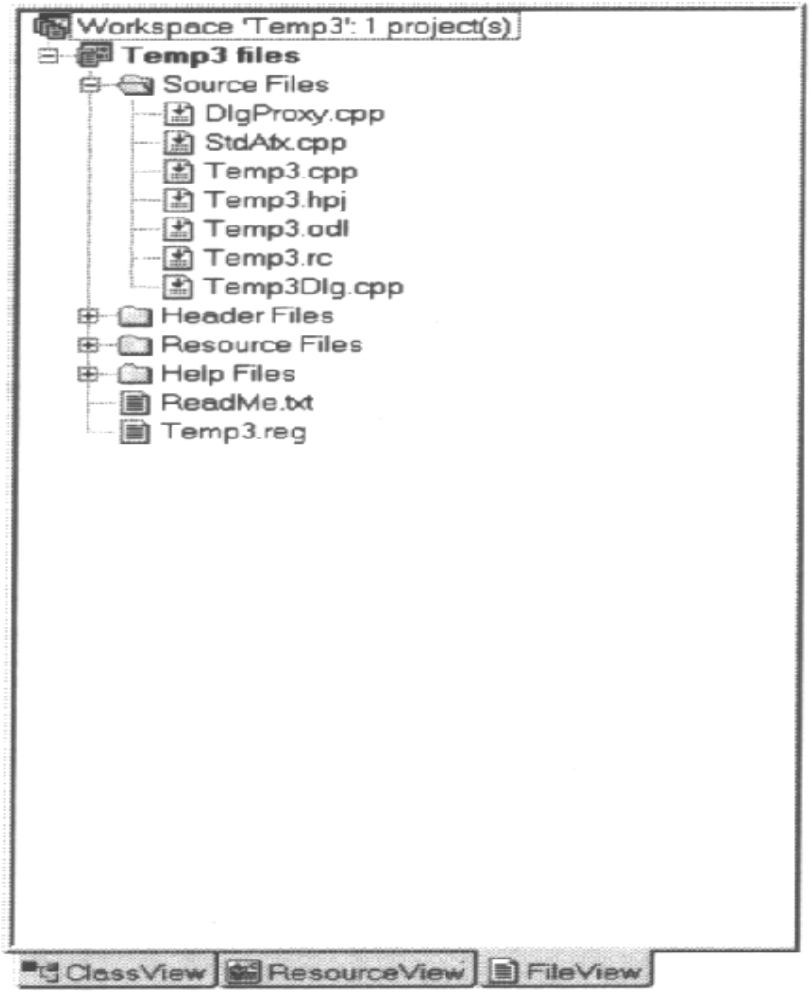
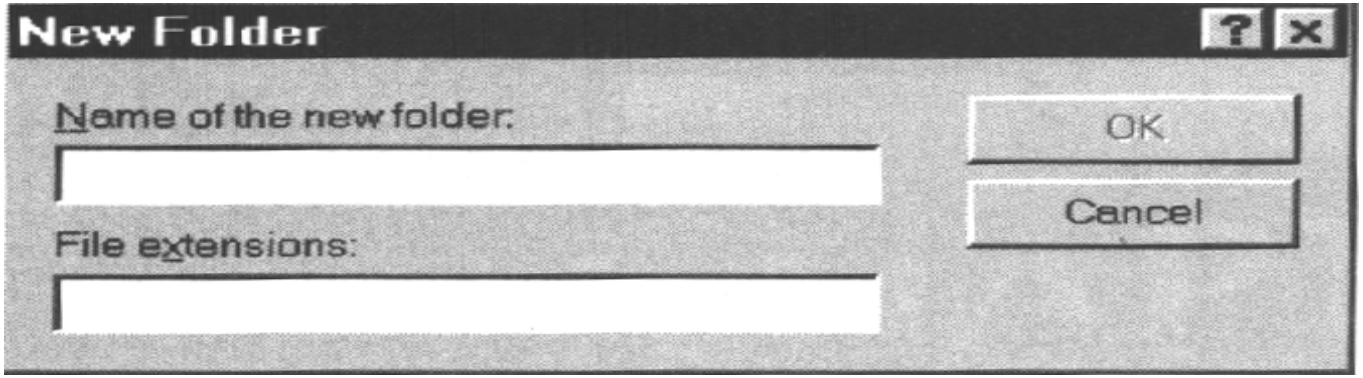


图 1.5　文件视图包含了工程中所有文件的完整列表——即使这些文件中不包含任何代码或资源

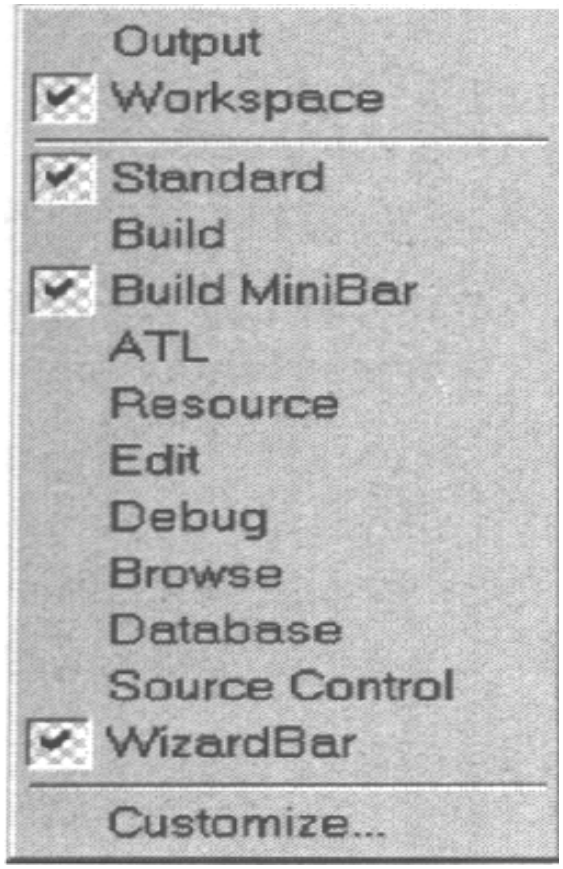
每个文件类型都有自己的文件夹，例如，所有的源代码文件都显示在 Source

Files（源文件）文件夹中。我们不仅可以把文件从一个文件夹移动到另一个文件夹中，也可以创建保存特定类型文件（根据其扩展名）的新文件夹。我通常要创建一个 Text File（文本文件）文件夹来保存所有扩展名为 TXT 的文件。创建新文件夹的方法是，右击要添加新文件夹的文件夹或工程项目，然后从上下文菜单中选择 New Folder 菜单项。此后系统将显示如下图所示的 New Folder 对话框，键入文件夹的名称以及相应的文件扩展名，然后单击 OK 按钮完成创建过程。

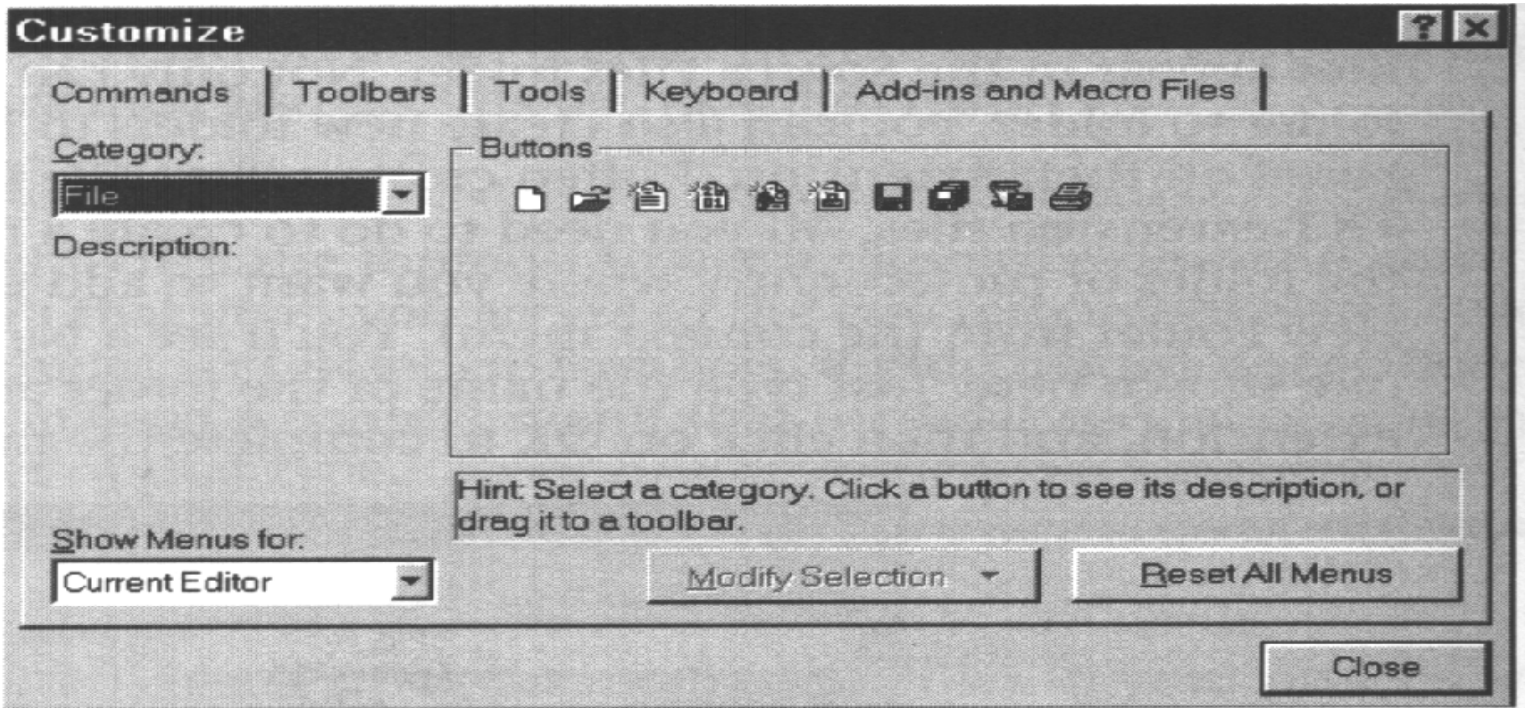


工具条

工具条（Toolbar）是我们讨论 Developer Studio 界面的最后一个话题。应该说明的是，Visual C++ 拥有比你启动 Visual C++ 后看到的工具条多得多的工具条。在工具条上的任何位置右击，你都会看到如下图所示的上下文菜单：



该菜单中列出了 Visual C++ 提供的所有标准工具条。带选中符号的菜单项对应的工具条当前显示在工具条区域中。如果想把某个菜单项对应的工具条添加到工具条区域，只需要单击该菜单项即可；反过来，如果想在工具条区域中去掉某个工具条，也只需要通过在上下文菜单中单击对应的菜单项、去掉其选中符号即可。



上下文菜单中还有一个 Customize 菜单项值得注意。选择该菜单项后你会看到如下图所示的对话框：

Customize 对话框包含了几个让你重新配置 Developer Studio 界面上绝大多数部件的选项卡。然而，这里我们最感兴趣的是 Commands 选项卡。对每个菜单和工具条它都包含了一系列的图标（代表对应的命令）或文本命令。如果想给现存工具条上增加一个命令，只需要抓住其图标（或文本命令）并把它拖曳到工具条上，这样，所选命令就显示在工具条上，你可以更快速地访问它们了。如果选择了某个现在还没有相应图标的命令，那么系统会显示 Button Appearance 对话框，在这个对话框中可以为命令选择一个图标。

同样的功能对菜单项也一样起作用。你需要做的工作就是打开要修改的下拉菜单，在 Commands 选项卡中抓住要添加到菜单中的命令，然后将其拖曳到菜单项中希望的位置。在这种情况下，你看到的总是命令的文本版本而不是图标。

从工具条或菜单中删除不需要命令的方法也很简单。简单地抓住不需要的命令，并把它拖曳到 Customize 对话框中。该命令将在菜单或工具条中消失，当然，你随时都可以重新把它添加到菜单或工具条中。

你也可以根据需要移动工具条的位置。单击工具条左边缘的双线竖杠，然后把工具条拖曳到所需位置。与此相似，如果你不喜欢某个菜单的当前位置，那么抓住它（在 Customize 对话框打开的情况下）并把它移动到所需位置。

第 2 章　建立基本应用程序

学习一门新语言，甚至更新你曾经用过一段时间的某门语言的知识，从来就不会像郊游野餐那么容易。旧的语言已经习惯，而新的语言总是包含一些不熟悉的东西，用起来别别扭扭。然而，如果你想紧跟当今商业化世界的潮流，就必须学习新的语言（或最低程度上更新老语言的知识）。

那么，如何以最短的时间跨越学习曲线底部呢？我认识的许多程序员都有一个用于学习新语言的基本应用程序。实际上，真正高水平的程序员都有几个基本的应用程序，称之为“测试套件”，其实际用途就是学习和测试新的语言。这类应用程序的功能相对比较全面，也可以再增加其它的功能，否则的话，你就不知道是否真正学会了这门语言。

上一章已经考察了 Visual C++的功能，本章打算给读者一个 Visual C++编程的初步印象。即使你是个 C++高手，也同样值得花点时间“玩一玩”Visual C++的这个新版本，这样才能确定如何使用该产品。建立这类“测试套件”将帮助你了解如何使用 Visual C++以及 Visual C++能为你做些什么。

当然，我们要花点时间建立能够使用的应用程序。为简单起见，对绝大多数情况来说，我把应用程序完整功能实现的任务留给了读者。然而，不要忘记本章的真正目的是演示 ——当样本程序与实际工作十分相似时，或许就到了抛

弃样本程序、开始实际工作的时候了。

Web 链接 任何严肃的程序员都会花点时间在 Internet 上学习新的编程技巧。Microsoft 主持了各种 Visual C++的新闻组，其中一些相当专业。最通用的新闻组是 microsoft.public.vc.language。如果你想了解 ActiveX 技术的进展情况，或许应该到 microsoft.public.vc.activextemplatelib 看一看。查找数据库信息的一个好位置是 microsoft.public.vc.database。最活跃的新闻组之一是 microsoft.public.vc.mfc，这里专门讨论 Microsoft Foundation Classes（MFC，微软基础类库），当然，还有另外两个与 MFC 相关的新闻组：microsoft.public.vc.mfc.docview 和 microsoft.public.vc.mfc.macintosh。最后，不要忘记看一看 microsoft.public.win32.programmer 下的一般 Windows 编程新闻组（它下面有许多与程序员有关的新闻组，因此，你需要选择最适合自己需要的新闻组）。

2.1 了解应用程序类型

Visual C++有能力创建你能够想像到（以及有些你想不到）的任何应用程序。然而，从整体上看，应用程序可以分为五种类型，这就是本文首先要集中讨论的内容：

- 控制台应用程序适用于你真正需要与遗留系统保持某种兼容性或不需要为用户提供全功能操作界面的情况。
- 基于对话框的应用程序通常是实用程序的专利，也适用于极小型不需要菜单系统的应用程序。
- 单文档应用程序适用于操作自有数据的简单应用程序，比如记事本或小型数据库前端应用。这类应用程序也需要某种类型的菜单系统。
- 多文档应用程序是提供完整功能的应用程序，比如字处理程序或电子表格。由于多文档应用程序是 C++编程中十分复杂的部分，因此，当你考虑建立这类应用程序时，应该在 Visual C++的灵活性与诸如 Visual Basic 之类的快速应用开发工具提供的速度之间进行权衡。
- 基于 HTML 的应用程序是 Visual C++ 6.0 新增的应用程序类型。它们操作某种类型的数据（像单文档或多文档应用程序那样），但与 Internet 结合在了一起。作为标准编辑器的替代物，你的用户会看到 Web 浏览器风格的前端应用。

注释 请注意本章我们在讨论应用程序。Visual C++有能力创建各种不同类型的代码。使用 Visual C++不仅可以创建 DLL、ActiveX 控件、ISAPI 扩展程序、设备驱动程序、像屏幕保护器之类的后台应用程序，甚至也可以扩展 Visual C++本身。本章中我们只讨论一般的应用程序，随着本书的展开，我们也会介绍许多其它类型的代码。

控制台应用程序

我前面已经介绍过控制台应用程序的概况，但由于这一介绍过于粗略，以至于并未真正告诉读者控制台应用程序到底是什么样的。控制台应用程序具备DOS风格的窗口外观，而不是读者更熟悉一些的Windows风格窗口。控制台应用程序使用等距字体，就像你在DOS窗口中看到的那样，在控制台应用程序中可以使用标准C函数完成输入输出，比如printf和scanf()。然而，从内部上讲，控制台应用程序确实是个Windows应用程序。下图是个控制台应用程序的典型示例（本章后文中我们将实际创建一个这样的程序）。

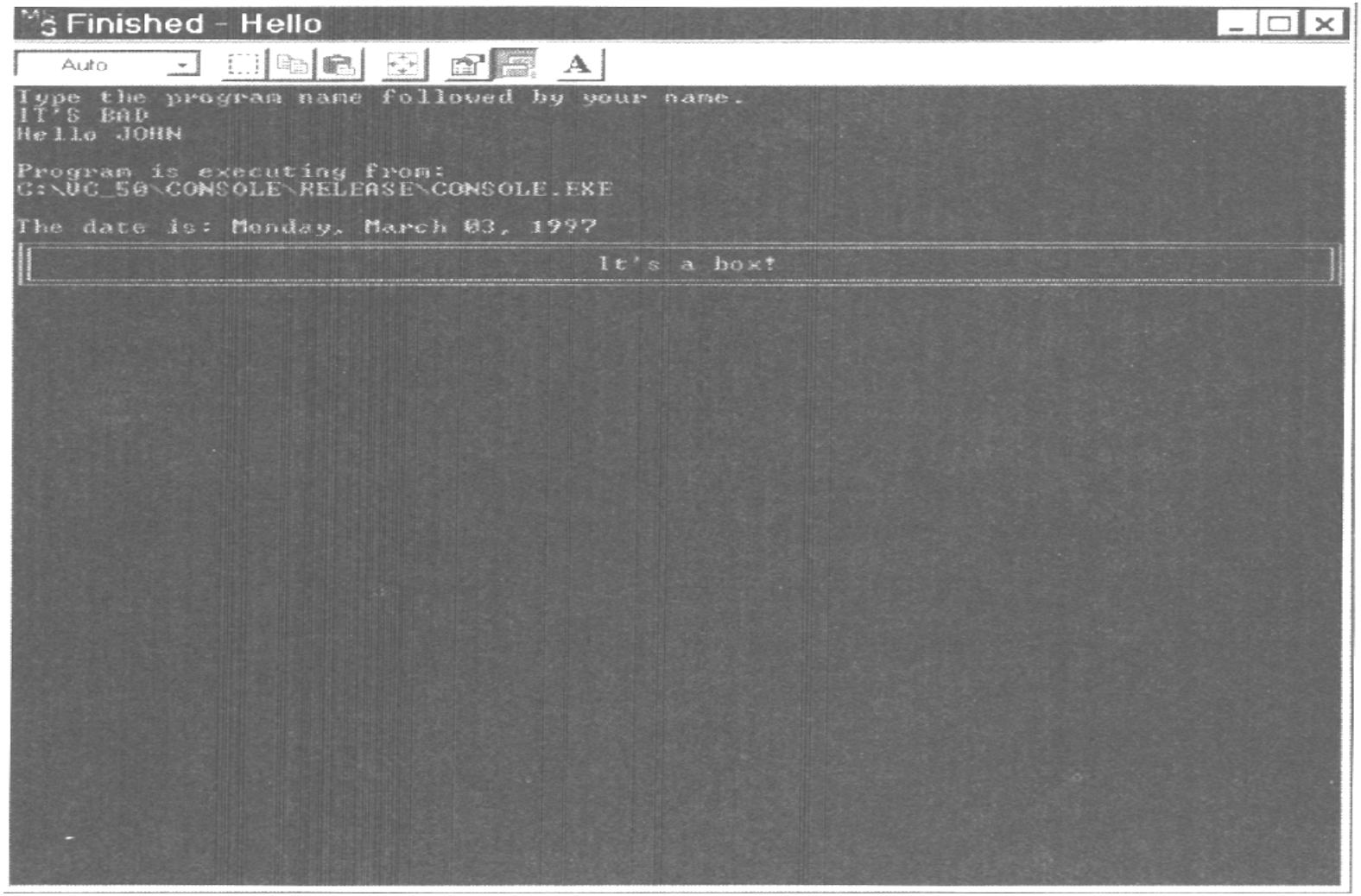
注 使用控制台应用程序可以把某些功能型DOS代码转移到Windows系统之下。

那么，为什么要创建这种杂交怪物呢？假设说你有一些工作得很好的老的遗留代码，但它们是DOS下的应用程序。你想把这个程序迁移到Windows系统下，但又没有时间把所有代码都转换成Windows调用，那么控制台应用程序提供了一个折中的解决办法。在这类应用程序中你可以使用老的代码（至少是其中的一部分），但在Windows环境中执行这一程序。

注 从长远的观点看，创建控制台应用程序并不一定会节省时间和精力——没有谁能保证让你使用所有现存代码。

使用控制台应用程序还有另外一个好处。当把应用程序从DOS搬到Windows中时，外观上就不一样。没有什么办法填平两者之间的鸿沟，从用户的观点来看，你引入了一个完全不同的应用程序，需要做出计划重新培训每一个工作人员及一切相关的事情。另一方面，控制台应用程序外观上与DOS应用程序十分

相似，每个人都已经相当熟悉，因此或许你并不需要花多少精力进行配置。你会感到烦恼的唯一的一件事就是如何让每个人都习惯以不同方式运行应用程序。



然而，不要以为控制台应用程序是每个 DOS 应用程序迁移到 Windows 环境

的理想途径。本质上两种方式下你都要创建新的应用程序，其间的唯一差别是可重用代码数量，因此也表现在创建新的应用程序所耗费的时间。事实上，如果你决定采用控制台应用程序这种方法时，只要考虑一下就会发现，你终将编写两个应用程序。绝大多数公司发现他们终将承受把DOS应用程序转换到Windows应用程序所付出的代价。

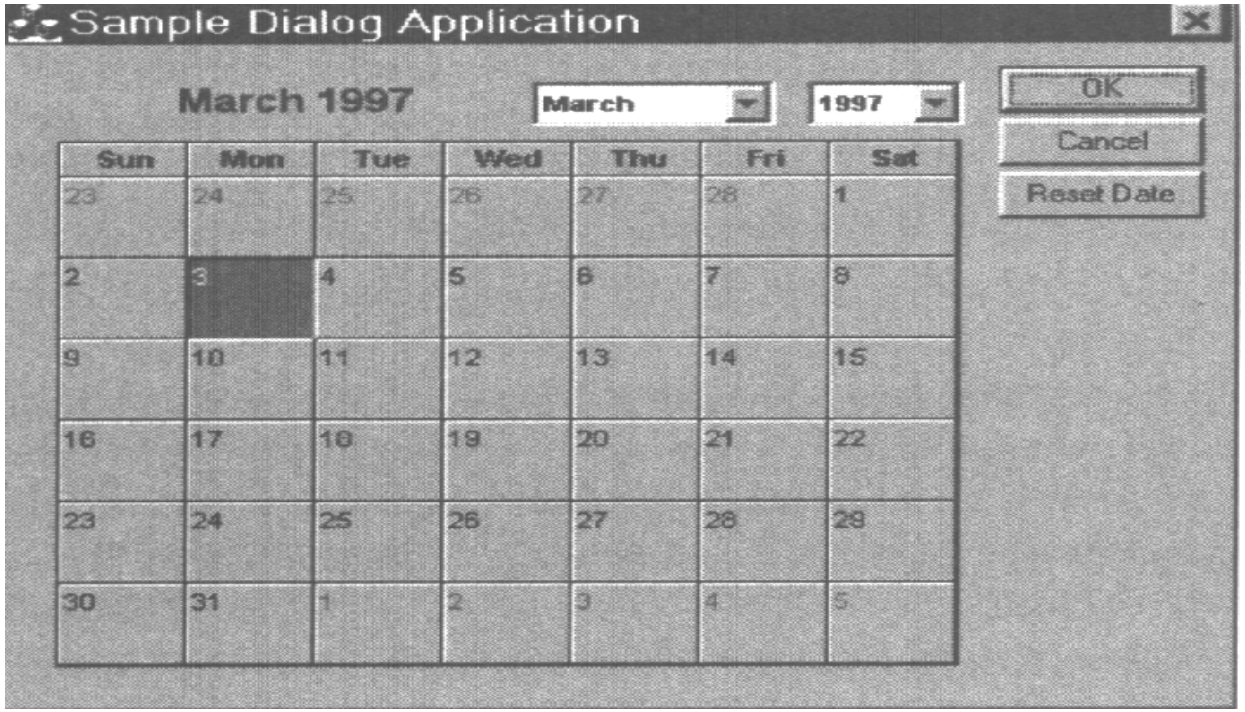
当然，如果DOS应用程序在DOS窗口中运行得相当好时，你也必须斟酌一下重新编写控制台应用程序是否明智。无疑地，控制台应用程序可以使用某些Windows服务，但或许所需的一切服务已经在DOS应用程序中实现了（你需要访问某些MFC函数以增强应用程序的功能并改进应用程序的易用性）。向你的DOS应用程序中添加越多的Windows功能，控制台应用程序的可采用性也变得越低。

可移植性 可以安全地假定使用控制台应用程序能够把DOS应用程序的“商务逻辑”迁移到Windows环境，也可以把某些显示和打印功能进行迁移。然而，想通过控制台应用程序使用一组代码同时实现DOS和Windows下应用程序的假定从来都不是安全的——这两个环境的差别实在太大了，当把应用程序从DOS迁移到Windows时总需要进行某些修改。

基于对话框的应用程序

许多人把配置屏幕、About对话框之类的应用与基于对话框的应用程序联系

起来，而把其它功能要求归类于全功能应用程序。基于对话框的应用程序在编程世界中也占有一席之地。它们对实用程序类应用程序极为有用，在这类应用程序中通常只需要显示少量的数据和获取少量的用户输入。下图是本章后面要创建的一个基于对话框的应用程序的示例：



技巧 当确定是创建基于对话框的应用程序还是创建基于窗口的应用程序时，需要考虑实用程序。如果你的应用程序适合作实用程序，那么基于对话框的应用程序界面或许是良好的选择。另一方面，如果打算在应用程序中添加众多的特性或者需要用户进行大量的交互操作，那么应该考虑选用基于窗口的界面。在做出决定时一定要考虑未来对应用程序的扩充问题——今天做出的错误选择将在明天的重新实现中付出沉重的代价。

那么，到底是什么原因使基于对话框的应用程序比基于窗口的全功能应用程序更好呢？最重要的原因之一是程序规模。你可以创建同一个应用程序的两种版本，一种使用对话框界面，另一种使用窗口界面。每次做这样的实验时你都会发现对话框版本的程序更小一些。除了节省资源外，对话框版本的应用程序的加载速度也更快些。基于对话框的应用程序比完成相同功能的基于窗口的应用程序更简单有效。

你还会发现创建基于对话框的应用程序的速度也很快。基于对话框应用程序的特点就是规模小、效率高。当你发现需要在这类应用程序中增加大量功能和特性时，或许你一开始就选错了要创建的应用程序的类型。基于对话框的应用程序通常避免使用菜单和其它基于窗口应用程序为提供友好界面所需的部件。更少的特性减少了程序员的开发和调试时间。显然，任何有利于提高程序员效率的方法都值得一试。

注 把过多的控件塞进基于对话框的应用程序的做法只能使该应用程序既笨拙又难以使用。

基于对话框的应用程序并不一定要承受缺乏重要特性的遗憾。例如，你可以把基于对话框的应用程序创建成完美的OLE服务器。在这方面Visual C++向导会为你提供相应的帮助，因此，在应用程序中添加OLE支持只需要你多做一丁点工作。

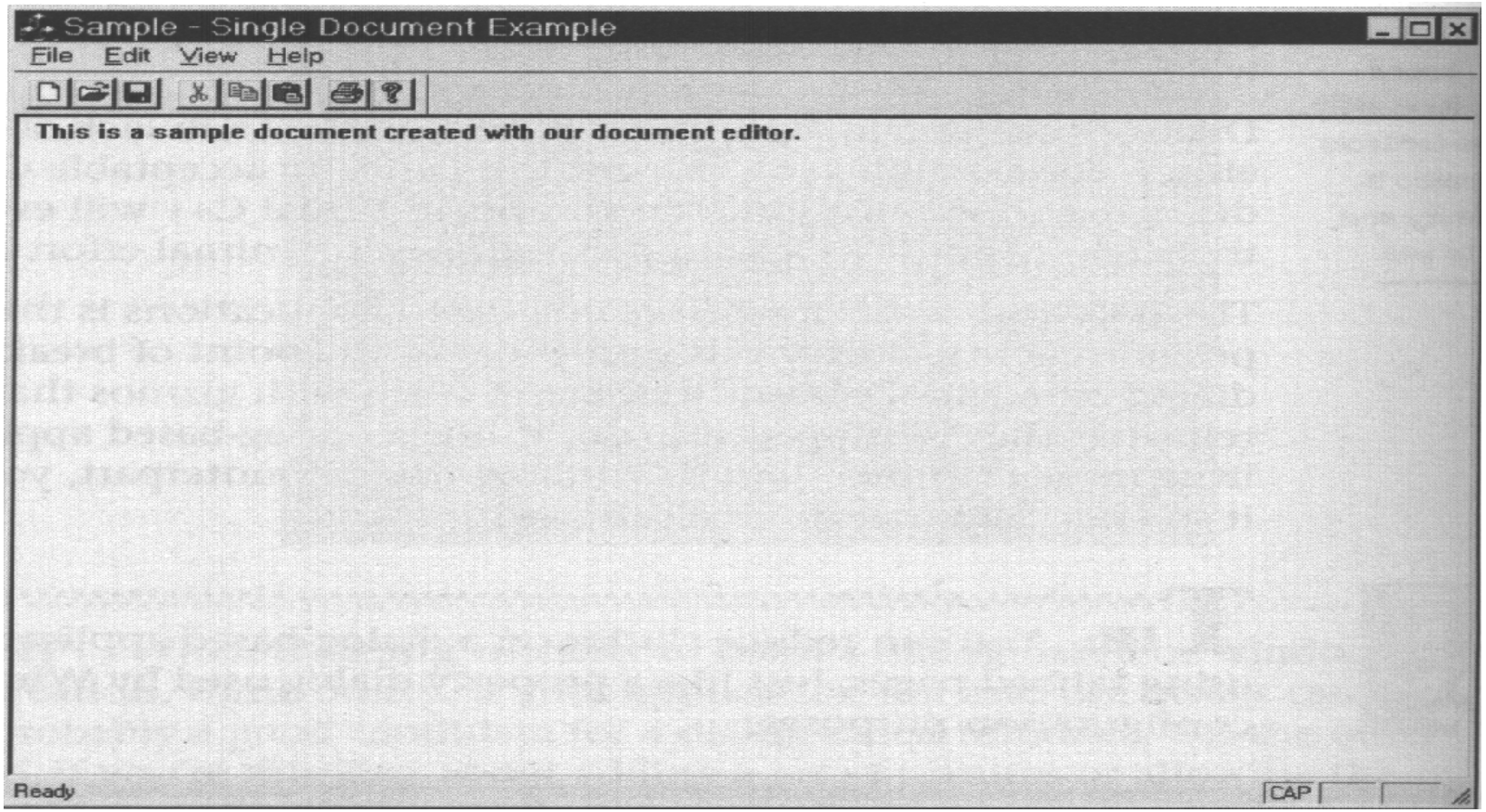
使用对话框应用程序唯一存在的问题是某些程序员感到难以划分对话框。我曾经见过一个基于对话框的应用程序，其上充满了让人不知所措的各种小玩意。基于对话框的应用程序看上去会比同等功能的基于窗口的应用程序拥挤些，但也不应该把它们拥挤到谁也不能使用的地步。

技巧 就像Windows中用于配置的属性对话框那样，你可以使用选项卡来降低基于对话框应用程序界面的拥挤程度。

单文档界面应用程序

单文档界面应用程序是像NotePad（记事本）或Microsoft Paint（画笔）这样的应用程序。它一次处理一个文档，降低了编程的复杂度并减少了运行程序时所需的资源。对某些小型应用（比如文本编辑器或小型图像编辑器）可以采用这种类型的窗口应用程序。单文档界面应用程序允许用户与其创建的文档进行全面的交互，但通常不如多文档界面的应用程序耐用。另外，单文档界面应用程序至少要比多文档界面的应用程序少一个菜单——就是用于选择要编辑文档的Windows菜单。

注 对只需要用户进行少量交互的小型应用程序来说，可以采用单文档、基于窗口的界面。



与基于对话框的应用程序相似，单文档应用程序也可以创建成 OLE 服务器。实际上，这类应用程序也可以作为 OLE 客户程序，尽管极少有程序员把这种能力添加到他的应用程序中。下图是本章后面要介绍的一个单文档应用程序示例。请注意这个示例可以作为 OLE 的客户端。

注 通过把应用程序的基视图类选择为 CHtmlView，可以把单文档界面的应用程序转换成简单的 Web 浏览器。

不幸的是，单文档界面的应用程序与基于对话框的应用程序有相同的问题——用起来太复杂了。我还记得以前使用老版本 CorelDRAW 时的问题。每当我想查看一幅图案时，不得不在查看之前首先关闭当前打开的文档。这种限制使得 CorelDRAW 比它应该提供的方法要难用一些。例如，我在比较两幅图案时浪费了太多的时间（所幸的是，Corel Systems 在当前版本的 CorelDRAW 中已经纠正了这一缺陷）。

技巧 当操作数据库管理系统时，单文档、基于窗口的应用程序工作的相当完美，其原因相当简单，极少有用户需要同时打开多个数据库。即使他们需要同时打开多个数据库，数据库本身的使用规则也减少了用户本身访问多个数据库的可能性。正常情况下，你需要以可编程方式控制对各种数据库元素的访问，并把结果显示给用户。

多文档界面应用程序

现在我们来谈谈多文档界面应用程序。使用这种类型基于窗口的应用程序可以创建像字处理程序或电子表格那样的应用程序。例如，Microsoft Word 和 Microsoft Excel 都是多文档应用程序的示例。如果你想一想，就会发现，文本编辑器具备有限吸引力的原因正是由于其一次只能打开一个文档。人们需要在文档之间进行比较，这就是多文档界面的应用程序不仅幽雅而且在众多情形下需要的原因。

注 通过把应用程序的基视图类选择为 CHtmlView，可以把多文档界面的应

用程序转换成简单的 Web 浏览器。

多文档界面的应用程序通常也都具有多种功能（你可能会走向另一个极端，看一看人们对当今厂商生产的主流产品臃肿特性的抱怨也就知道了）。文本编辑器可以提供十分简单的查找功能但并不提供替换文本的任何方法。而全功能的字处理程序则把查找和替换作为标准功能来对待。

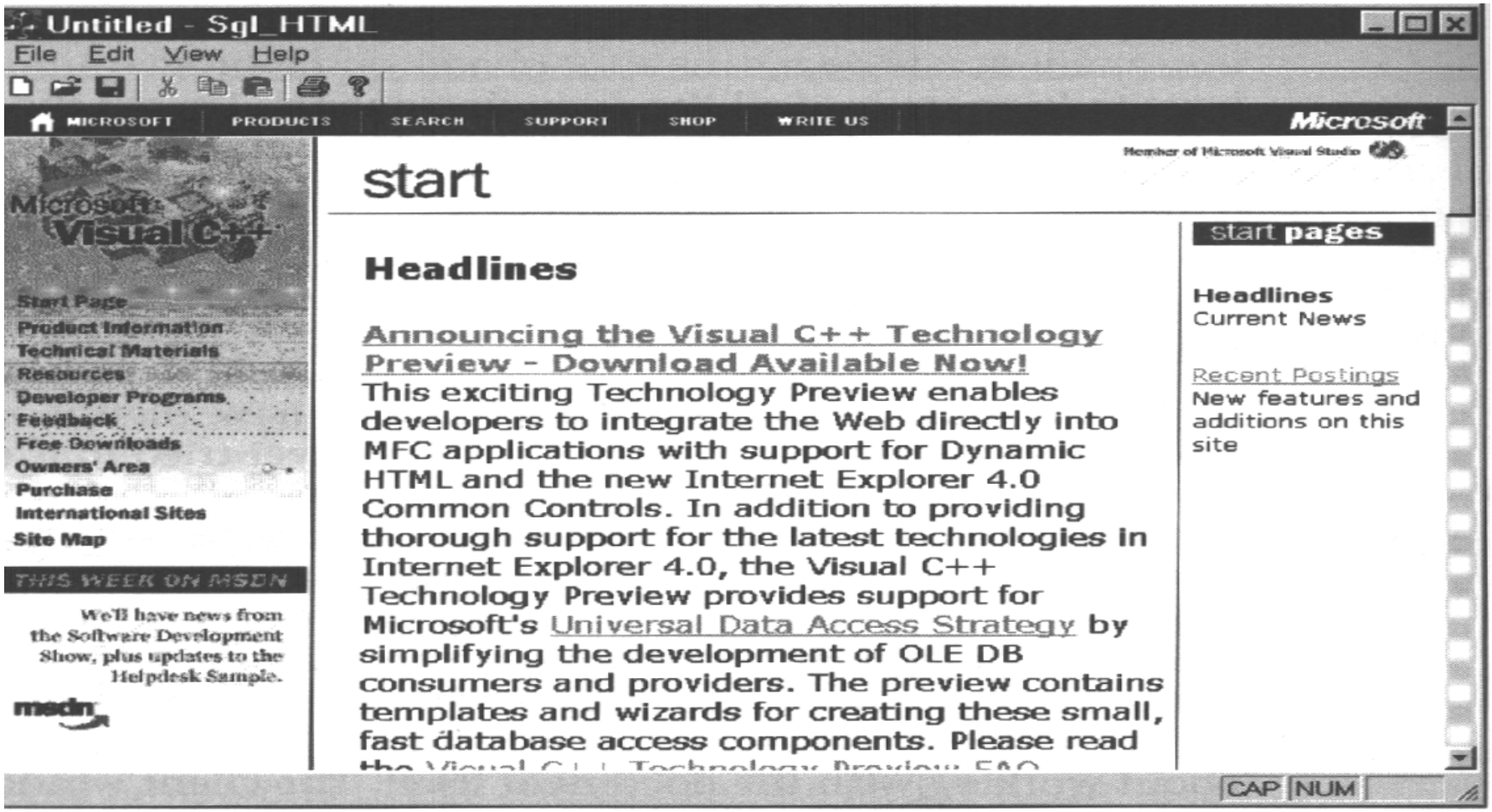
多文档界面应用程序的缺点就在于它处理多个文档。同时处理多个文档的能力也意味着需要更多的额外编程工作。你并不只是要跟踪所有打开的文档，也要提供 Windows 菜单来管理特殊的程序特性，比如要考虑屏幕划分问题。你还需要决定是否允许用户同时显示多个文档。像最小化其中一个文档，而最大化另一个文档这样的任务也需要额外的程序代码。总而言之，即使在开始编写多文档应用程序之前，就需要做大量的编程准备工作了。

当然，多文档界面的应用程序也有不少的缺点。例如，如果你以前把 Word 用做过 OLE 服务器，你就会知道，在另一个应用程序中单击链接之后，每次打开这个庞大的应用程序时都要等待很长时间这种烦恼了。而且，你或许也经历过内存不够的困惑。在最近之前，每当你要使用 OLE 时，你就必须有足够的内存同时运行两个应用程序（客户和服务器）。所幸的是，通过让服务器接管客户端窗口的方法，Microsoft 已经降低了这类内存需求，现在只负责操作文档本身了。客户窗口为服务器菜单和工具条提供了框架，因此不会再浪费额外的内存空间了。

基于 HTML 文档的应用程序

Visual C++ 6.0 提供了一种新的应用程序类型，但你在 New 对话框的 Project 选项卡中却找不到它的踪影。你可以通过 MFC AppWizard 第 6 步的对话框创建基于 HTML 文档的应用程序，本章中我们将花费几节的篇幅讨论这方面的内容。上述对话框 Base Class（基类）组合框中包含了一个 CHtmlView 选项，正是使用这个选项来创建这种新型的应用程序。

那么，基于 HTML 文档的应用程序有什么优点呢？想一想创建自己定制的 Web 浏览器的好处吧！你可以把这个浏览器设置成自动浏览公司 Web 站点并限制用户访问 Web 上非商务站点的方式。由于定制的浏览器不需要具备全功能浏览器所有的通用功能，因此，定制浏览器会减少内存需求和磁盘空间需求。换句话说，你可以建立一个提供浏览器所有功能、而又不存在它们所含问题的程序环境（至少在访问公司 Web 站点上是这样）。下图是个用于访问 Web 站点的基于 HTML 文档的应用程序的示例（显示的是 CHtmlView 类提供的缺省 Web 站点）。



不管怎么说，这种新型应用程序比你原先想像得更有价值。例如，你可以把 CHtmlView 类添加到现存应用程序中，让它能够访问基于 Web 服务器的帮助桌面（在第 15 章的“给你的应用程序添加基于 HTML 的帮助”一节我将介绍基于 HTML 的帮助）。作为创建标准帮助文件并把它添加到应用程序的一种替代方法，你可以创建十分专业化的 Web 浏览器，并把它置入到应用程序中。

基于 HTML 的帮助的优点十分明显。使用老的帮助文件就意味着一旦把应用程序交付给用户或在整个公司内分发后，你就不能够轻易地更新帮助文件。而更新 HTML 帮助则简单到只需要在 Web 服务器上修改文件即可。另外，使用 Microsoft Help Workshop 还需要做一些额外的工作。而基于 HTML 的帮助则既不需要编译器，也不需要特殊工具，只要有个文本编辑器即可（理论上说，在编写巨型帮助文件时，你需要个专门为操作 HTML 而设计的编辑器）。

HTML 帮助也有一些缺陷。一方面，难以在基于 HTML 的帮助中建立能够满足需要的查询功能。由于查询用户所需的信息与建立这些信息处于同等重要的位置，因此，基于 HTML 的帮助并不适合于初级用户。另外，基于 HTML 的帮助必须建立 Internet（至少为内部网）链接。如果你的公司中有许多旅途中的用户，那么试图建立 Internet 链接或许并不现实。当然，总可以复制所需的 HTML 文件，但这与以前的帮助形式存在相同的问题：过期的帮助文件。

这种新的应用程序类型还有许多其它的用途——多的这里无法罗列。需要记住的重要一点是，从 Internet 访问中能够得到好处的应用程序从 CHtmlView 类的使用中也会得到好处。利用这个类，可以完成从销售代表处远程更新公司数据库到让用户易于登记产品之类的一切任务。换句话说，CHtmlView 类为你和你的用户打开了一个新的世界。

2.2 编写控制台应用程序

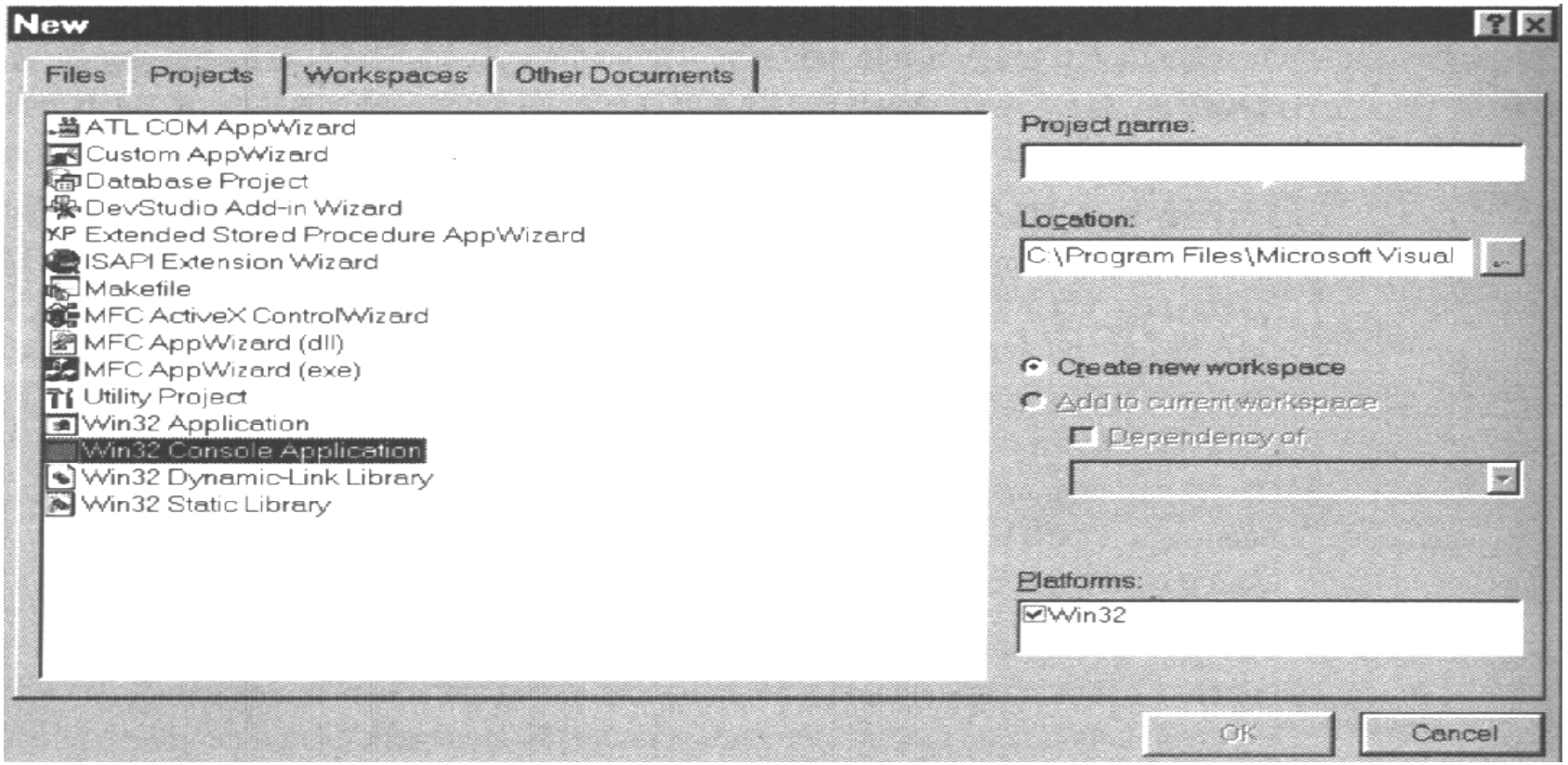
如前所述，控制台应用程序让你能够把应用程序的商务逻辑从 DOS 环

境迁移到 Windows 环境。你也可以迁移某些（甚至是绝大多数）显示逻辑，但也需要一些 MFC 提供的功能。本质上讲，控制台应用程序就像添加了一些特性的 DOS 应用程序。编码之后，需要彻底地测试控制台应用程序，以确保从 DOS 迁移到 Windows 的各种特性依然能够正确地工作。

让我们探讨一个相当简单的控制台应用程序，看看在这个程序中你能做些什么。这个示例中，我们并不把程序的功能看得很重要，只是要了解应该如何完成示例。当然，第一步的工作是创建程序框架。按下述步骤进行操作：

1. 如果你还没有启动 Visual C++，那么启动 Visual C++。

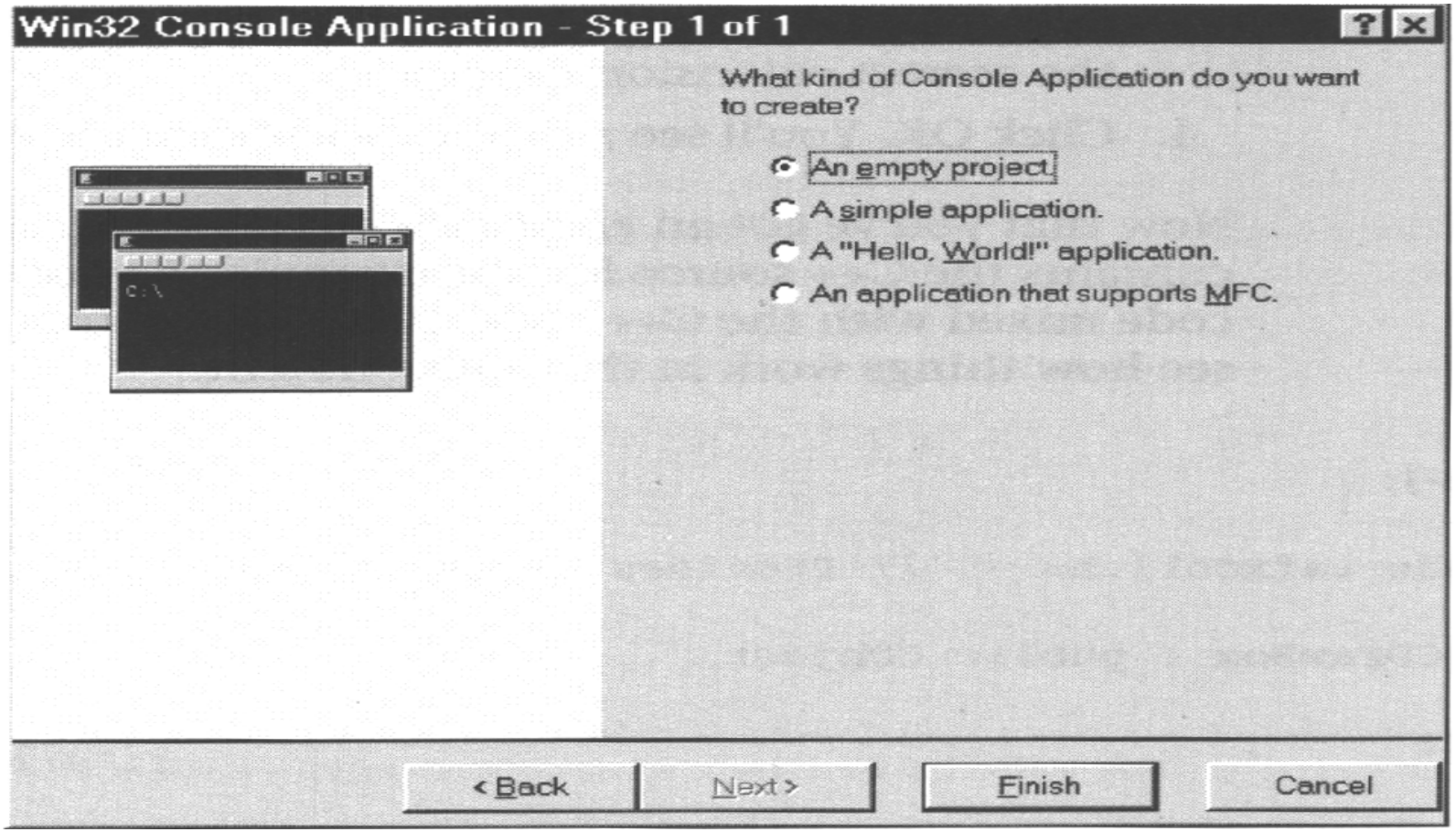
2. 使用 File|New 命令显示如下图所示的 New 对话框。注意，我已经选择了 Projects 选项卡并加亮了要在本例中使用的工程类型。



3. 当选择了 Win32 Console Application 后，在 Project Name 域中键入程序名称。这个样本程序使用的名称为 Console。你或许还需要修改一下 Location 域的内容，那么单击一下该域旁边的浏览按钮，系统会显示 Choose Directory 对话框，在这个对话框中选择应用程序的存放目录。

4. 单击 OK 按钮。你会看到 Win32 Console Application - Step 1 of 1 对话框，如下图所示。注意，在这个对话框中有几个应用程序类型供选择。这也是 Visual C++ 6.0 的一个新特性。以前版本的 Visual C++简单地创建一个空的工程。现在

到了选择要创建哪种类型工程的时候了（即使空工程也要进行选择）。



5. （必要时）选择 An Empty Project，然后单击 Finish。你会看到一个 New Project Information 对话框，它告诉你都选择了哪些选项。

6. 单击 OK 按钮创建示例程序。

在该工程真正能够运行之前，还需要完成其它一些步骤。该工程需要 MFC 类。当你把对该类的支持加入到上面的应用程序向导的设置中时，再次使用该

向导把 DOS 应用程序转换到 Windows 下时所需的重写代码的代码量就会显著减少。选择 Project | Settings 命令显示 Project Settings 对话框，选择该对话框的 General 选项卡，并在 Settings For 组合框中选择 All Configurations，在 Microsoft Foundation Classes 组合框中选择 Use MFC in a Shared DLL 选项。单击 OK 完成设置工作。

现在到了给示例程序添加代码的时候了。需要完成的第一件事是向工程中添加一个文件，让我们看看添加过程。

1. 使用 File | New 命令打开 New 对话框，选择 File 选项卡，其中显示了文件类型的完整列表，包括 Resource Template 以及像 Icon File 这样的各种图像文件。

2. 加亮 C++ Source File 选项。由于我们要在样本代码中添加一些类，因此要选择 C++源文件选项。

3. 在 File Name 域中键入 Console（Visual C++会自动添加上正确的文件扩展名）。

4. 单击 OK。系统显示一个空的 C++源文件。

现在我们有了一个要使用的空文件，该添加一些代码了。程序列表 2.1 显示了该示例的 C++源程序。注意，其中它既直接包括了一些 C 代码，也包括了一些 C++代码。我这样做的目的就是为了让读者看清楚在这种环境下系统是如何工作的。

程序列表 2.1

```
#include   <afxcoll.h>      //Provides Access to MFC functions
Class CDrawBox : public CObject
{
public :
     // Draws the box.
     void DoDraw(char * string)
};

void CDrawBox :: DoDraw(char * cValue)
{
     int        iCount;      //Loop counter
     int        iSpace;      //Amount of spaces to add for string.
     // Draw the top of the box
     fprintf(stdout,"\311");
     for (iCount = 1;   iCount <= 78 ; iCount ++)
     {
          fprintf(stdout,"\315");
     }
     fprintf(stdout,"\273");
```

```
// Figure out the center of the string, then display it
// with the box sides.
iSpace = (80 - strlen(cValue)) / 2;
fprintf(stdout,"\272");
for (iCount = 1;   iCount <=   iSpaces ; iCount ++)
{
     fprintf(stdout,"   ");
}
fprintf(stdout, "%s", cValue);
// Compensate for odd sized strings, then complete the side.
if ((strlen(cValue) % 2) == 1)
{
     iSpaces --;
}
for (iCount = 1;   iCount <=   iSpaces ; iCount ++)
{
     fprintf(stdout, "   ");
}
fprintf(stdout,"\272");

// Draw the bottom of the box
```

```
        fprintf(stdout, "\310");
        for (iCount = 1;    iCount <= 78 ; iCount ++)
        {
            fprintf(stdout, "\315");
        }
        fprintf(stdout, "\274\n");
}

int main(int    argc, char **    argv)
{
        char *          cName;      // Name of person typed at command line.
        char *          cLocale;    // Program execution location.
        CTime           oMyTime;    // A time object.
        CString         cDate;      // String used to hold time and date.
        CDrawBox    oMyDraw;    // Special text display.

        // See if we have enough command line arguments.
        if ( argc != 2)
        {
            fprintf(stderr, "Type the program name followed by your name.\n");
            return 1;
```

```
    }

    // Get the command line arguments
    cLocale = argv[0];
    cName = argv[1];

    // Get the current time and put it in a string.
    oMyTime = CTime::GetCurrentTime();
    cDate = oMyTime.Format( "%A, %B %d,%Y" );
    //Display everything we've collected.
    fprintf(stdout, "Hello %s\n\n", cName);
    fprintf(stdout, "Program is executing from: \n%s\n\n", cLocale);
    fprintf(stdout, "The date is: %s\n", cDate);

    // Use our class to draw a box around some text.
    oMyDraw.DoDraw("It's a box!");

    return 0;
}
```

如你所见，在这个示例中我展示了四个基本技巧。第一个技巧是对命令行参数适宜个数的检查，如果不满足所需个数的话，就在 stderr 设备上显示一条

出错信息，然后返回出错代码并退出应用程序。你可以在 DOS 命令行中检测这个错误代码，但对 Windows 来说根本不起作用。一旦代码检测到有足够的命令行参数，它就把这些参数放入一对变量中，用于随后的显示。

到现在为止你看到的还都是 DOS 应用程序的代码。代码中使用的第二个技巧是得到当前时间，它使用了一个 MFC 调用来完成这项任务。那么，如何才能够在控制台应用程序中访问 MFC 函数呢？正如你所看到的，我在代码的一开始就包含 AFXCOLL.H 文件，这个文件包含了控制台应用程序中调用的有限个 MFC 函数的所有定义和类定义。

不要以为你可以任意使用 MFC 调用。例如，不能创建 CDialog 对象并实际使用该对象。如果这样做的话，即使你让代码通过了编译，也可能会产生下述两种情况：或者程序以运行时错误而终止运行，或者应用程序忽略与对话框有关的代码。

技巧 在确定哪些 MFC 类可以使用时的一条重要原则是看该类是否操作图像元素。如果某个类操作图像元素的话，你就不能在控制台应用程序中使用该类。另外，你还会发现某些系统调用不再有效，并且当系统调用涉及安全问题和磁盘访问时，你还必须小心仔细地进行实验。如果存在疑问的话，那么就只好使用 AFXCOLL.H（或其它任何相关的头文件，比如 AFX.H）中列出的调用，而不使用其它调用。在任何控制台应用程序中你都可以安全地使用 AFXCOLL.H 文件中列出的所有调用。

现在我们已经得到了一些要显示的数据，随后的代码把这些数据发送到 stdout（标准输出）设备上，这就是我要向读者展示的第三个技巧。在绝大多数

t（标准输出）设备上，这就是我要向读者展示的第三个技巧。在绝大多数情况下，stdout 代表了显示，当然，如果需要的话，你也可以轻易地把输出发送到其它地方。这里的要点是使用了与以前使用格式化方式相同的格式化方式。这样，你也可以在代码中使用 fprintf()函数调用的地方使用简单一些的 fprint()函数调用。

在 main()函数中还有一个调用没有讲到，我们把数据发送到一个名称为 CDrawBox 的类中，这是我要向读者展示的第四个技巧。这个类要完成的任务就是在一个文本框中让文本居中对齐（使用 ASCII 字符集的高位字符）。你或许在许多 DOS 应用程序中见到过这种做法。这里的想法是从 MFC CObject 类中导出一个新类，然后在控制台应用程序中使用该新类。同样地，在你自己创建的应用程序也可以使用类似的技巧。正如前面我曾经介绍过的那样，当你把 DOS 应用程序迁移到 Windows 环境时，你会有一种向应用程序中添加更多功能的强烈愿望。添加功能意义的大小依赖于应用程序迁移时所耗费的时间长短以及增强功能对整个应用程序的相对价值的大小。

这个控制台应用程序示例可以开始运行了。然而，与你写过的许多 DOS 应用程序相似，为了便于测试，我们需要一个批处理文件。程序列表 2.2 是本例使用的批处理文件的源代码，它完成的任务是调用程序、测试是否出错、如果出错时反馈一条出错信息。前面你已经看到了该应用程序的输出，这里就不再重复了。

程序列表 2.2

```
@ ECHO  OFF
```

```
CONSOLE
IF ERRORLEVEL==1 ECHO IT'S BAD
CONSOLE JOHN
IF ERRORLEVEL==1 ECHO IT'S BAD
@ ECHO  ON
```

2.3 编写基于对话框的应用程序

基于对话框的应用程序十分常用于像实用程序、系统监控程序甚至向导这样的小型任务中。绝大多数情况下，它们专门用于把复杂程度降到最低限度的应用程序中。实际上，可以安全地假定基于对话框的应用程序只使用很少的控件。

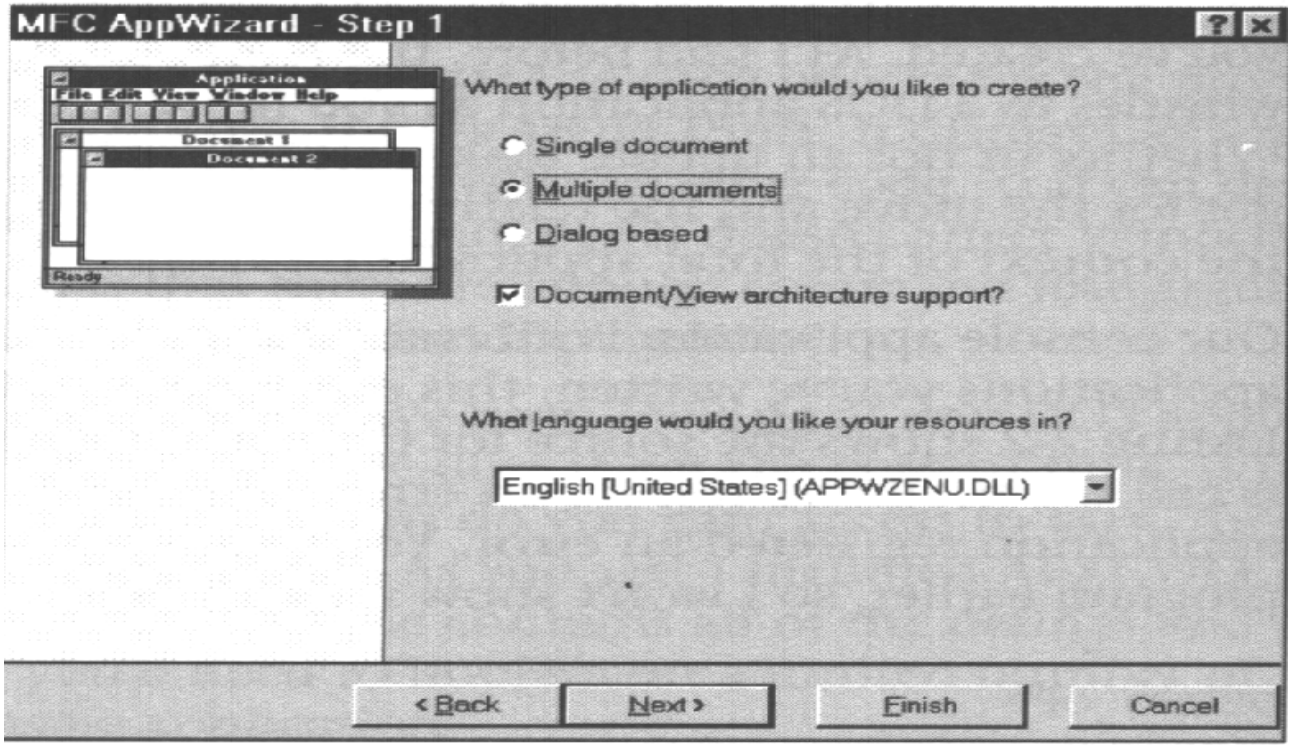
我们要看一看另外一个简单示例。在这个示例程序中，我们要组合使用ActiveX 控件和内置部件，以把所需的实际代码量减少到最低限度。下面的步骤将帮助你建立一个空的结构，随后再填充代码。

1. 打开 Visual C++（如果还没有打开的话）。

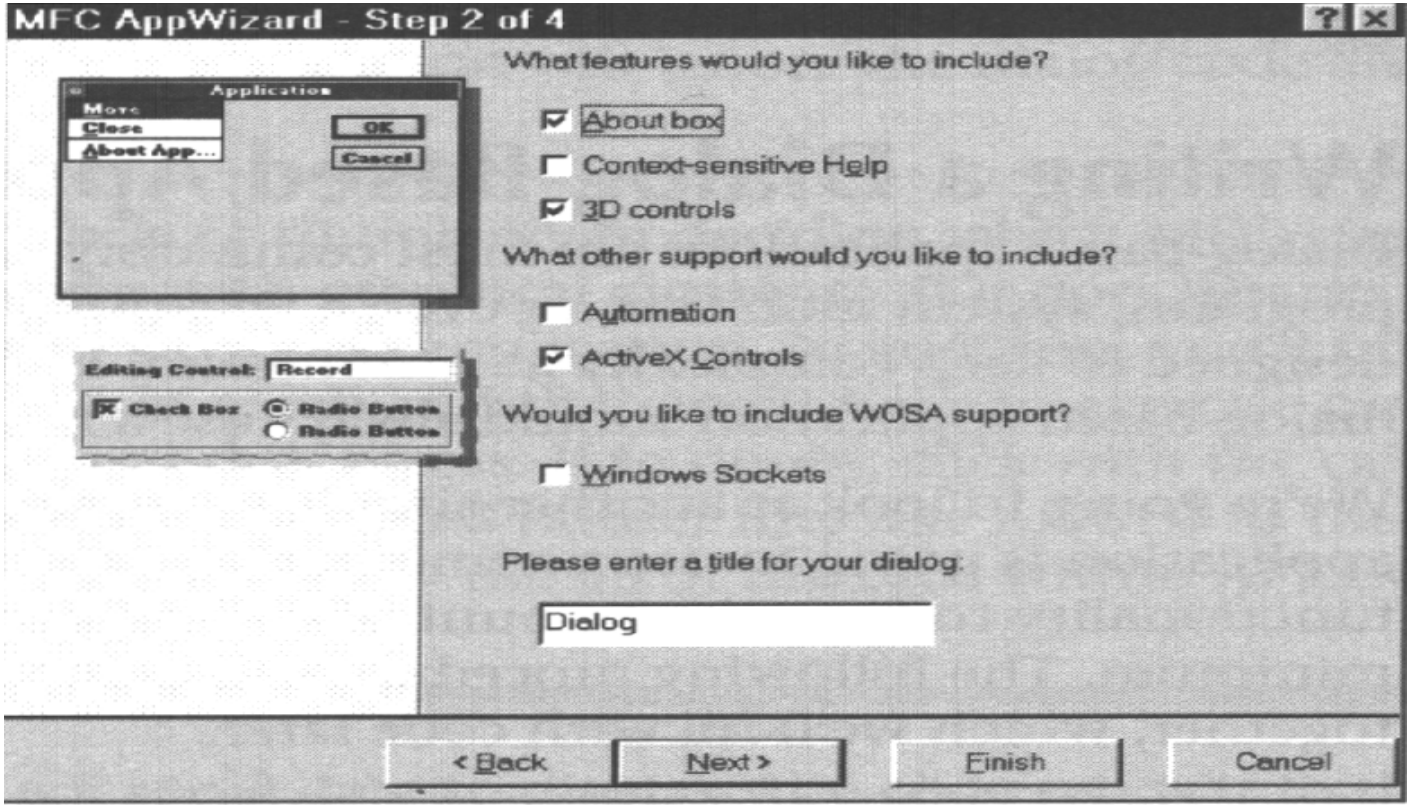
2. 使用 File|New 命令打开 New 对话框，然后选择 Project 选项卡。本例中，我们将在工程类型列表中选择 MFC AppWizard（EXE）工程类型。

3. 在 Project Name 域中键入应用程序的名称。本样本程序使用的名称为 Dialog，但你也可以使用任何自己喜欢的名称。必要时修改 Location 域的值（单击 Location 域旁边的按钮进行选择）。

4. 单击 OK 按钮，你会看到如下图所示的 MFC AppWizard - Step 1 对话框。



5. 选择 Dialog Based 选项按钮，然后单击 Next，系统显示如下图所示的 MFC AppWizard - Step 2 of 4 对话框。

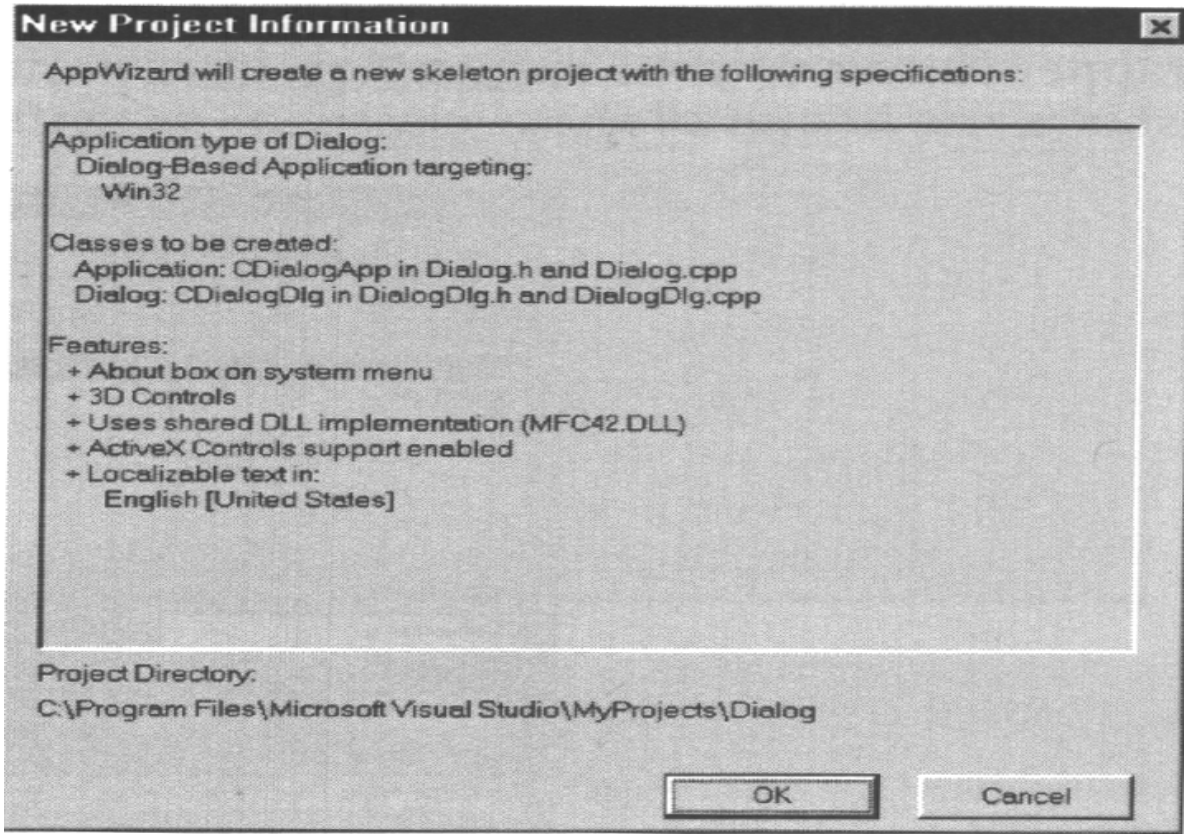


6. 在 Please Enter a Title for Your Dialog 域中键入 Sample Dialog Application。注意，在这个对话框中可以给应用程序增加一些其它特性，比如上下文相关帮助等。Automation 复选框允许把 OLE 自动化功能添加到应用程序中。你也可以在这个对话框中对应用程序增加 Windows Sockets 支持——该支持允许应用程序通过 TCP/IP 网络进行通信。

技巧 把 MFC 静态链接到应用程序中可以减少发布应用程序时必须分发的文件的数量。实际上，如果愿意的话，你可以只向其他人提供应用程序的可执行文件。这种方式也提高了应用程序在各个所安装机器上正常运行的机会，原因在于应用程序总是访问设计应用程序时使用的 MFC 版本。静态链接的缺点是应用程序变得更加庞大，而且在加载时占用更多的内存。另外，无论什么时候，当你想给应用程序增加新的特性时，都必须重新链接应用程序，不久这种方式就会让人感到厌烦。

7. 单击 Next，然后再单击 Next。我们不需要改变这两个对话框中的设置，但应该知道它们的作用。前一个对话框让你选择是否要在代码中添加注释。你还可以选择 MFC 是静态还是动态链接到应用程序中。

8. 单击 Finish。Visual C++显示如下图所示的 New Project Information 对话框。这是你确保各种设置都正确无误的最后一次机会。不正确的设置选择将延长工程的开发时间而不是缩短开发时间。

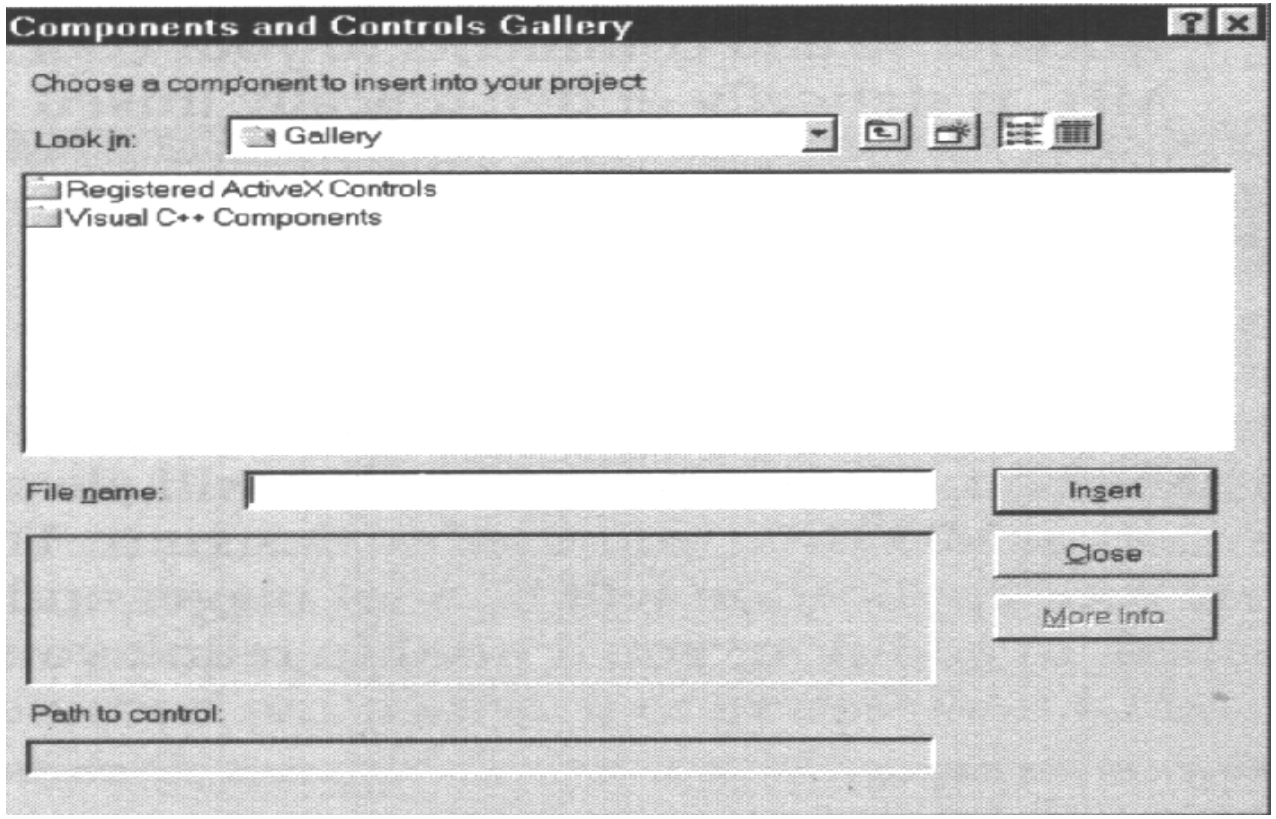


9. 单击 OK。MFC AppWizard 将生成应用程序的框架。

现在我们已经得到了一个可以使用的基本程序框架，到了实际开始编写示例程序的时候了。由于这个程序要使用 Microsoft Calendar 控件，所以我们要做的第一件事就是安装所需的 ActiveX 控件。使用 Project | Add to Project | Components and Controls 命令显示如下图所示的 Components and Controls Gallery 对话框。

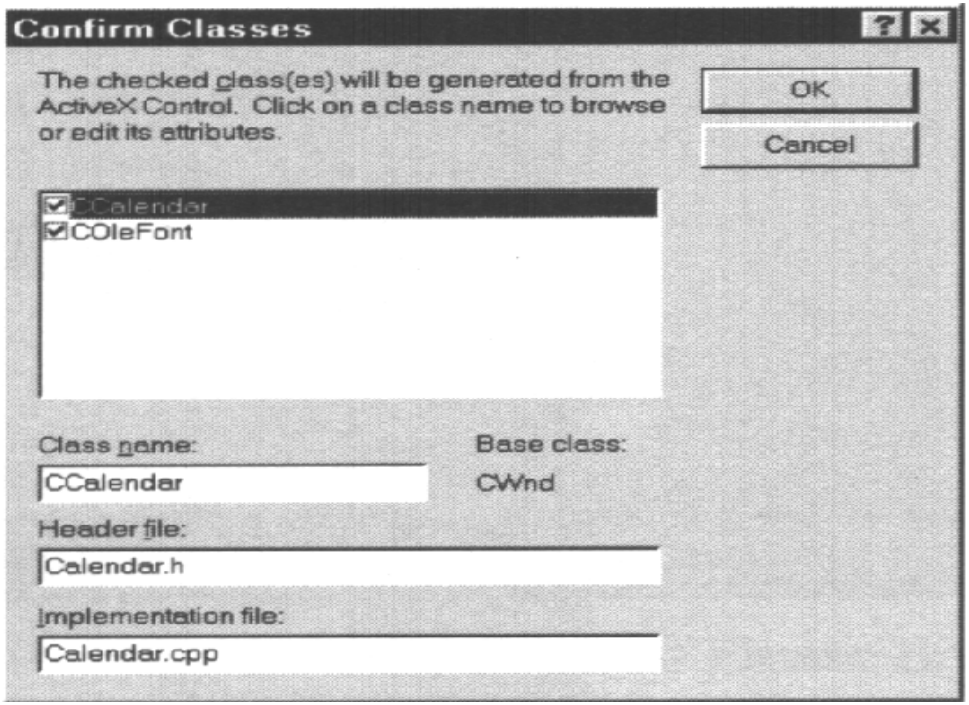
注　确保已经安装了 Visual C++携带的 OCX 控件，否则可能就没有可用的

Microsoft Calendar 控件。



双击 Registered ActiveX Controls 文件夹，你就会看到机器中已经注册的一系列控件。找到 Calendar Control 8.0 列表项（这是本版 Visual C++携带的控件之一），然后单击 Insert。Visual C++将显示一个对话框询问是否真要把这个控件插入到工程中。单击 OK。Visual C++将添加某些包装类到应用程序中，以便为其提供该 ActiveX 控件。Visual C++在如下图所示的 Confirm Classes 对话框中显示这些类的类名。

单击 OK 接受缺省的 ActiveX 控件类设置。现在我们有了一个可以使用的新控件，它被添加到了工具箱中。单击 Close 按钮关闭 Components and Controls Gallery 对话框。



现在到了设计对话框的时候了。图 2.1 显示了设计好的对话框。该对话框的当前大小为 300×200，我把 Calendar 控件的大小调整为 230×186，这样该控件上的数字就看得清楚些了。请注意，虽然该应用程序现在并没有运行，但 Calendar 控件依然立即显示当前日期，这是由于在把 ActiveX 控件放置到对话框上时，该控件自动激活了自身。如果在你把某个控件放置到对话框上时并没有发生其它反应，或许该控件没有正确地安装。显然，需要检查控件的文档以确保控件

按所期待的方式工作。

技巧 在屏幕右部状态条第二个方框中总能看到当前控件的大小。紧挨着该方框的左边方框中显示了所选控件相对于显示区左上角的位置。利用这两个方框可以十分精确地控制控件在对话框上的大小和位置。

如果现在编译和运行这个应用程序，你会发现它只是个半成品。日历会让你选择新的日期，单击 OK 按钮后对话框也会消失，但也只是如此，该程序现在还没有更多的其它功能。

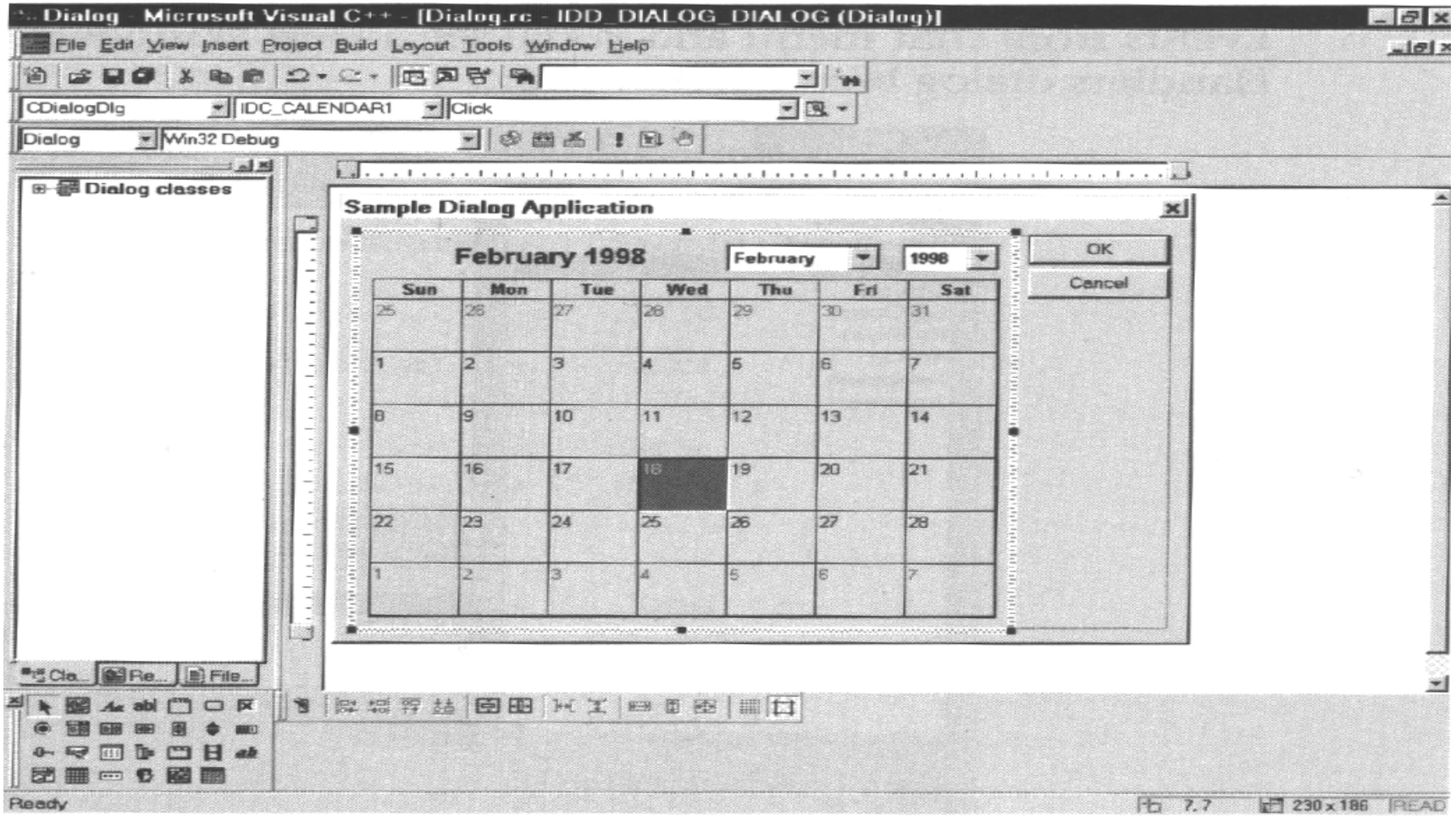
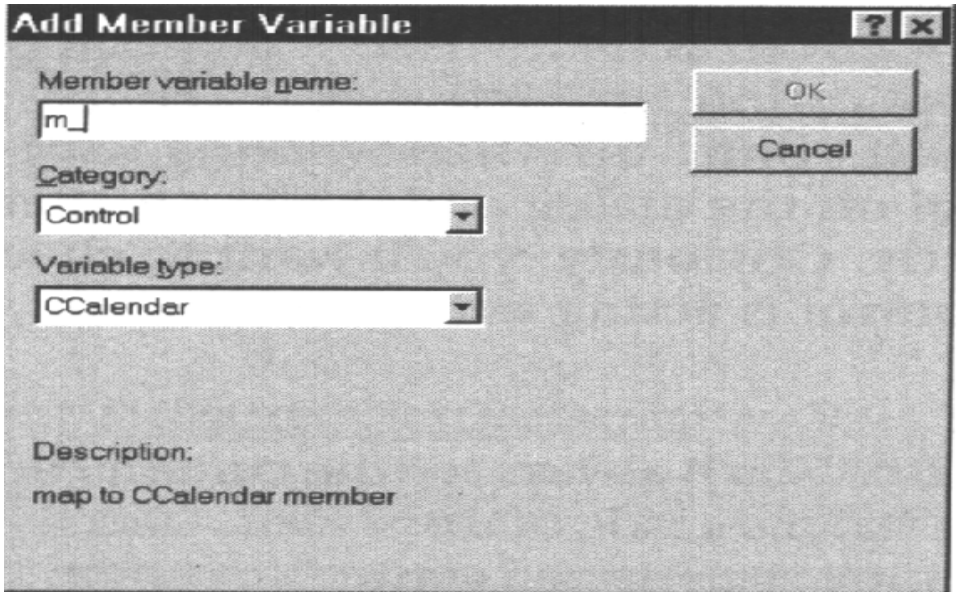


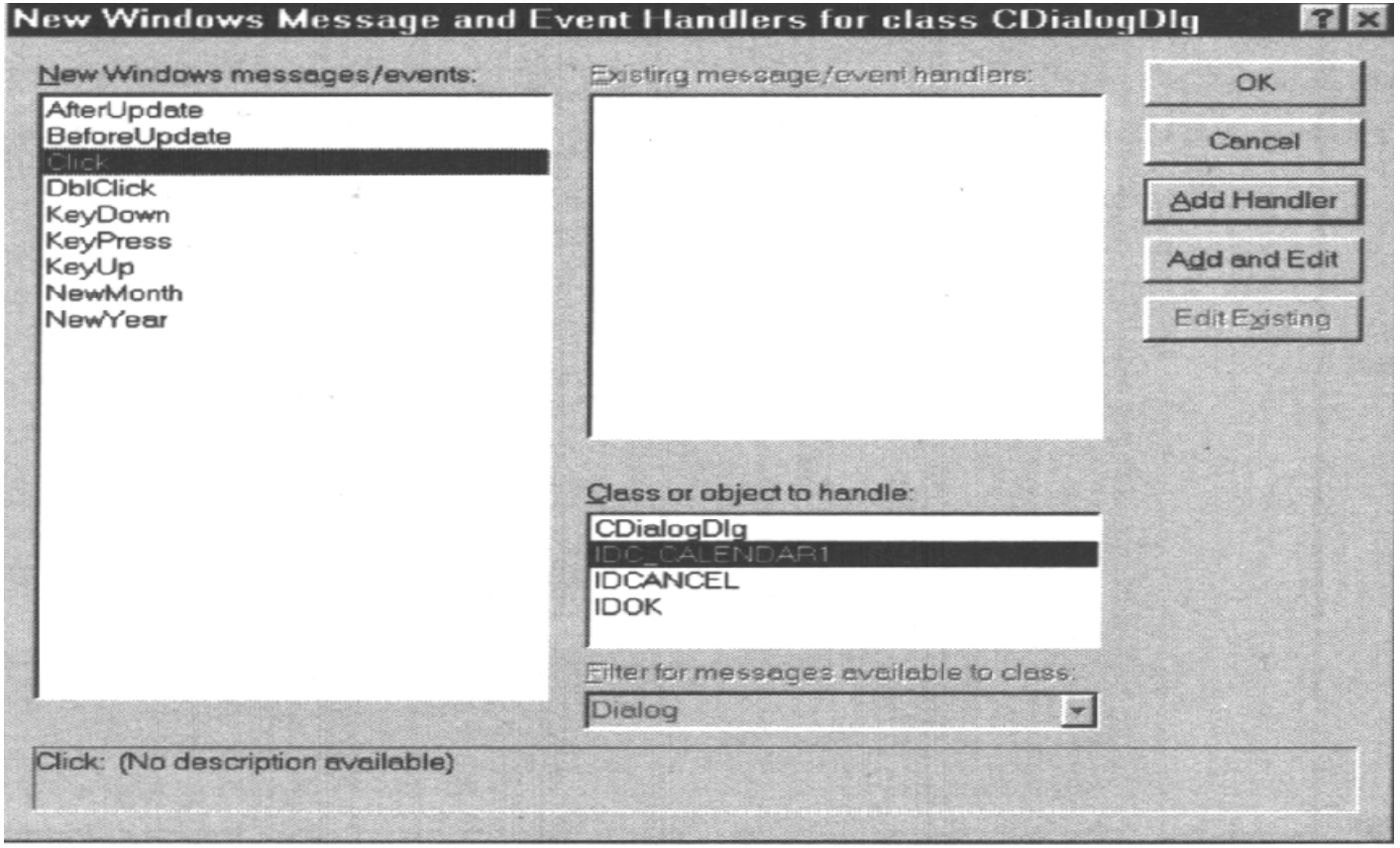
图 2.1　加入控件后，基于对话框的示例程序开始成型了

在给 Calendar 控件添加代码之前，我们还要为它创建一个成员变量。创建成员变量是件简单的事，按住 CTRL 键双击 Calendar 控件，系统显示如下图所示的 Add Member Variable 对话框。



由于该控件是应用程序中使用的第一个 Calendar 控件，因此在 m_（不要删除 m_）后键入 Calendar1，单击 OK 创建该变量。

下一步要完成的任务是检测用户是否改变了该控件。右击 Calendar 控件，你会看到一个上下文菜单，从中选择 Events，系统显示如下图所示的 New Windows Message and Event Handlers 对话框。



该对话框包含了控件可以监视的所有事件的完整列表。程序员在设计该控件时设置了这些事件。在本例的情况下，我们要监视 DblClick 事件。加亮 DblClick 事件列表项，然后单击 Add and Edit 按钮，Visual C++显示 Add Member Function 对话框。单击 OK 接受缺省的函数名，你会看到一个空的函数体。现在我们要完成的任务就是加些代码让函数正常工作。程序列表 2.3 显示了所需的程序代码。

程序列表 2.3

```
void CDialogDLg::OnDblClickCalendar1()
{
    CString     cSelectedDate;      // Date selected by user.
    Char *      cDay = "    ";      // Selected day.
    Char *      cYear = "    ";      // Selected year.

    // Get day from calendar control.
    itoa (m_Calendar1.GetDay(), cDay,10);
    cSelectedDate = cDay;

    // Get month from calendar control.
    switch (m_Calendar1.GetMonth())
    {
    case 1:
        cSelectedDate = cSelectedDate + " January ";
        break;
    case 2:
        cSelectedDate = cSelectedDate + " February ";
        break;
    case 3:
```

```
        cSelectedDate = cSelectedDate + " March ";
        break;
    case 4:
        cSelectedDate = cSelectedDate + " April ";
        break;
    case 5:
        cSelectedDate = cSelectedDate + " May ";
        break;
    case 6:
        cSelectedDate = cSelectedDate + " June ";
        break;
    case 7:
        cSelectedDate = cSelectedDate + " July ";
        break;
    case 8:
        cSelectedDate = cSelectedDate + " August ";
        break;
    case 9:
        cSelectedDate = cSelectedDate + " September ";
        break;
    case 10:
```

```
            cSelectedDate = cSelectedDate + " October ";
            break;
        case 11:
            cSelectedDate = cSelectedDate + " November ";
            break;
        case 12:
            cSelectedDate = cSelectedDate + " December ";
            break;
        }

        // Get the year.
        itoa(m_Calendar1.GetYear(), cYear,10);
        cSelectedDate = cSelectedDate + cYear;

        //Display the date.
        AfxMessageBox("You double-clicked on:" + cSelectedDate , MB_OK |
            MB_INFORMATION, 0);
    }
```

现在，你可以运行该应用程序了，它也确实会完成一些任务。编译并运行这个应用程序。双击任意日期，程序显示如图 2.2 所示的对话框，该对话框以易于阅读的方式显示了你在 Calendar 控件中所选的日期。虽然这个功能现在看

起来并没有多大用处，但你有多种方法轻易地扩展这个实用程序的功能。例如，你可以在每次双击日期时显示一个弹出式的小型记事本，在记事本中记录每天的日程安排。毫无疑问，这个程序不会取代你的合同管理器，但作为一个笔记本它确实干得不错——特别是在空间有限的膝上机和你并不想让大型应用程序用尽电池的情况下。

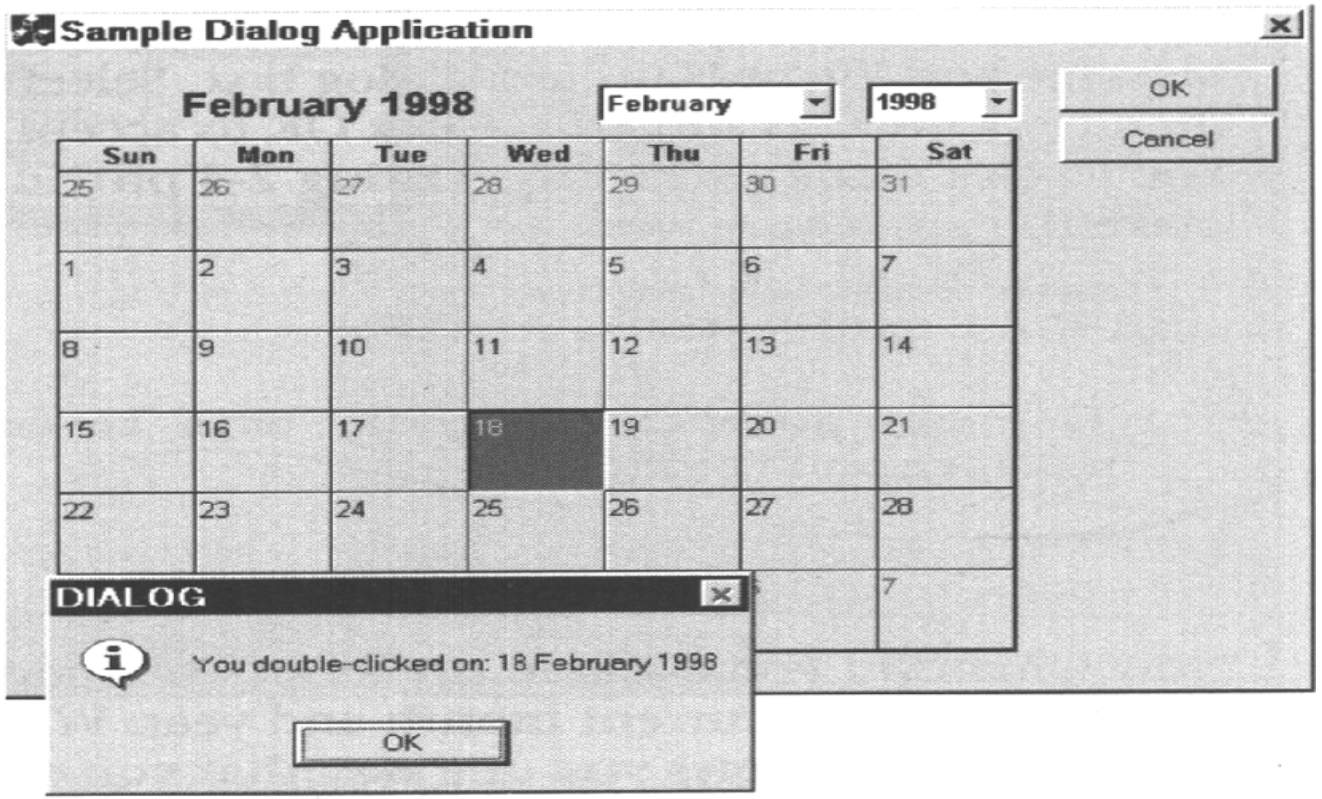


图 2.2　样本对话框应用程序显示一个对话框告诉用户它所选择的日期

Calendar 控件提供了不少的功能——绝大部分本章中都没有涉及。再做一个把日期设置回当天日期的按钮简单方便，这样，当你打电话离开一会儿时就可以用了。添加这样的按钮一点也不难，图 2.3 显示了我已经添加到现存工程中的按钮。它的 ID 为 IDC_RESET_DATE，标题为 Reset Date。如果把该按钮放置在 243,39 并把它调整到 50×40 的大小时，对话框看上去更舒服些。

既然我们有了按钮，那就再添加几行代码吧。右击该按钮并从上下文菜单中选择 Events，系统显示 New Windows Message and Event Handlers 对话框，选择 BN_CLICKED 事件，然后单击 Add and Edit，单击 OK 接受缺省的过程名，你会看到一个空的过程。程序列表 2.4 提供了要在该过程中添加的代码。

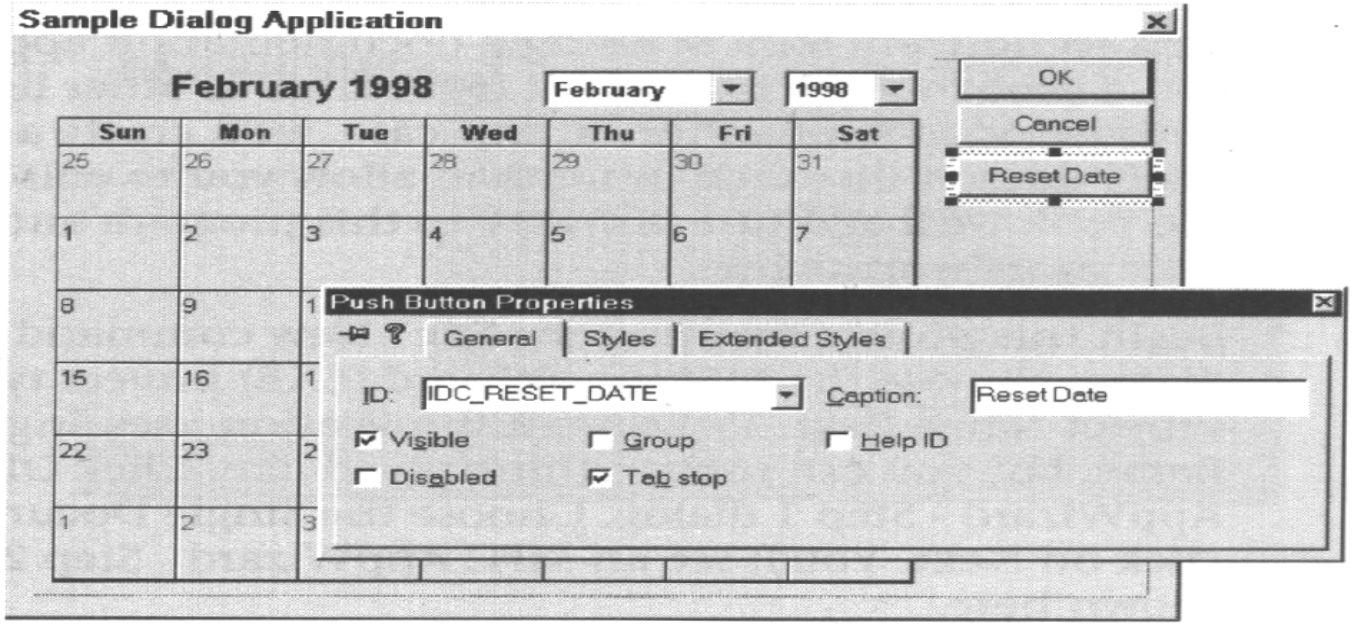


图 2.3　提供个按钮在操作了一阵日历后把日期重新设置到当前日期确实很方便

程序列表 2.4

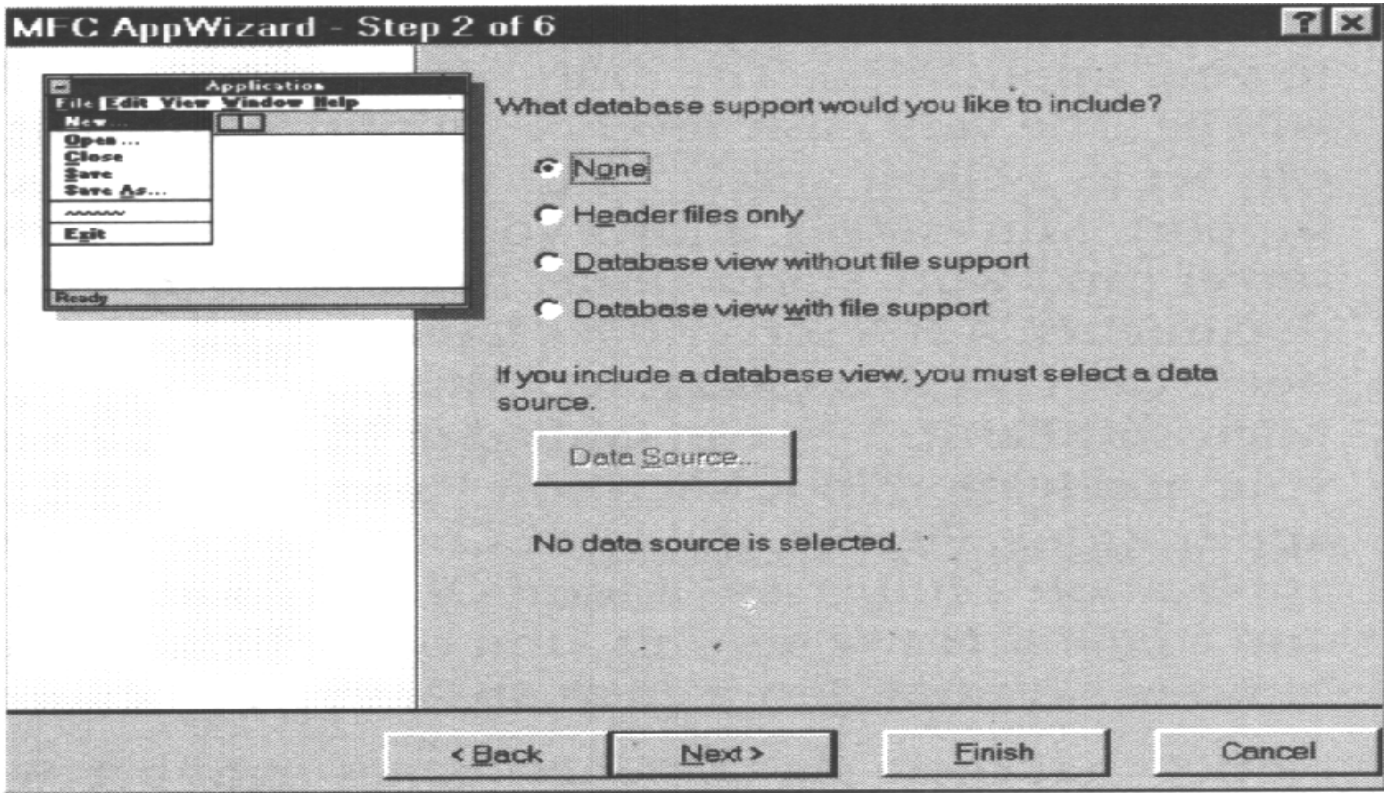
```
void CDialogDLg::OnResetDate()
{
    // Reset selected calendar date to today.
    m_calendar1.Today()
}
```

现在只要你单击一下 Reset Date 按钮，Calendar 控件就会向用户返回当前的月份和年度。虽然这段代码并不复杂，它却展示了在不增加应用程序使用复杂程度的情况下给对话框应用程序添加功能的途径。当然，即使只有少量的这种按钮过上一阵子也会让人厌倦。

2.4 编写单文档应用程序

每个版本的 Microsoft Visual C++ 向导都有轻微的改进，在你集中精力使用某个向导的过程中，设置应用程序时遗漏某些东西也是件毫不奇怪的事情。

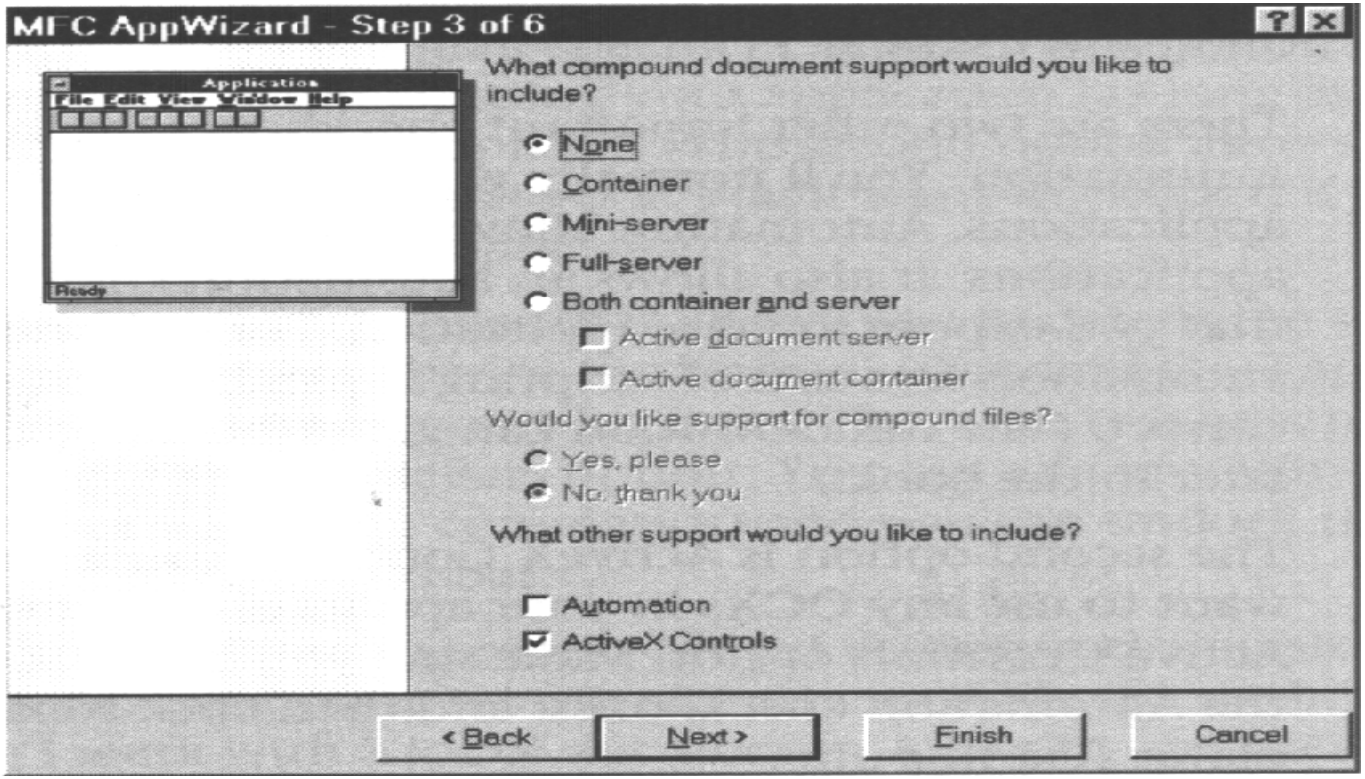
本节研究单文档应用程序——或许你已经用它编写过某个小型的文本编辑器或其它轻型、通用的文档编辑器。本例中，我们将创建一个丰富格式文本编辑器，它不需要做多少工作就可以很好地编辑文本了。在本书后续章节中，我们将给这个程序增加新的功能，使其更加有实用价值。



开始这个工程时，使用 File|New 显示 New 对话框，选择 MFC AppWizard(EXE) 工程类型，在 Project Name 域键入应用名称，示例应用中使用 Sng1_Doc 作为应用程序名称，但你也可以使用与此相似的任何名称。单击 OK，系统显示 MFC AppWizard - Step 1 对话框，选择 Single Document 选项，然后单击 Next。系统显示如下图所示的 MFC AppWizard - Step 2 of 6 对话框。

这是需要你做出决策的第一个位置。如果你想在应用程序中增加对数据库的支持，那么需要决定在 MFC AppWizard 中得到哪个层次的支持，在本书的第二部分我们将花些时间详细讨论这个问题，因此这里我就不多讲了。单击 Next，

你会看到如下图所示的 Step 3 对话框。

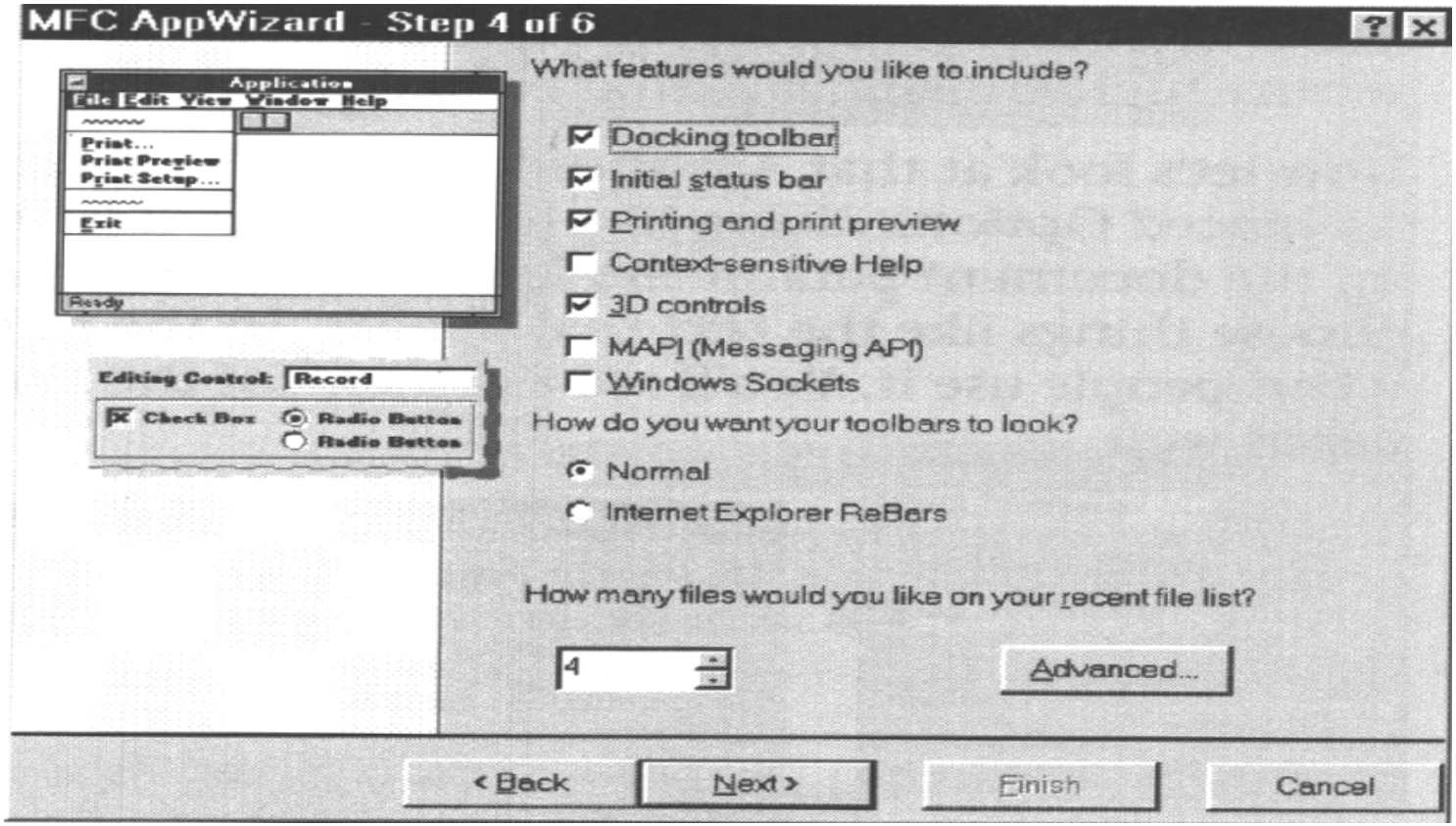


这个对话框的上半部决定了要在应用程序中添加哪个层次的 OLE 支持。添加的支持越多，应用程序也就越大。最基本的 OLE 支持层次是用做容器（Container）。容器既可以作为客户，又可以存储链接对象和嵌入对象。下一个 OLE 支持层次是 Mini-server（小型服务器），它允许创建复合文档，小型服务器不能独立工作。另外，它只能操作嵌入对象。作为服务器使用时，全功能服务器（Full-Server）确实具备了 OLE 的全部功能，但它却不能作为容器使用。这样的应用程序用起来与 Microsoft Paint（画图）十分相似，你可以把 Microsoft Paint 文档嵌入或链接到你的应用程序中，但 Microsoft Paint 本身不能存储其它

入或链接到你的应用程序中，但 Microsoft Paint 本身不能存储其它应用程序创建的对象。最后，Both Container and Server 选项让你的应用程序具备本地 OLE 支持的所有功能。你既可以把应用程序用做 OLE 服务器，也可以将它用做 OLE 客户。但是，这种应用程序不能用做 Internet 浏览器——原因在于它没有设计成 ActiveX Document 服务器。

注意在最后一个单选钮（Both Container and Server）的下方有两个复选框，如果选中第一个复选框，Visual C++还将增加使你的应用程序用做 ActiveX Document Server（ActiveX 文档服务器）所需的支持。Active Document Server 能够创建和管理 ActiveX 文档。第二个复选框是 Active Document Container，它允许你的应用程序在其框架内包含 ActiveX 文档。本质上说，它允许你在应用程序中嵌入像 Microsoft Word 或 Excel 这样的应用程序生成的文档。现在，我们选择 Both Container and Server 选项并选中 Active Document Server 复选框，在第 11 章中我们将探讨这项选择的细节。

在 Step 3 对话框中还有另外两个重要的复选框。第一个是 Automation，对大型应用程序来说通常要选中这个复选框。Automation 允许你操作其它应用程序创建的对象，也允许自动化（Automation）客户端操作你的应用程序创建的对象。从现在开始，也可以把自动化看成是一种可编程能力，虽然这种说法不甚精确（自动化的完整定义相当长，需要放开讲才能说清楚——在本书的后续章节中我们将讨论这个问题）。



第二个选项是 ActiveX Control。如果在应用程序中要使用任何 OCX 控件（无论 OCX 控件是否设计成 ActiveX 控件），那么都必须选中这个复选框。确实选中 ActiveX Control 和 Automation 复选框，然后单击 Next，系统显示如下图所示的 Step 4 对话框。

该对话框的上半部分用于选择应用程序中要包含的界面元素的类型。正常情况下，保留 Visual C++的缺省设置，这样的设置将创建标准的应用程序。如果想为用户提供上下文相关帮助，那么选中 Context-Sensitive Help 复选框。在

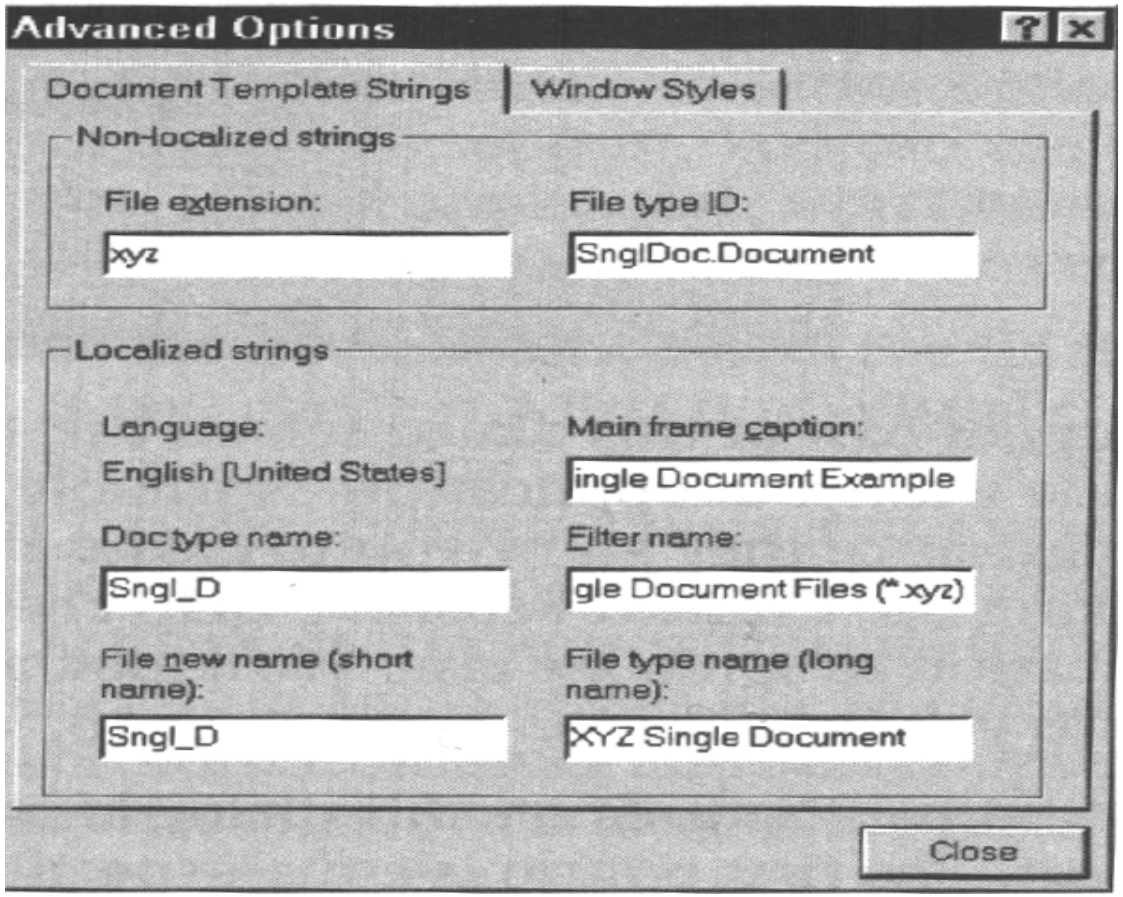
第 15 章中我们将介绍向应用程序中添加上下文相关帮助的方法。

这个对话框中下面的两个选项让你在应用程序中添加通讯支持功能。第一个选项专门用于电子邮件，选中这个选项后应用程序可以发送和接收电子邮件（只要电子邮件应用程序与 MAPI 兼容）。第二个选项让应用程序通过 TCP/IP 网络进行通讯。如果要在你的应用程序中添加 Internet 支持功能，那么选中这个复选框。

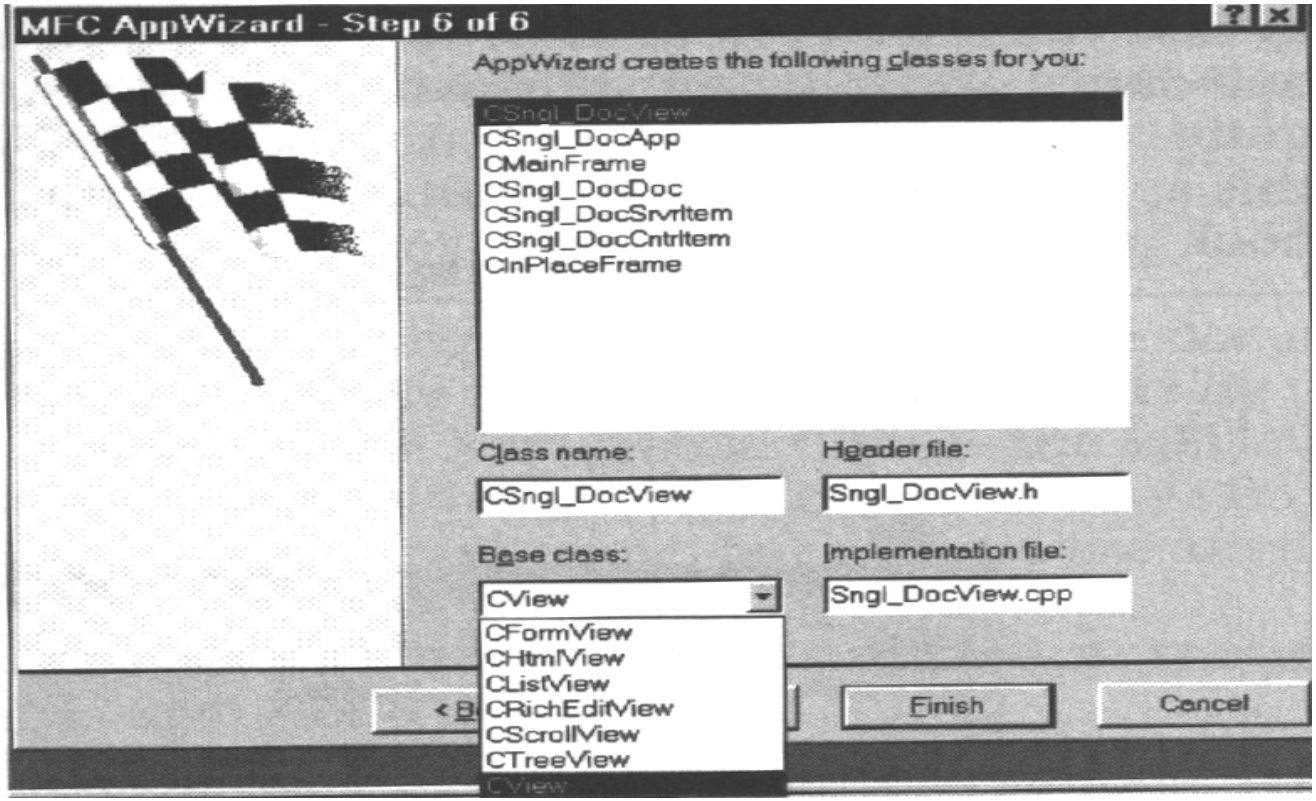
Visual C++包含的新特性让你能够选择工具条的外观。我们既可以选择 Normal 使用标准 C++工具条的外观，也可以选择 Internet Explorer 提供的新外观。使用 Internet Explorer ReBars 选项有两个好处：我们既可以在工具条上添加任何 Windows 控件（并不仅限于菜单命令），也可以像命令条那样“漂浮”ReBars。

剩下的两个选项是最近使用文件列表中列出的文件个数和一个标题为 Advanced 的神秘按钮。正常情况下，我把最近使用文件列表中列出的文件个数设置为尽可能大的值，原因在于我经常要操作许多文档。在这里给用户提供最大的灵活性并不影响应用程序的大小和速度，因此，将其设置为尽可能大的值并不是什么大是大非的问题。

现在让我们看一看 Advanced 按钮。单击这个按钮，系统显示如下图所示的 Advanced Options 对话框，这个对话框是设置文档参数的地方，也是选择用户使用时显示在标题条中文本这类设置的地方。在标题条中也将显示当前文档的名称。



按上图所示设置 Advanced Options 对话框。完成后的应用程序使用扩展名为 XYZ 的文件，应用程序的标题为“Single Document Example”，每次显示 File Open 或 File Save 对话框时，过滤器域显示为“XYZ Single Document File(*.xyz)”。最后，当显示该应用程序创建的文档的属性对话框时，对话框将显示该文档的类型为“XYZ Single Document ”。单击 Close 按钮接受上述设置，单击两次 Next，系统显示如下图所示的 Step 6 对话框。



这是我要向你介绍技巧的另一个地方。如果你没有真正留意的话，就有可能遗漏在文档视图类中可以使用你喜爱的任何基类这一事实。为什么说要重点考虑基类的选择呢？如果接受了 Visual C++的缺省设置，那么应用程序对文档的任何操作你都必须编写代码。当你要创建一种全新的文档时，上述设置并没有任何不妥，但绝大多数应用程序并非要创建一种全新的文档，而是建立一个与其它应用程序相似、并提供其它应用程序所缺乏特性的应用程序。本例中，我们要创建一个文本编辑器，因此没有必要使用 CView（应用程序的缺省基类）。

选择 CEditView 或 CRichEditView 类作为应用程序的基类将省去许多工作，原因在于该应用程序一经设计就已经知道作为简单的字处理程序应该如何完成任务。你甚至不需要添加任何代码就能得到上述功能。

为了展示基类的作用，在示例程序中我们选择 CRichEditView 作为 CSng1_DocView 类的基类。单击 Finish 按钮完成设置。当看到 New Project Information 对话框时，单击 OK，Visual C++为你创建该示例应用程序。

现在生成应用程序并启动运行。你会看到可以在应用程序中输入文本并将其保存到 XYZ 文档中。添加一些代码后，该应用程序就可以读取其它文本和 RTF 文档。当然，也可以以其它方式扩展这个应用程序。例如，由于该应用程序使用了富文本编辑控件，因此可以为其增加文本格式化功能和颜色功能。我们已经添加的 OLE 能力意味着需要时可以在文档中插入图像。已经完成的应用程序只做了很少的工作就具备了许多潜在的功能。

或许现在你还没有注意到，这个特殊的示例程序并没有要你编写一行代码。最终的程序相当完美，是 C 程序员不敢想像的。图 2.4 展示了到现在为止这个应用程序的显示结果，随着本书的展开，我们将进一步改进这个应用程序。

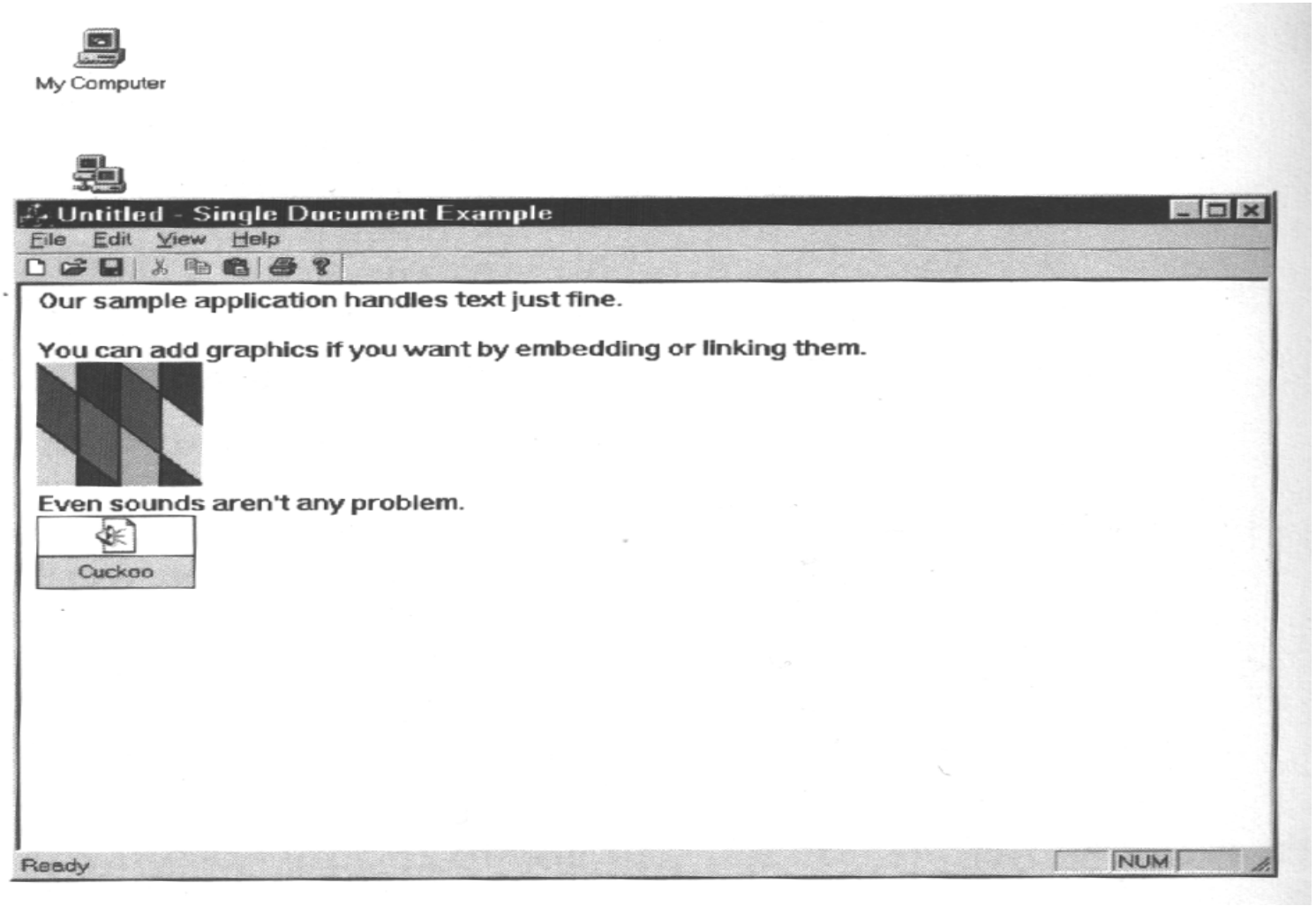


图 2.4　考虑到我们并没有给这个示例应用程序添加任何代码，应该说它运行的够完美了

2.5 编写基于 HTML 文档的应用程序

毫无疑问，Internet 已经成为许多公司降低成本、提高效率计划的一部分，另外，Internet 也是销售产品、查询信息的新方法，并能完成许多几年前人们还认为不能完成的许多商务方面的其它任务，这就是本例极为重要的原因。该示例展示了把 Internet 能力置入应用程序的方法，它把两种不同的媒体融合成了一个和谐的整体。

本节展示如何创建基于 HTML 文档的程序框架。虽然用向导生成的框架功能还有限，但它既能显示 Internet 上的 HTML 文档，也能显示不在 Internet 上的 HTML 文档。当我们拥有了可以使用的基本框架后，就可以考虑添加一些绝大多数应用程序都需要的某些增强功能了。

创建基本程序框架

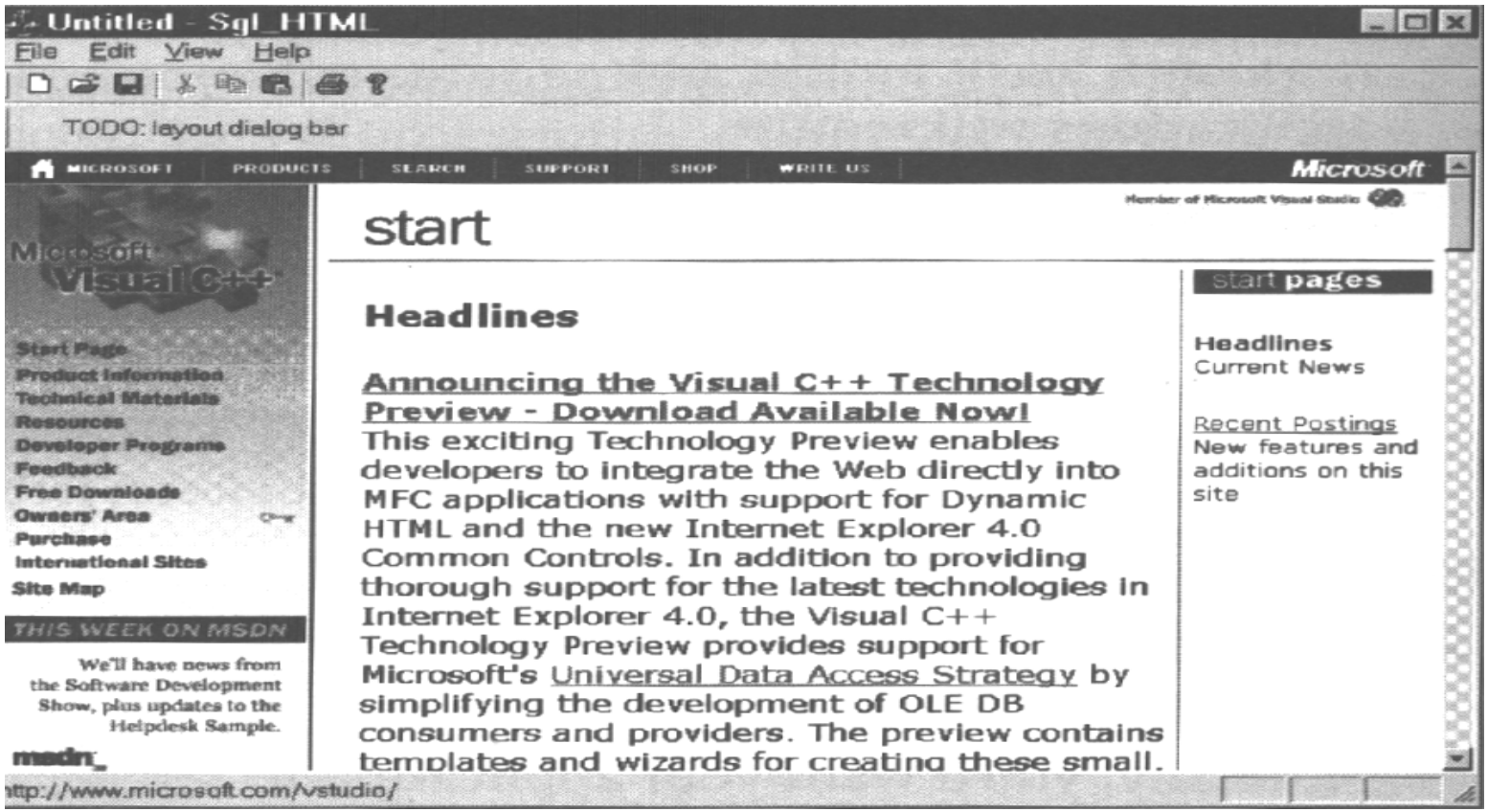
对本例来说，我们将使用本章前面介绍过的单文档应用程序的一种变体。作为本工程的开始，使用 File | New 命令显示 New 对话框，选择 MFC AppWizard(EXE)工程类型，在 Project Name 域中键入名称，示例应用程序使用的应用程序名为 Sg1_HTML，但你可以选择自己喜爱的任何名称。单击 OK 按钮，系统显示 MFC AppWizard - Step 1 对话框，选择 Single Document 选项，然后单击 Next 按钮两次，你会看到 MFC AppWizard - Step 3 对话框。

正常情况下，你会希望你的应用程序既用做 OLE 服务器又用做 OLE 容器。

但针对当前这种应用程序来说，绝大多数用户并不需要 OLE 服务器特性，原因在于只是使用这样的应用程序显示 Web 页面，而无须建立新的数据。正是出于这样的考虑，我们选择 Container 选项并选中 Active Document Container 复选框。回忆一下 Active Document Container 复选框的功能，就会知道它允许你显示像 Microsoft Word 这样的应用程序生成的 ActiveX 文档。再选中 Automation 和 ActiveX Controls 复选框，单击 Next，系统显示 MFC AppWizard - Step 4 对话框。

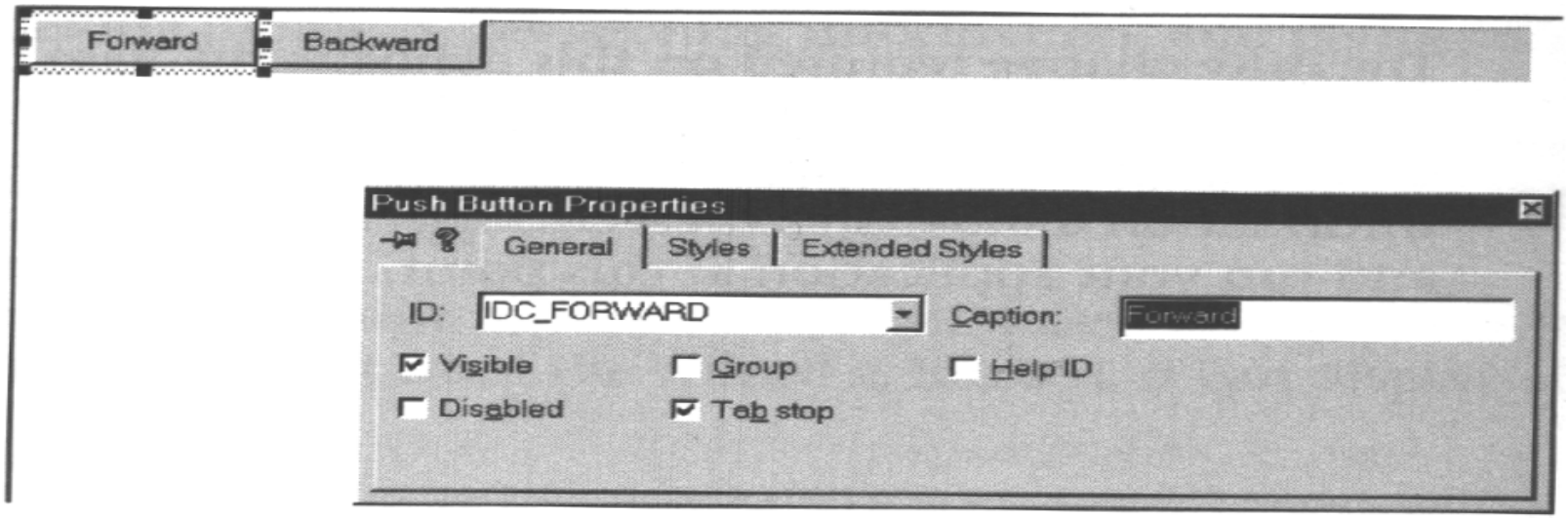
在这个对话框中你至少要看一看其中的几个选项：第一个选项是 Windows Sockets，当应用程序要访问 Internet 或内联网时，应用程序需要 Windows Socket 支持。如果应用程序还要支持电子邮件，那么也要选中 MAPI 选项。最后，由于你的应用程序或许还需要相当复杂的工具条来向用户提供全功能的导航能力，那么也要选中 Internet Explorer ReBars 选项。我还把最近使用文件列表的文件数设置成了 16。单击 Next 两次，系统显示 MFC AppWizard - Step 6 对话框。

这个对话框中唯一需要修改的选项是 Base Class，必须把这个选项设置为 CHtmlView，以便应用程序得到我们所期待的 Internet 功能。现在我们已经配置好了这个示例程序，单击 Finish 按钮，系统显示 New Project Information 对话框，然后单击 OK 生成框架应用。此时编译和运行应用程序时，你会看到如下图所示的屏幕显示。



增强示例程序功能

如果你只是打算显示你的 Web 站点的主页，那么使用上一节创建的应用程序框架就能完成这个任务。然而，在绝大多数情况下，还需要给应用程序添加一些基本的导航能力。只有单一的链接或许并不能让用户满意，因此，该应用程序至少要增加的功能是“向前”与“向后”按钮。现在就让我们看看增加这两个按钮的方法吧。



首先需要完成的任务是打开 IDR_MAINFRAME 对话框，没错，Internet Explorer ReBar 实际上是个对话框，而不是标准的工具条。该对话框的类是 CDialogBar 而不是过去我们常用的标准 CDialog 类。与其它任何对话框相同，你可以向这个对话框上添加标准的 Windows 控件，我们在这个对话框上添加了两个按钮，如下图所示。请注意，这两个按钮的标题为 Forward 和 Backward，其 ID 分别为 IDC_FORWARD 和 IDC_BACKWARD。

现在我们在 ReBar 对话框上放置了两个按钮，但还没有定义单击它们时做些什么事。实际上，如果现在就编译和运行这个应用程序，你会看到这两个按钮以灰色显示，并且什么事也做不了，其原因在于，我们告诉了 Visual C++在哪里绘制这两个按钮，但并没有告诉 Visual C++用它们来干什么。让这两个按钮发挥作用需要下述三个步骤：

1. 定义按钮与消息处理程序之间的连接。

2. 声明消息处理程序函数。

3. 编写消息处理程序代码。

让我们先完成第一步，定义按钮与消息处理程序之间的连接。由于我们希望这两个按钮影响 HTML 页面的内容，我们需要在 CSgl_HTMLView 类中使用 ON_COMMAND 宏建立连接。程序列表 2.5 以粗体形式显示了需要添加到 CSgl_HTMLView.CPP 文件中 MESSAGE_MAP 节的代码。请注意，ON_COMMAND 宏提供了这两个按钮（标识为 IDC_FORWARD 和 IDC_BACKWARD）与其对应的消息处理函数之间的连接。

程序列表 2.5

```
BEGIN_MESSAGE_MAP(CSgl_HTMLView,CHtmlView)
    //{{AFX_MSG_MAP(CSgl_HTMLView)
        // NOTE - the ClassWizard will add and remove   mapping macros here.
        // DO NOT EDIT what you see in these blocks of generated code!
    ON_WM_DESTROY()
    ON_WM_SETFOCUS()
    ON_WM_SIZE()
    ON_COMMAND(ID_OLE_INSERT_NEW,OnInsertObject)
    ON_COMMAND(ID_CANCEL_EDIT_CNTR,OnCancelEditCntr)
    // }}AFX_MSG_MAP
    // Standard printing commands
    ON_COMMAND(ID_FILE_PRINT, CHtmlView::OnFilePrint)
```

```
    ON_COMMAND(ID_FILE_PRINT_DIRECT CHtmlView::OnFilePrint)

    ON_COMMAND(ID_FILE_PRINT_PREVIEW,CHtmlView::OnFilePrintPreview)

    //Add two macros for our buttons
    ON_COMMAND(IDC_FORWARD, CSgl_HTMLView::OnForward)
    ON_COMMAND(IDC_BACKWARD, CSgl_HTMLView::OnBackward)
    END_MESSAGE_MAP()
```

当我们定义了这两个按钮与其对应的消息处理程序之间的连接后，需要在CSgl_HTMLView.H 文件中声明相应的函数。程序列表 2.6 显示了声明这些函数的方法。需要保证的一点是，函数要声明成希望创建的消息处理程序。在这个声明中有几点需要注意：首先，把函数声明成 public 方式，这样在该类的外部也可以访问这些函数；其次，把函数声明成虚函数类型，以便在必要时重载它们。

程序列表 2.6

```
// Message Handlers
public:
    // Add some movement handlers.
    virtual   void   OnForward();
    virtual   void   OnBackward();
```

这两个消息处理函数相当简单，程序列表 2.7 显示了需要添加到 CSgl_HTMLView.CPP 文件尾部的程序代码（在消息处理程序节）。这两个消息处理程序所做的事情就是直接调用 CHtmlView 类的成员函数 GoForward()和 GoBack()。

程序列表 2.7

```
////////////////////////////////////////////////////////////////////////////////////////////////
// CSgl_HTMLView message handlers
void  CSgl_HTMLView::OnForward()
{
     GoForward();
}

void  CSgl_HTMLView::OnBackward()
{
     GoBack();
}
```

如果现在编译和运行这个应用程序，就会发现 ReBar 上已经添加了两个按钮。单击 Backward 移动到最近浏览过的前一个 Web 页面，单击 Forward 按钮就会看到刚刚退出的那个页面。这两个按钮的工作方式与它们在 Web 浏览器中的工作方式相同。当然，这还仅仅是个开始，你可以在该对话框工具条上添加

任意个数的定制按钮或其它控件。

第 3 章　理解 Visual C++的资源

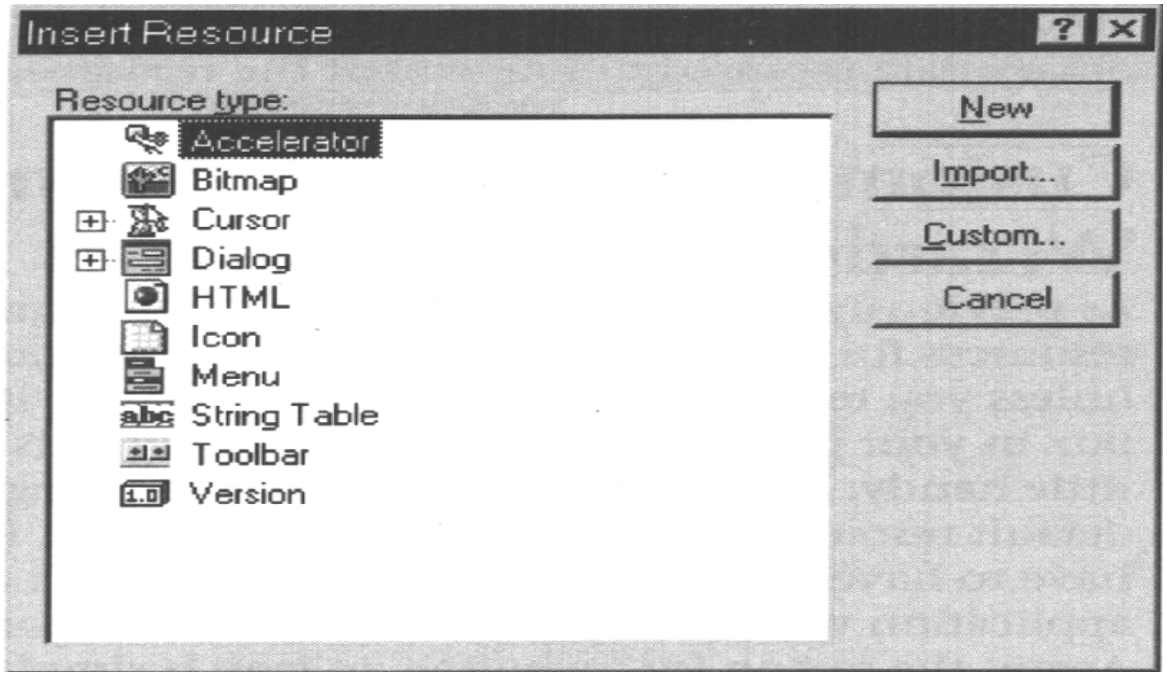
资源，是指各种各样可供利用的财富，它们可能埋藏在地表下面，也可能就在你的后院中。就物理意义而言，从森林中的树木到地层中开采出来的煤炭，这一切都可以称为资源。虽然不能从 Visual C++中获得物理意义上的贵重珍宝，但可以挖掘到建立程序时所需的资源。

和任何一种矿产资源一样，Visual C++中的资源提供了一种原材料，将其与程序元素结合起来就可以创建一个成品。在这里，资源与代码结合，创建应用程序的一部分，如菜单或工具条。事实上，在程序中看到的大多数元素也都来自某类资源。当然，Visual C++资源不仅仅是原材料，但原材料的作用是非常大的。我们在下面的章节中讨论资源的用途，牢记住这一点的话，就会得到很大的帮助。

那么，可以从 Visual C++的资源矿产中挖掘到什么呢？这些资源的外形和大小各不相同，包括加速键、位图、光标、对话框、图标、菜单、串表、工具条和版本信息。随着课程的进展，我们最后将使用十种资源中的九种（将在第 15 章讨论 HTML 资源）。你会看到，某些资源存在子类，每种资源都有特定的用途。在某些情况下，用向导设计程序时，Visual C++自动创建资源，如 About 对话框。所创建的其它资源是设计另外一些程序的一部分，如字符串。最后，

你会人工设计一些资源，如对话框。

注 Visual C++支持十种资源：加速键、位图、光标、对话框、HTML、图标、菜单、串表、工具条和版本信息。



注释 我们要在本章中使用第二章介绍过的 Sng1_Doc 示例。如果想把这两个练习分开，只需创建 Sng1_Doc 工程文件夹的一个拷贝。为了便于本章使用，请将所有的 Sng1_Doc 工程文件复制到 Resource（资源）工程文件夹。

幸运的是，你可以随时创建任何所需要的资源。只需显示 ResourceView（资源视图），右击 Resources（资源）文件夹，从上下文菜单中选择 Insert（插入）。

你会看到一个 Insert Resource（插入资源）对话框，如上图所示。请注意，它显示出我们已提到过的所有十种资源。还应该看到，有三种创建资源的方式，即使用对话框上的 New（新建），Import（导入）或 Custom（自定义）按钮。

可以在当前程序中使用来自其它工程的资源。例如，可能想在编写的每一个程序中都使用相同的公司徽标。只需右击 Resources（资源）文件夹，从上下文菜单选择 Import（导入）而不是 Insert（插入）。你会看到 Open-type（打开类型）对话框。选择包含了想要导入的资源的文件。正是由于这个原因，你会想到把你的大多数资源与程序分离。可惜的是，这种方法只对图标、光标、声波文件以及 Visual Basic 窗体文件有用。

技巧 有几种使用来自其它工程的其它类型资源的方式，如 About（关于）对话框。只需简单地为想要共享的资源创建一个分离的 RC 文件就行了。这在遇到像 About 对话框这样的资源时很有用。你只要将 RC 文件添加到工程中，剩下的由 Visual C++来处理。另一种共享资源的方法是，将其复制到剪贴板，再在创建了正确类型的空白资源后，将其粘贴到工程中。

自定义资源就是自己设计的资源。实际上，它并不适合 Visual C++所提供的预定义类型。创建一个自定义资源项目很容易：只要单击 Insert Resource（插入资源）对话框上的 Custom（自定义）按钮。你会看到包含单个空白的 New Custom Resource（新建自定义资源）对话框。简单地输入自定义资源名。这时，Visual C++会产生一个新的文件夹，它含有你提供的自定义资源名和该文件夹内的一个新资源。还要提供实现这个资源所需的二进制数据。

3.1 定制使用应用程序向导生成的应用程序所用的资源

如前所述，在设计应用程序的工作区时，Visual C++会自动创建某些类型的资源。例如，除非另外告知 MFC AppWizard，否则它会在程序中包括 About（关于）对话框。这样做的原因是简化应用程序的设计，为开发人员提供一些便利条件，还可以告诉用户是谁设计了他们正在使用的程序。一些缺省资源很有趣，而不是显得很功利。即使你确实不需要特殊的程序图标，还是没有任何办法告诉应用程序向导，你并不想要程序的自定义图标。另外，包括图标的原因也很简单——Windows 需要它在 Explorer 中显示程序。创建的每一个工程都应该有版本信息。这一次这个信息对你是有益处的，因为它帮助你掌握用户拥有哪一种版本的产品。

注 Visual C++ 至少要提供下述缺省资源：自定义应用程序图标、About（关于）对话框和版本信息。

如你所见，自动创建这些缺省资源的原因是很明显的。事实上，这三种资源代表了对每个工程你都应该考虑定制的部件。尽管你应该考虑自己的标准定制技术，但是本节仍会给出一些关于处理缺省资源的提示和技巧。

应用程序图标

用 Visual C++创建的每一个 MFC 应用程序都有一个缺省的应用程序图标。事实上，图标的名称总是相同的：IDR_MAINFRAME。你会发现，这个图标不

仅定义程序图标在 Explorer（或任何其它显示程序图标的程序）中的外观，还同样影响程序的内部表现。例如，About（关于）对话框显示这个图标作为向用户介绍应用程序信息的一部分。

你所创建的所有 MFC 应用程序开始时都有一个相同的图标，如图 3.1 所示。

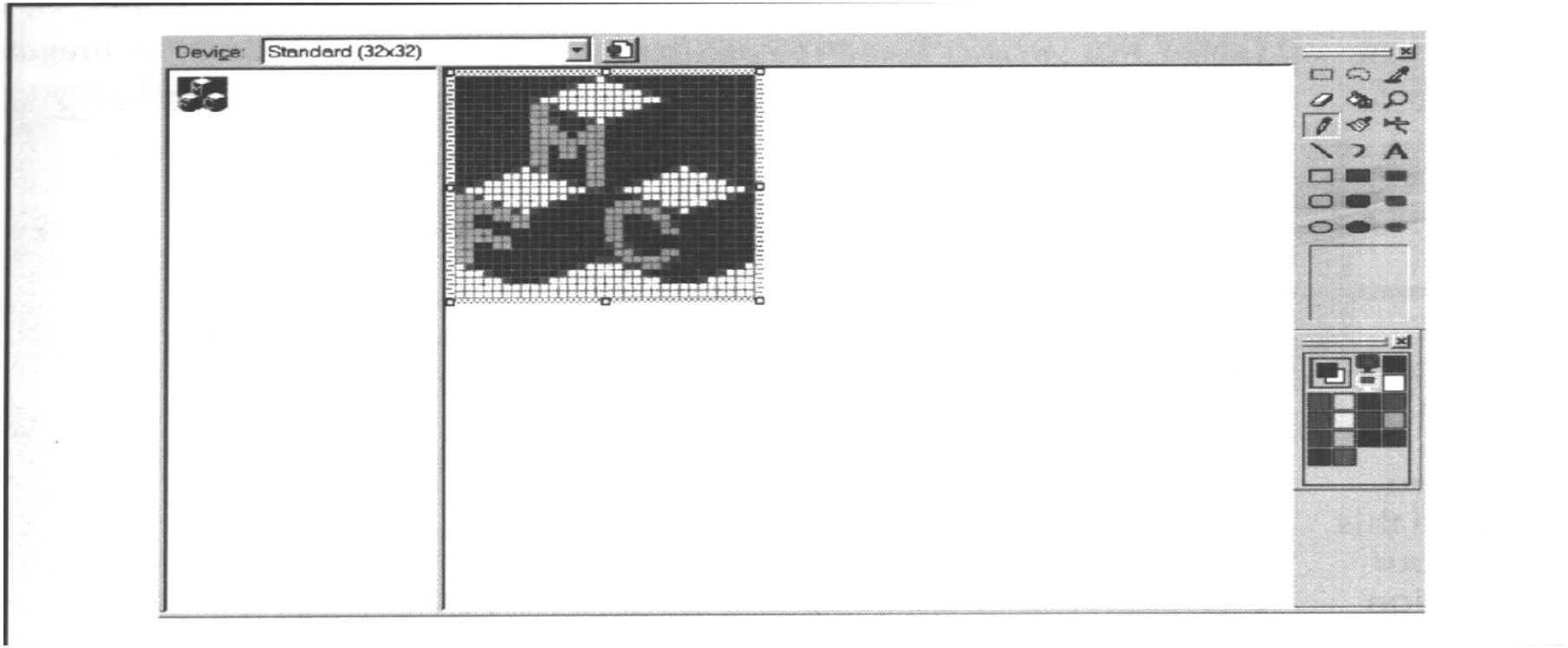


图 3.1　MFC AppWizard 提供了这个可以使用的缺省图标，但你确实应该定制一个自己的图标

注　虽然可以在应用程序中添加许多需要的图标，但缺省的自定义应用程序图标是 IDR_MAINFRAME。

如果你并不在乎人们在安装你的应用程序时看到的图标是什么，那么他们就会看到这个 MFC 徽记。就我个人来说，希望徽记看上去更有趣一些，你也应该如此。毕竟，每次程序用户在 Explorer 中寻找或从 Start（开始）菜单中选择时，都会看到这个图标。

技巧 一定要为应用程序同时修改 32×32 图标（如图 3.1 所示）和 16×16 图标。32x32 图标是在 Explorer 中看到的图标。16×16 图标显示在程序的控制菜单中和 Windows 任务条上。Device 下拉列表框（如图 3.1 所示）是选择 32×32 或 16×16 图标的位置。

决定用应用程序创建文档时，出现第二个缺省图标，就像我们在第二章的 Sng1_Doc 实例中所做的一样。在这种情况下，会看到已创建的每一个文档类型的 IDR。在我们的实例中，它是 IDR_SNGL_DTYPE。它们会有相同的起始图标；图 3.2 显示了 MFC AppWizard 提供的缺省图标。与应用程序图标不同，如果应用程序支持多种文档类型的话，它几乎总是要强制你定制文档的图标。

除了绘图功能外，如果想创建有效的图标，还要了解可供随意使用的工具。所需要的全部工具都显示在窗口的右侧。它们包括一套标准的绘图工具（Graphics 工具条）和一个颜色图表(Colors 工具条），这些工具条可以象我们讨论过的任何其它工具条那样隐藏或显示它们。

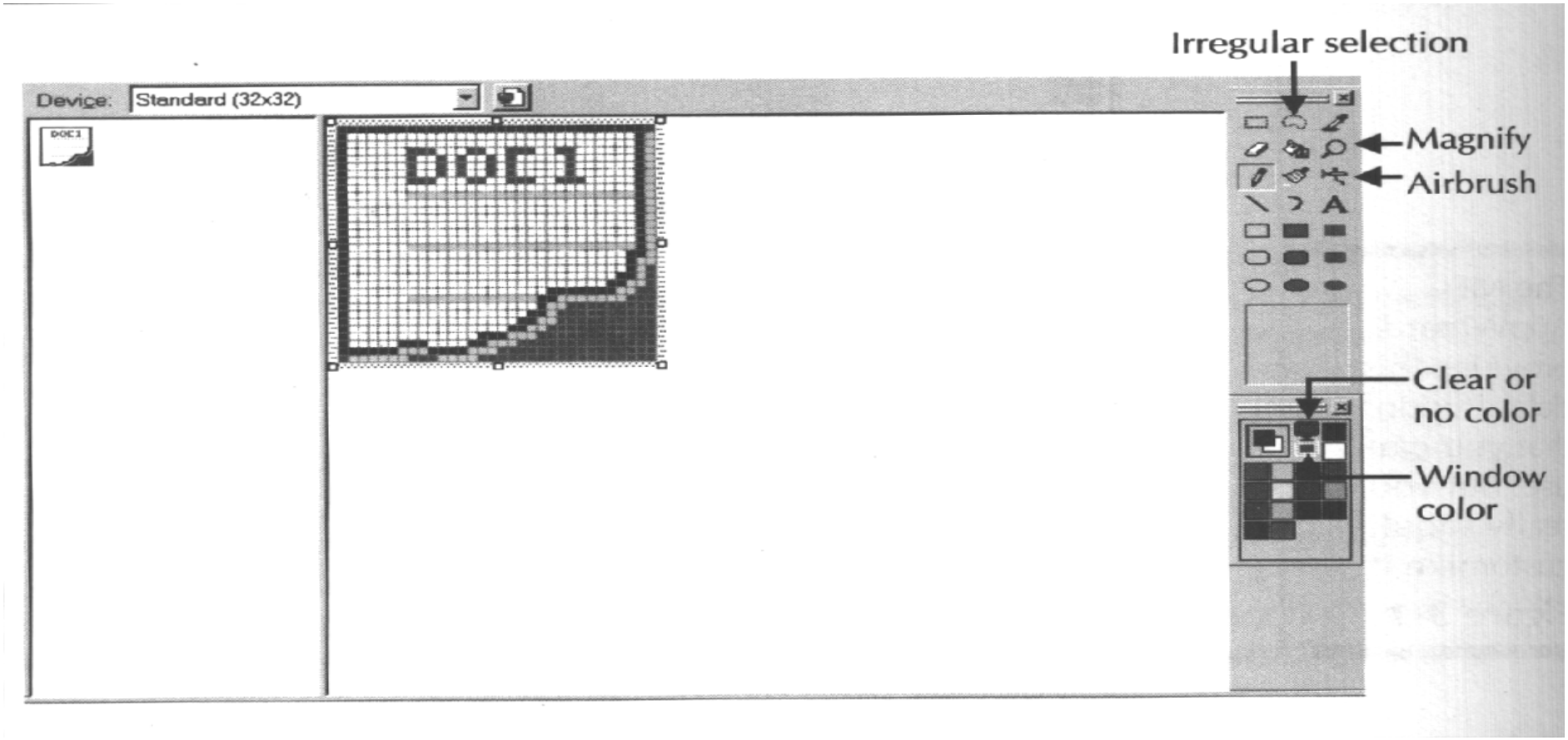


图 3.2　如果应用程序需要，MFC AppWizard 还会提供一个像这样的文档图标

如图 3.2 所示的 Graphics 工具条上的工具你不应该都不熟悉。你已经熟悉了标准的选择工具、画线和各种椭圆及矩形的工具。但是，Visual C++还包括喷雾器工具和不规则选择工具，它们使绘画变得更加容易。你还会发现，Magnify（放大）工具比一些绘图程序中的相同工具更为出众——它可以将图画放大至

正常大小的八倍。

你还需要了解 Colors 工具条上的两种特殊颜色，由于它们使用监视器符号代替了彩色方框，所以可以很容易地找到它们。

注 Colors 工具条中的两个监视器包含了当前窗口颜色和透明颜色，后者可以把图标下面的东西显示出来。

上面的监视器可以创建一个透明区。换言之，你会看到桌面上或放置程序图标的其它地方的该区下面所出现的东西。下面的显示器允许你创建一个区，它使用和用户窗口前景颜色相同的颜色。换言之，改变窗口颜色时，图标中该区的颜色也随之改变。你会看到当前的前景和背景颜色出现在两个监视器的左侧。前景颜色出现在上面的矩形中，背景颜色出现在下面的矩形中。

让我们来看一个示例，看看可以用这两个图标做些什么。图 3.3 显示了为我的程序版本而绘制的示例图标。和你看到的其它图标相比，也许它们的艺术性不是很高——你们当中一定有艺术家——但比你从 Visual C++得到的缺省图标要好些。很明显，可以用任何方式定制图标。请试着使用各种颜色。一定要试验所提到的两种自定义颜色，因为在创建图标时它们特别重要（很多程序员因为不知道如何有效使用这两种特殊的颜色，所以创建的图标看上去很奇怪，它们确实不能与桌面上其它图标协调一致）。

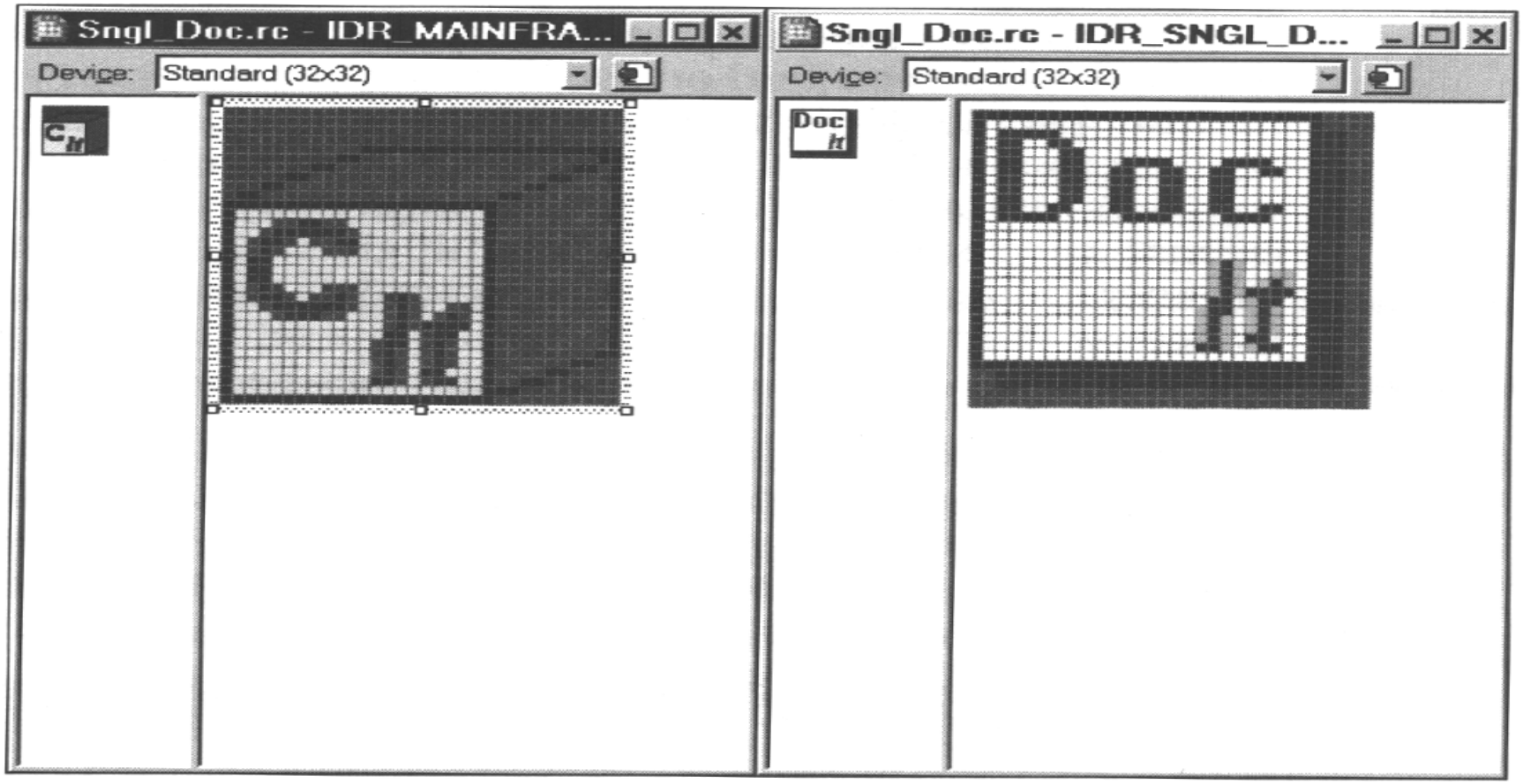


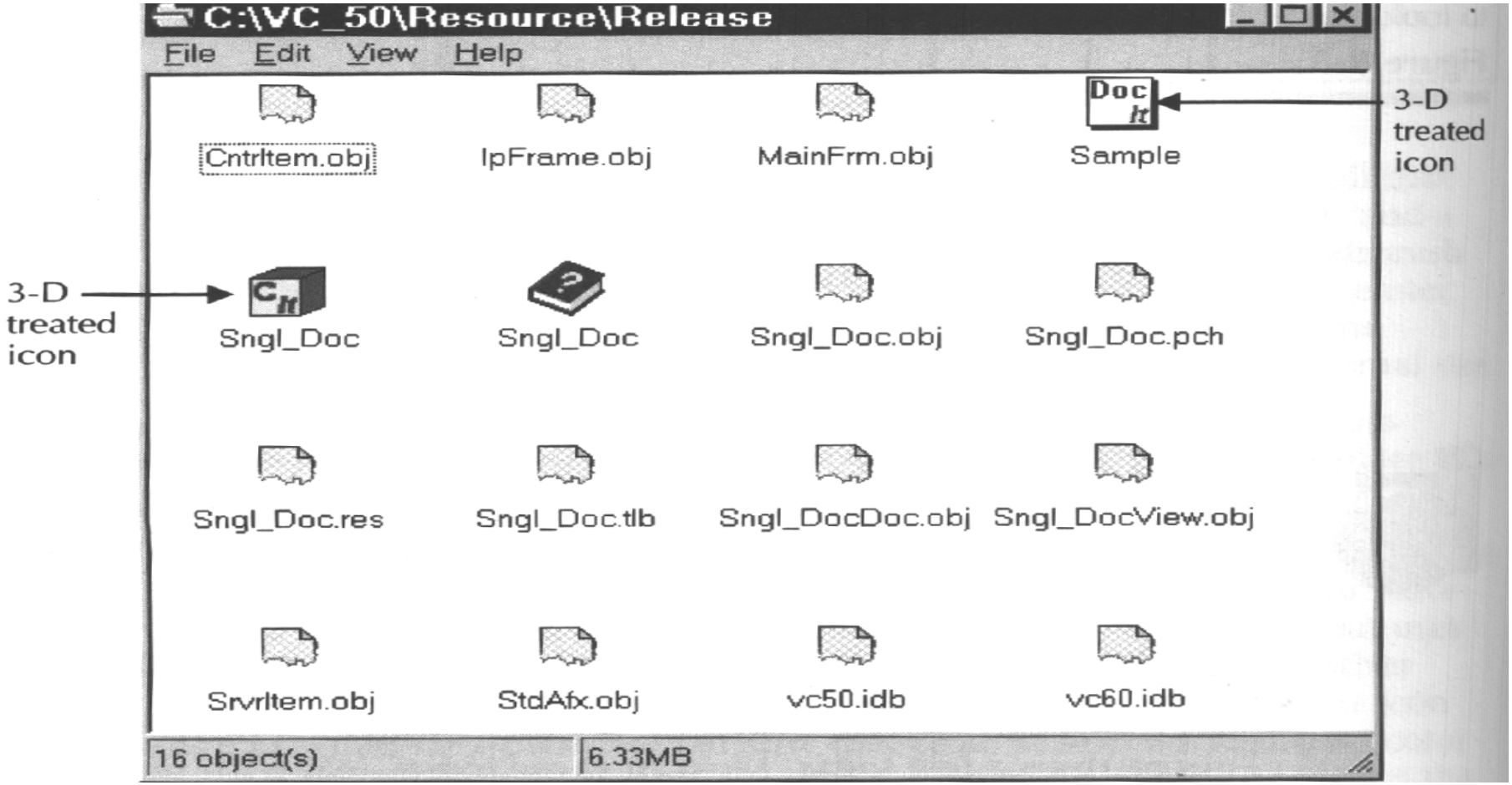
图 3.3　向程序中添加特殊的图标一点也不困难；怎么想的就怎么画

Web 链接　如果创建图标时需要一些美妙的灵感，Internet 上众多的站点正合乎你的需求。一个较好的站点是 http://personal.solutions.net/hillel/ico.htm。它包含了一些很好的光标、图标示例以及创建它们的实用程序。如果这个站点还不能为你提供足够的灵感，可以看一看 http://crab.

看 http://crab. rutgers.edu/icons_new/icons.html。虽然这个站点主要提供 GIF 图标，但其 3,000 个示例应该可以带给你一些灵感。这些图标中的绝大多数都涉及流行的主题，像 The Simpsons。幸运的是，图标做了索引，所以你不必一次把全部 3,000 个示例都试一遍。你还可以找到数量可观的特定主题站点。例如，http://www.geocities.com/Area51/8604/xfiles.htm 包含一批相当完整的 X 文件专用图标（ICO 格式）。

警告 决不要以为你可以在计划出售或转赠他人的程序中使用 Internet 上找到的任何图标（或与此相似的其它图形）。而要假定这些资源有利于引发灵感，除非你得到了原始编写者的书面许可，不可把这些资源用于其它用途。在 Internet 提供的开放环境中，侵权现象是严重的，也比较容易发生。从经验来看，最好的办法是创建你自己的图标，或者从可靠的来源那里得到准许用于商业目的的许可。

请注意，我在两个图标上都使用了透明颜色，并赋予它们三维的外观。再说一句，虽然它们并非那么富于美观，但确实给了用户一种特别的感觉。很明显，编辑器内的透明着色所表现出来的和用户看到（实际上或者看不到）的不同。下面是在 Explorer 内看到的两个图标的外观。



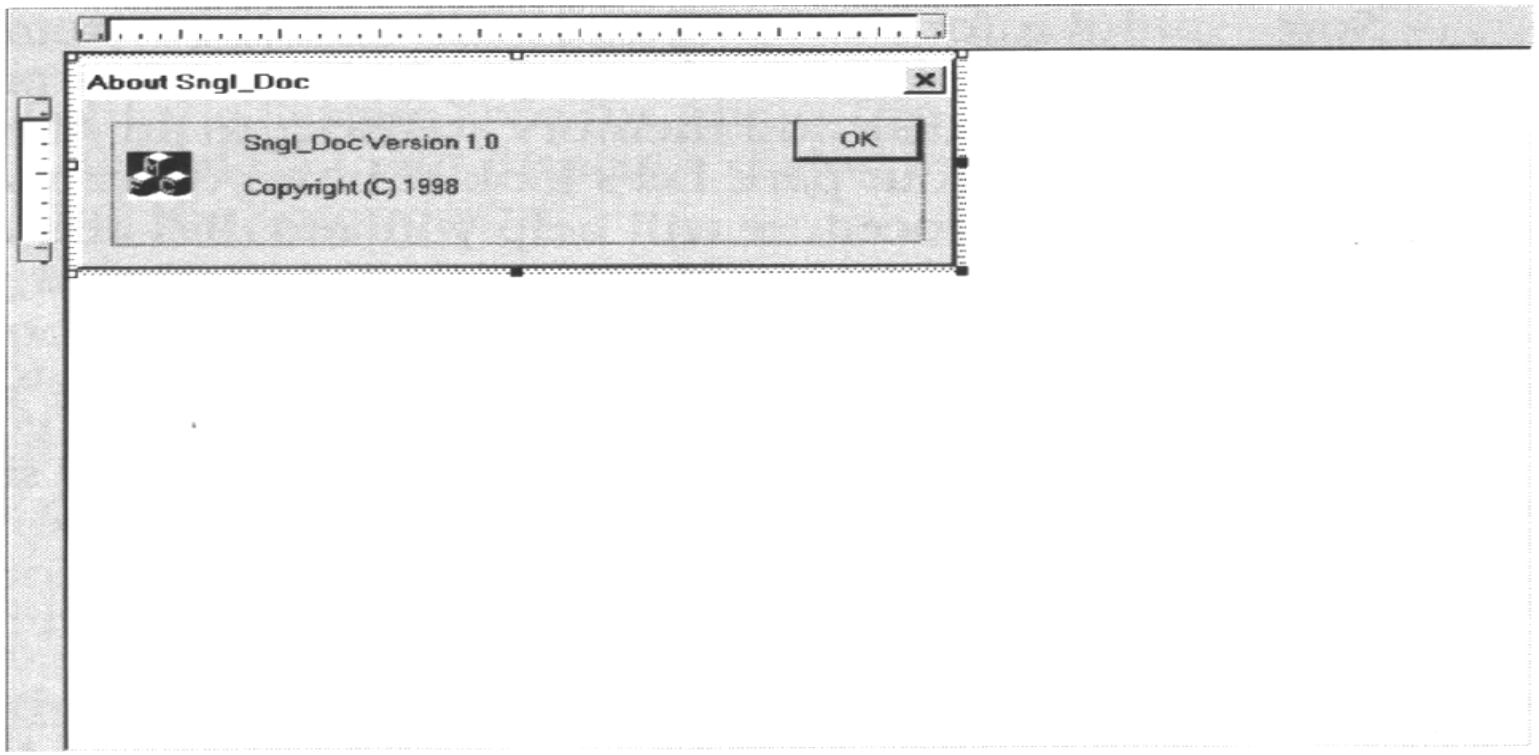
技巧 在修改图标后，一定要再次编译程序，否则它不能在 Explorer 中显示。运行程序也是一个好主意，因为在第一次运行程序时，Visual C++会制作注册表项目。最后，一定要在 Explorer 内使用 View | Refresh 命令。否则，你看到的依然是 Explorer 先前存储的老图标。

About 对话框

为应用程序定制 About 对话框并不麻烦。事实上，使用前一节中介绍的信息修改应用程序图标时，就已经开始定制该对话框了。Visual C++自动将你为应用程序创建的图标添加到缺省 About 对话框。下图是我们的示例应用程序使用的缺省 About对话框。

技巧 因为显示在缺省 About 对话框内的信息总是基于你提供给 MFC App-Wizard 的输入，所以要给出尽可能清楚的信息。这样做将减小后续的工作量。这些设置还会影响我们将在本章后面看到的版本信息。

那么，需要在标准的 About 对话框中提供些什么呢？缺省 About 对话框为你提供了一些灵感。基本上，一个 About对话框应该提供四条信息：公司名称，版权信息，产品名称和版本号。如果提供了这样四个必要的项目，应用程序的 About 对话框就有了它绝对应该拥有的一切。请注意，缺省 About 对话框提供了这四条必要信息中的三条——如果你不想做太多工作的话，也只需要添加上公司名称即可。



当然，你可能想将其它一些必要的信息提供给应用程序中的 About 对话框。例如，让用户知道该去哪里获得技术支持是很重要的。添加电话号码和几个访问指令是个好主意，这也无须消耗太多的空间。一些公司还添加了产品注册序号，作为技术支持信息的一部分，但提供这类信息要从注册表或 INI 文件开始。

某些有用的信息包括了可用于支持应用程序的当前系统资源。例如，Visual C++提供了一种将当前磁盘和内存统计值添加到 About 对话框的方式，而且并不需要你自己多做多少工作。让我们看看如何才能在对话框中添加这种能力。下面的过程帮助你将磁盘和内存统计值添加到所创建的任一 About 对话框（或

者与此相似的任一其它对话框）中。

1. 调整 About 对话框的大小。现在的对话框太小，放不下额外的信息。很明显，你也不想把它做得太大，因为 About 对话框应该足够小，以适应可能存在的最小显示区。在本示例中，将 About 对话框的大小调整为 250*150，这应该足以显示磁盘和内存统计值了，也为版权和其它辅助信息留出了空间。

2. 将两个静态文本框控件添加到 About 对话框中。前一个用于显示内存统计值，后一个用于显示磁盘统计值。

3. 使这两个控件的大小为 150*18。将内存统计值静态文本框控件定位在 40,40。将磁盘统计值静态文本控件定位在 40,65（这些定位数字假定，你没有为了适应更多的公司名称或产品信息而改变缺省静态文本框控件的大小）。

4. 右击内存统计值静态文本框控件，然后从上下文菜单中选择 Properties（属性）。将该控件的 ID 字段改成 IDC_MEMORY。对磁盘统计值静态文本框控件做相同的处理。但是，在这个事件中将 ID 字段改变成 IDC_DISK。不要选中两个控件的 Group（组）框——我们想把它们作为独立的静态文本框控件使用。现在，我们有地方放置信息了，可以添加收集信息的代码了。

5. 使用 Project | Add to Project | Components and Controls 显示 Components and Controls Gallery 对话框。在 Look In 字段中双击 Visual C++ Components，你会看到如下图所示的一系列部件。

6. 加亮 System Info for About Dlg 部件，然后单击 Insert（插入）。Visual C++会显示一个对话框，询问是否确实要安装该部件。

7. 单击 OK。Visual C++会显示另一个对话框，告诉你正在安装的部件会添加信息到 About 对话框中。

8. 单击 Yes 继续安装。Visual C++完成 System Info for Abourt Dlg.部件的安装。

9. 单击 Close 关闭 Components and Controls Gallery 对话框。

此时，好像什么事也没有发生。Visual C++只是停在那儿，什么也不做。除非你知道到哪里去看，否则永远也找不到 System Info for About Dlg.部件所添加的代码。打开 ClassView，然后双击 CAboutDlg::OnInitDialog()函数。你会在初始化代码中看到一些新的条目，按程序列表 3.1 所示进行修改。

程序列表 3.1

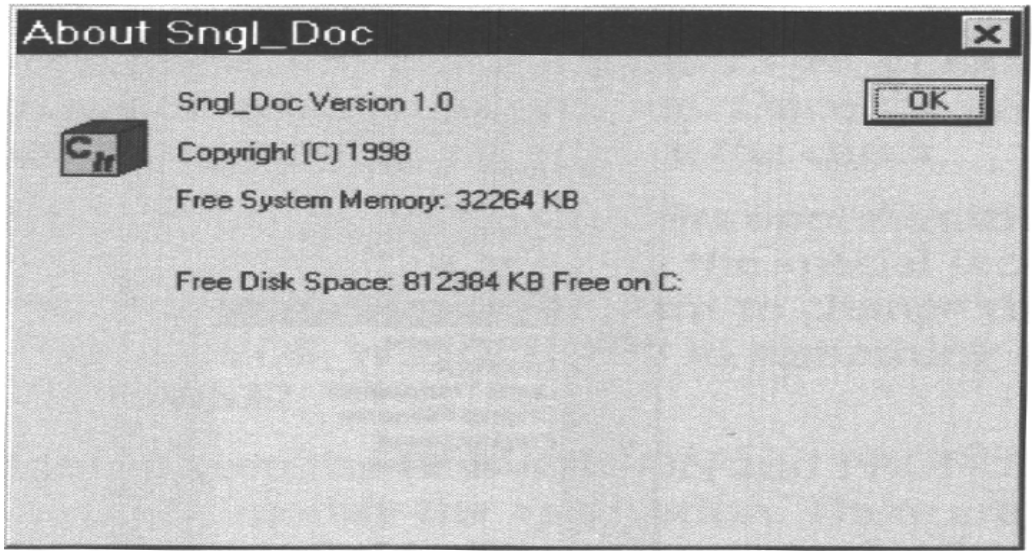
```
BOOL CAboutDlg::OnInitDialog( )
```

```
{
        CDialog::OnInitDialog();       // CG: This was added by System Info Compoment.
// CG: Following block was added by System Info Component.
{
      CString strFreeDiskSpace;
      CString strFreeMemory;
      CString strFmt;

      // Fill available memory
      MEMORYSTATUS    MemStat;
      MemStat.dwLength = sizeof(MEMORYSTATUS);
      GlobalMemoryStatus (&MemStat);
      strFmt.LoadString (CG_IDS_PHYSICAL_MEM);
      strFreeMemory.Format(strFmt, MemStat.dwTotalPhys / 1024L);

      // Display the amount of free memory.
      strFreeMemory = "Free System Memory:" + strFreeMemory;
      SetDlgItemText (IDC_MEMORY,strFreeMemory);

      // Fill disk free information
      struct_diskfree_t diskfree;
```

```
    int nDrive = _getdrive(); // use current default drive
    if (_getdiskfree(nDrive, &diskfree) == 0)
    {
        strFmt.LoadString(CG_IDS_DISK_SPACE);
        strFreeDiskSpace.Format(strFmt,
            (DWORD)diskfree.avail_clusters *
            (DWORD)diskfree.sectors_per_cluster *
            (DWORD)diskfree.bytes_per_sector /  (DWORD)1024L,
            nDrive-1+_T('A'));
    }
    else
        strFreeDiskSpace.LoadString(CG_IDS_DISK_SPACE_UNAVAIL);
        // Display the amount of free disk space.
        strFreeDiskSpace = " Free Disk Space:" + strFreeDiskSpace;
        SetDlgItemText(IDC_DISK, strFreeDiskSpace);
    }
    return TRUE;      // CG: This was added by System Info Component.
}
```

请注意，System Info for About Dlg.部件提供的缺省代码只从缺省驱动器中检索数据。可以很容易地修改这个代码，以检索用户机器上所有驱动器的驱动器空间信息。你得到的内存及磁盘空间字符串缺少像“Free Disk Space”这样的

提示，这一提示说明了 About 对话框提供的是什么信息。可惜的是，没有提示信息将给用户带来更多的疑惑，而不是帮助。一定要添加某种提示信息（像程序列表 3.1 中显示的那样）来告诉用户数据的意义。最后请注意，缺省字符串实际上由更为复杂的统计值组成。可以很容易地将这个额外的信息添加到你的 About 对话框。下面是这个代码产生的最终结果。



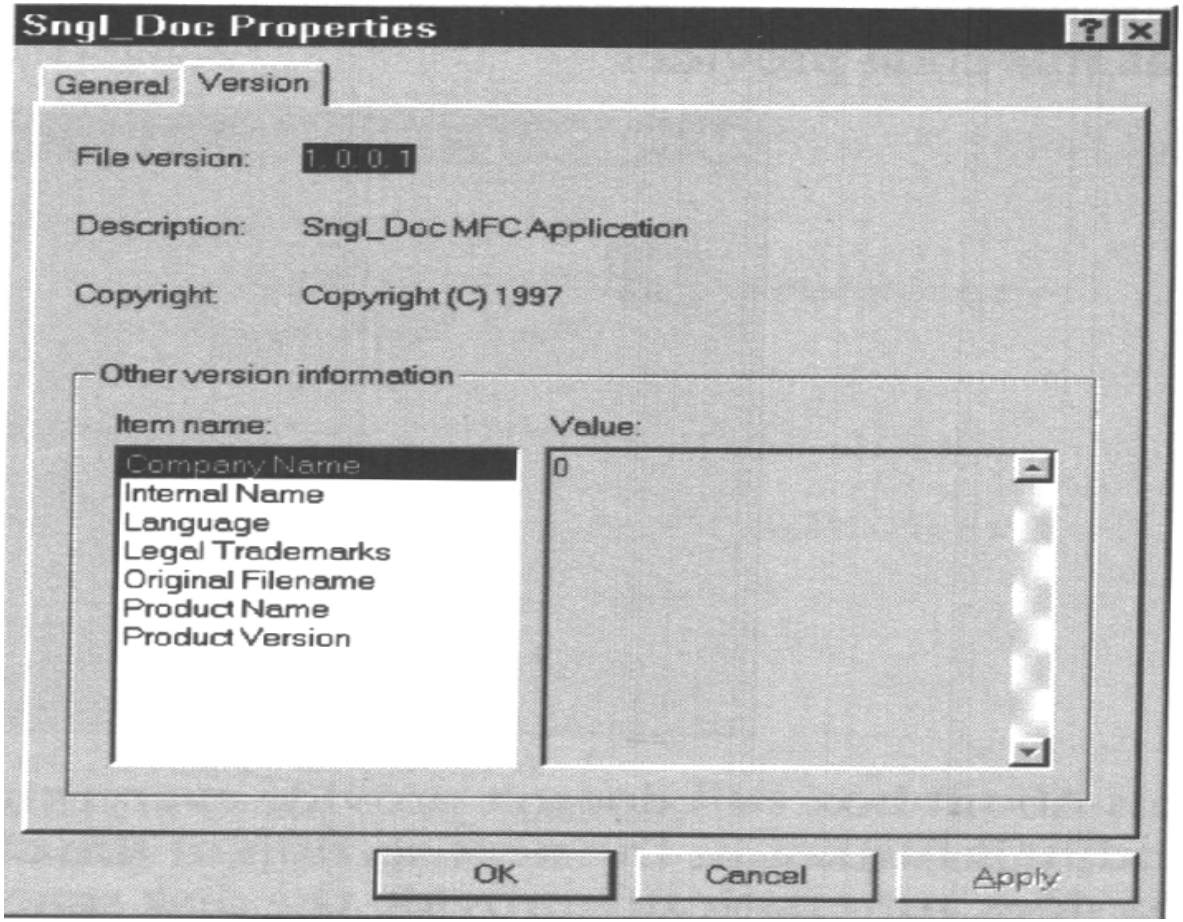
这个 About 对话框并不提供用户所需要的一切，但我们将在本章的下一节中进一步修改它。现在，About 对话框自动提供了磁盘空间和内存的统计值。它还提供了程序名称、版本和版权信息。我们将在下一节看到，你同样可以自动生成这三条信息。

现在再看一看定制问题。谈到定制这个话题，你可以做许许多多的工作。目前，应用程序中的某些 About 对话框本身就是个袖珍型的应用程序，它们或许承担的任务太多了。如果发现有必要向 About 对话框中添加按钮和另外一些

对话框时，就应该考虑将此信息放置到别的地方，比如某个帮助文件中。

版本信息

你或许未认真思索就跳过了 Visual C++自动提供的版本信息资源。以前，除了知道如何找到版本信息的程序员外，该信息对每个人来说都深深地隐藏了起来。问题在于，使用 Windows 95 和 Windows NT 4 提供的 Explorer 界面，你不能再回避版本信息了。现在要显示应用程序提供的版本信息，只需右击 Explorer 中的程序图标，然后从上下文菜单选择 Properties（属性）。选择 Properties（属性）对话框的 Version（版本）选项卡，会看到如下图所示的画面。



如你所见，缺省版本的信息量很少。如果是这样的话，用户甚至会不知道他们在和哪一家公司打交道。因为越来越多的用户开始关注 Properties（属性）对话框必须提供些什么，所以对程序员来说，在这里放入适当类型的信息就变得越来越重要了。

技巧 填写应用程序的版本信息并不是单向的。也可以用这个信息填写应用程序的其它区域，这意味着你必须只改变一个地方的信息，使其处于当前状态。在本节中我们看看如何为 About 对话框使用这种技术，当然你也可以在其它地方使用它。

高级技巧

在某些情况下，你或许想修改版本信息资源中的文件专用信息。例如，如果双击 FILEFLAGS(文件标记)条目，会看到 FILEFLAGS(文件标记)对话框包含两个复选框。第一个复选框说明版本信息是调试版的版本信息还是发行版的版本信息。第二个复选框说明该版本信息是否是程序的预发行版。对你的β版程序来说，可以选中 VS_FF_PRERELEASE 复选框。一旦该程序能够正常发行，那么你应该不选中这个复选框。

FILEOS 条目是另一个提供定制机会的地方。假定程序依赖 Windows NT 提供的安全特征，你可能想将 FILEOS 条目从缺省的 VOS_WINDOWS32 改成 VOS_NT 或 VOS_NT_WINDOWS32。

修改应用程序时保持 FILEVERSION(文件版本)和 PRODUCTVERSION(产品版本)域为最新也很重要。使用这些区域有几个不同的方式。最好的方式是，简单地忽略 Visual C++建议的方法，如果这是程序的第一个版本，输入像 1.0 这样的简单数字。

让我们看一看示例应用程序的版本信息。可以在 ResourceView（资源视图）中的 Version（版本）文件夹下找到缺省信息。缺省资源名是 VC_VERSION_INFO。图 3.4 显示了缺省的版本信息。粗实线上的条目通常反映应用程序的设置，并不需要经常修改它们。以 Block Header 开始的粗实线下

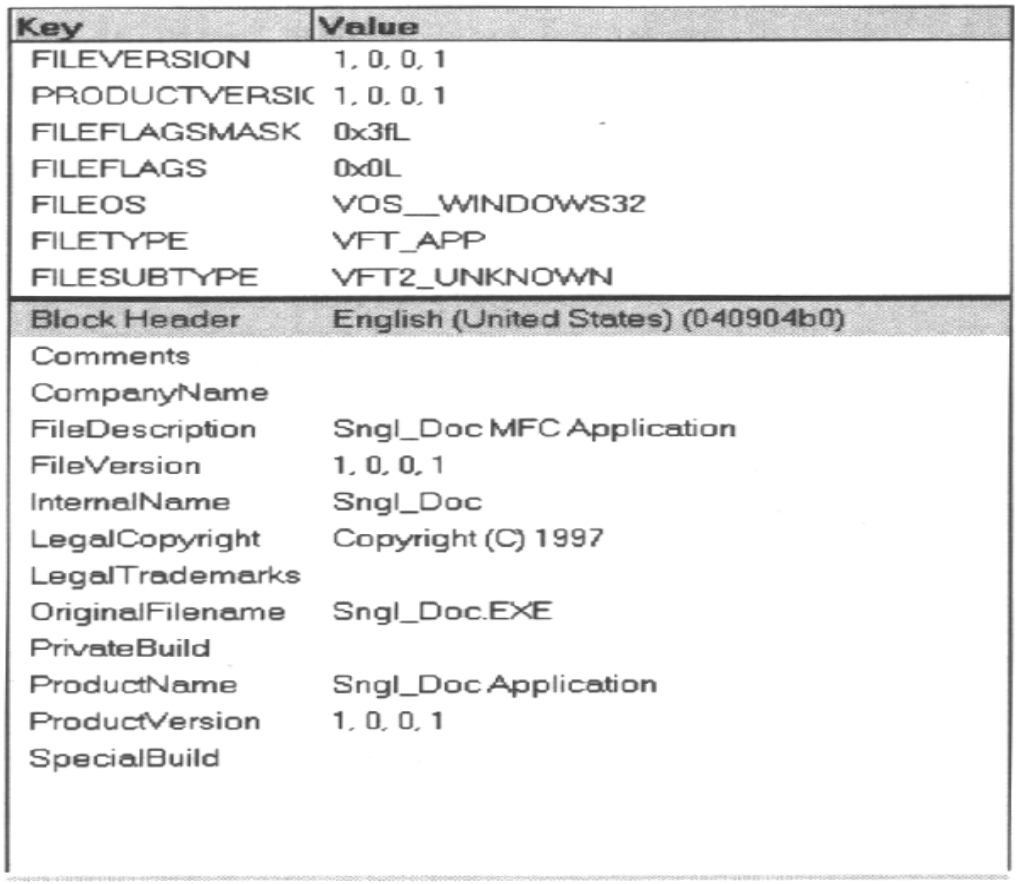
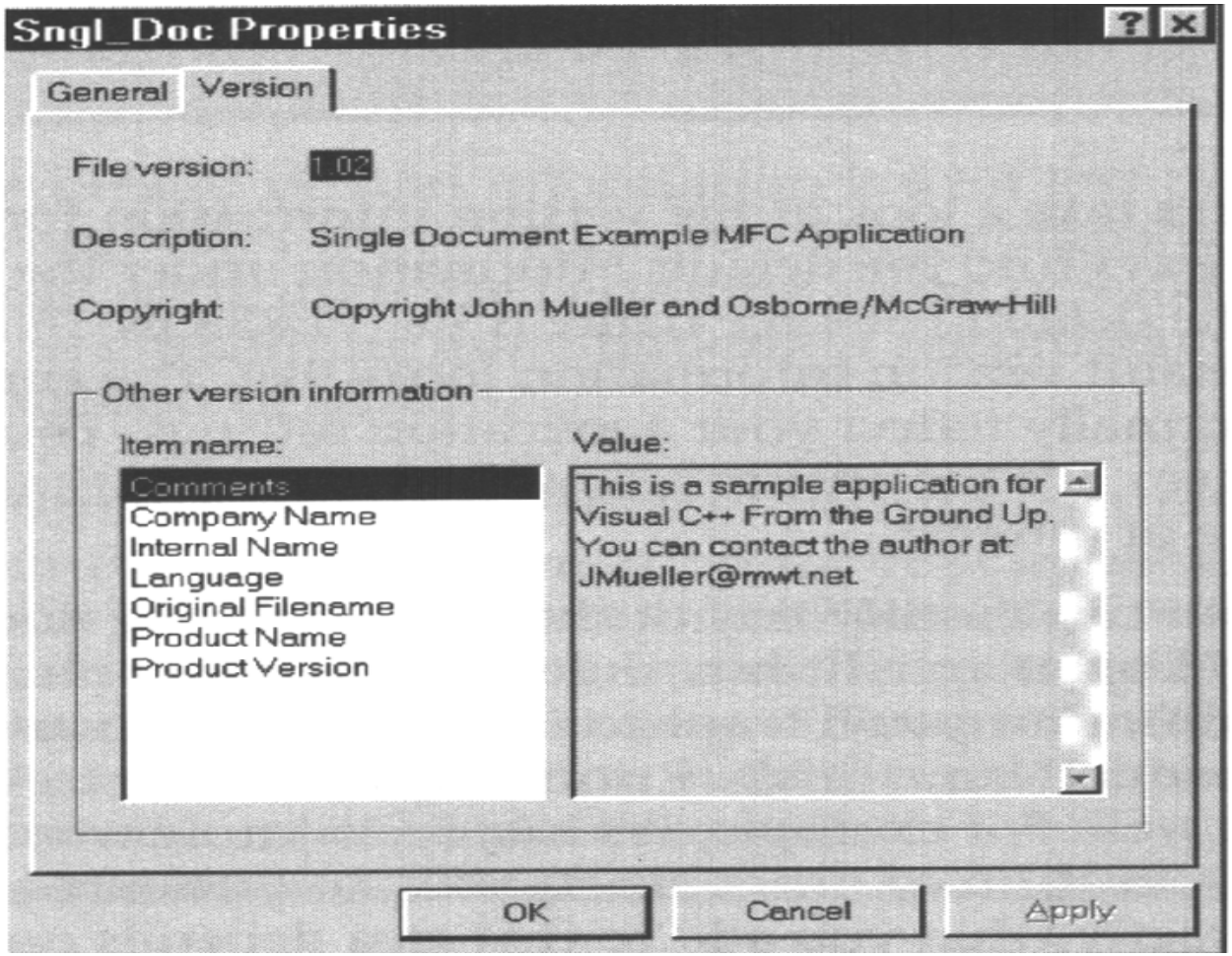


图 3.4　要修改缺省版本信息，使其和公司及产品的实际信息一致

可以双击任何一个文本条目来修改它们。Visual C++会打开一个用来更改信息的编辑框。至少，你想更新CompanyName（公司名称），LegalCopyright（合法版权）和ProductName（产品名称）字段。我通常还添加一些信息到Comments（注释）字段。例如，关于某个应用程序或其它可执行程序，知道该与谁联系是很方便的，所以我经常添加我的名字和电子邮件地址。到底向这个区域添加些什么，要视公司政策、合法需要和个人爱好而定。下面是在Properties（属性）对话框中查看时，我的已修改版本的信息。



既然已经花时间向应用程序中添加了版本信息，那么充分发挥这些信息的作用就显得尤其重要了。About 对话框中相同类型的信息通常还出现在其它需要版本信息的某些地方，所以将两者结合起来是一个好主意。这样就不会出现 About 对话框和版本信息不合拍的情形。程序列表 3.2 显示了这样的代码，你需要将其添加到 CAboutDlg::OnInitDialog()函数中，以将版本信息添加到 About 对

话框（我们在本章的 About 对话框一节中看到过这个函数，所以这里只说明新的代码——将此代码放在函数的开始位置）。

程序列表 3.2

```
LPTSTR      lpstrFileName;      // Name of our program.
DWORD       dwHandle;           // A placeholder handle.
DWORD       dwSize;             // Size of the version information block.
LPVOID      lpBuffer;           // Buffer to store version information block.
LPVOID      lpData;             // One version information value.
UINT        uiDataSize;         // Size of the version information value.
Cstring     strProduct;         // Product name and version.
Cstring     strCompany;         // Company name and copyright information.

// Initialize our variables and allocate memory.
lpstrFileName = "Sngl_Doc.EXE";
dwHandle = 0;
uiDataSize = 80;
lpData = malloc(uiDataSize);

// Get the version information block size,
// then use it to allocate a storage buffer.
dwSize = GetFileVersionInfoSize(lpstrFileName, &dwHandle);
```

```
lpBuffer = malloc(dwSize);

// Get the version information block.
GetFileVersionInfo(lpstrFileName, 0, dwSize, lpBuffer);

// Use the version information block to obtain the product name.
VerQueryValue(lpBuffer,
      TEXT("\\StringFileInfo\\040904b0\\ProductName"),
      &lpData,
      &uiDataSize);
strProduct = LPTSTR(lpData);
strProduct = strProduct + "\n";

// Use the version information block to obtain the product version.
VerQueryValue(lpBuffer,
      TEXT("\\StringFileInfo\\040904B0\\ProductVersion"),
      &lpData,
      &uiDataSize);
strProduct = strProduct + LPTSTR(lpData);

// Display the product name and version.
SetDlgItemText(IDC_PRODUCT,strProduct);
```

```
// Use the version information block to obtain the company name.
VerQueryValue(lpBuffer,
      TEXT("\\StringFileInfo\\040904B0\\CompanyName"),
      &lpData,
      &uiDataSize);
strCompany = LPTSTR(lpData);
strCompany = strCompany + "\n";

// Use the version information block to obtain the copyright information.
VerQueryValue(lpBuffer,
TEXT("\\StringFileInfo\\040904B0\\LegalCopyright"),
       &lpData,
       &uiDataSize);
strCompany = strCompany + LPTSTR(lpData);

// Display the company name and copyright information.
SetDlgItemText(IDC_COMPANY, strCompany);

// Free the memory we allocated.
free(lpBuffer);
free(lpData);
```

这段代码比你一开始想像的要更容易理解一些。实际上，可以将整个过程

分解成如下四个简单的步骤：

1. 用 GetFileVersionInfoSize()函数获得 VS_VERSION_INFO 结构的大小。

2. 用 VS_VERSION_INFO 结构大小创建一个足以保持该结构的缓冲区，然后用 GetFileVersionInfo()函数获得它。

3. 一旦有了 VS_VERSION_INFO 结构的本地副本，就用 VerQueryValue()函数获得各个字符串。

4. 在 About 对话框中显示字符串。

现在，你已经很好地理解了程序的总流程，让我们看看一些细节。你想知道的第一件事，可能是 VerQueryValue()函数调用中的下面一行字符串：

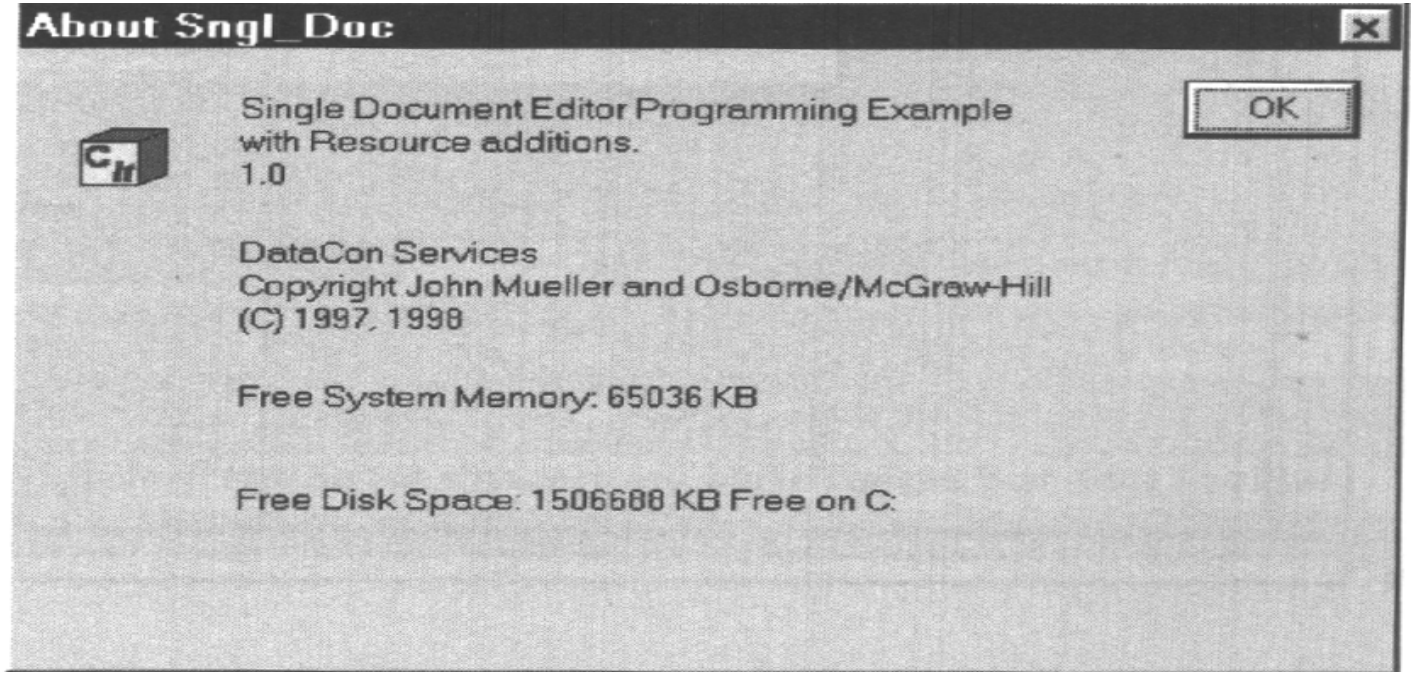
```
TEXT("\\StringFileInfo\\040904b0\\ProductVersion")
```

该字符串的第一部分 StringFileInfo 说明我们要在 VS_VERSION_INFO 结构中引用什么样的结构。在处理图 3.4 中粗实线下的信息时，StringFileInfo 是一个常量。第二参数说明想使用信息的语言版本。请注意，图 3.4 中的 Block Header（块标题）条目说明了我们正使用英语（美国）。在该条目的右侧，会看到一串 8 个数字，刚好与我们字符串的数字相匹配。从这里获得 VerQueryValue()调用的数字。最后，该字符串的最后一部分是版本信息块中字符串的一部分。在样本字符串的事件中，我们正在寻找 ProductVersion 字符串。如果仔细浏览程序列表 3.2 中的示例代码你会注意到，从一个 VerQueryValue()调用到下一个调用，唯一发生改变的，是我们要查找的字符串。

在编译程序之前，还要注意最后一件事。用 Project | Settings（工程|设置）命令显示 Project Settings（工程设置）对话框。选择对话框的 Link（链接）选

项卡。我们要添加一个特殊的库到程序中，这样才能访问版本信息。在Object/Library Modules（对象/库模块）字段中添加一个名称为VERSION.LIB的条目，然后单击OK。

如你所见，作为一段可以反复使用的代码（编写新程序时，即使需要修改的话，也不会改得太多），你得到了一个能自动更新自身的About对话框。下面是About对话框的外观。请注意，我重新安排了静态文本框控件的位置，还分别赋予了两个缺省控件新的标识IDC_PRODUCT和IDC_COMPANY。



3.2 使用加速键和菜单

菜单和加速键密切地联系在一起。这两种资源类型联手协作，让用户能更容易地完成任务。大家都知道菜单是什么——它是分层命令结构的物理表示。加速键提供该结构的快捷方式，以提高用户操作的速度。例如，要创建一个新文件，通常用 File | New（文件|新建）命令或 CTRL-N 加速键，两种方法可以产生相同的结果。

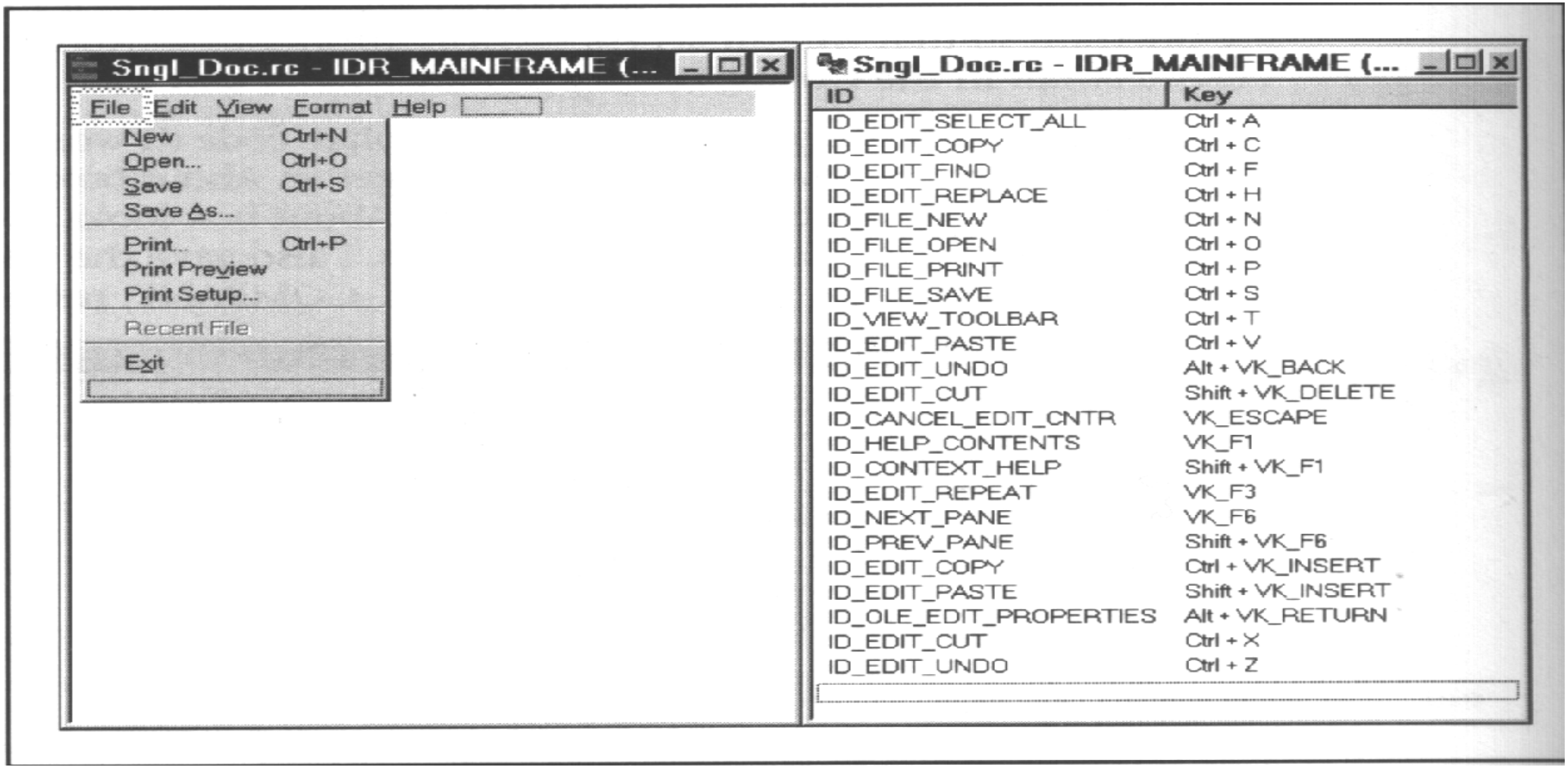
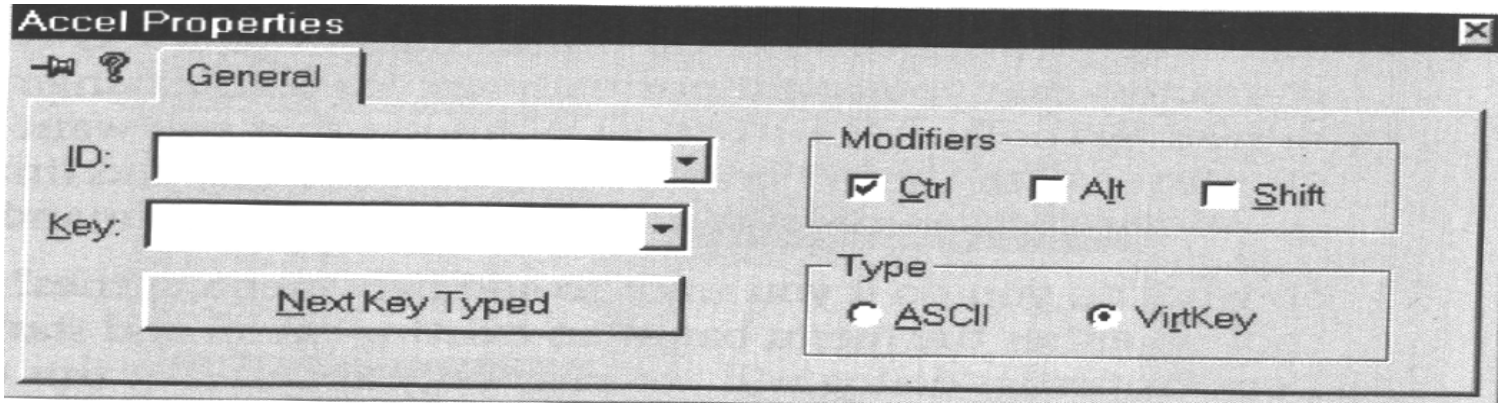


图 3.5　缺省菜单提供了你所期待的标准功能。加速键资源通过你给它赋于的名称与相应的菜单联系在一起

Visual C++将菜单和加速键作为两种不同的资源存储。图 3.5 显示了示例应用程序的主菜单和相应的加速键。注意有一点很有意思，两种资源使用相同的名称 IDR_MAINFRAME。应该记住这一点，因为该名称是将两种资源（菜单和

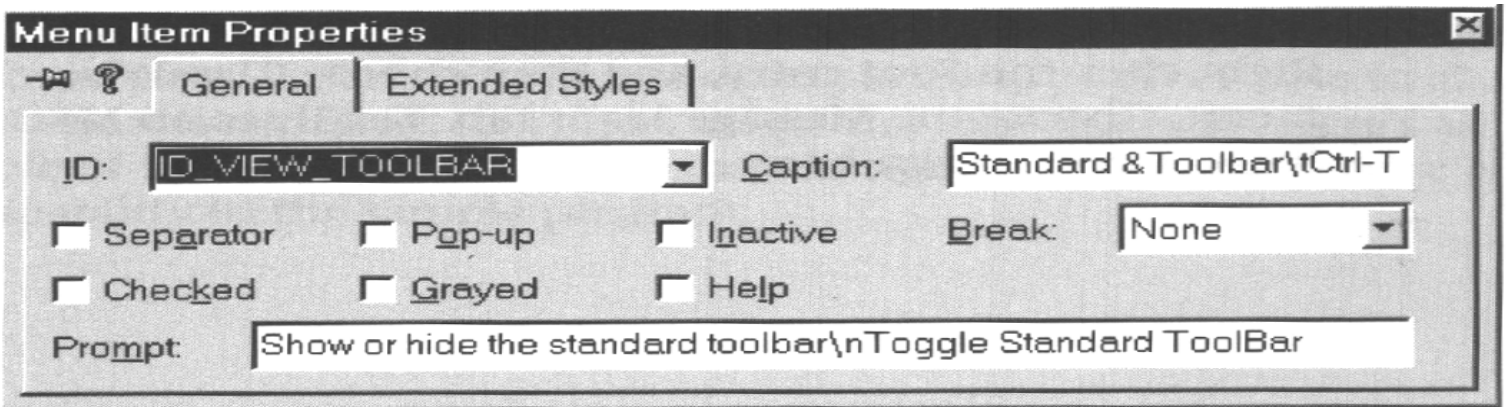
加速键）链接在一起的资源名。



让我们看看菜单和加速键是怎样协作的。将新的条目添加到现有的加速键资源中，这一点很容易。在 Accelerator（加速键）窗口中右击，然后从上下文菜单中选择 New Accelerator（新加速键）。就会看到 Accel Properties（加速键属性）对话框，如下图所示。

此时，需要从下拉列表框中选择一个 ID。Menu ID 通常使用所需菜单层次的组合来得到名称，然后在名称前加上字符 ID。例如，如果想为 View | Toolbar（视图|工具条）命令创建一个加速键（我在实例中就是这样做的），就应在 ID 字段中选择 ID_VIEW_TOOLBAR。也可以在 ASCII 码和虚拟键（VirtKey）之间进行选择。当你准备添加与某个菜单命令相关的按键时，只需单击 Next Key Typed 按钮，你会看到一个比较小的对话框，告诉你按下想要当作加速键使用的组合键。在这个示例程序中，我按下 CTRL-T。你会看到，CTRL 复选框中有一个复选标记，Key 域中有一个"T"，单击 Close（关闭）方框结束添加加速键操作。

如果现在就编译并运行程序，刚刚添加的加速键即能正常工作。事实上，你可能正想这样。但是，用户可能想不到，快速执行一个菜单命令可以用加速键。要向菜单中添加加速键，需要修改当前的菜单。

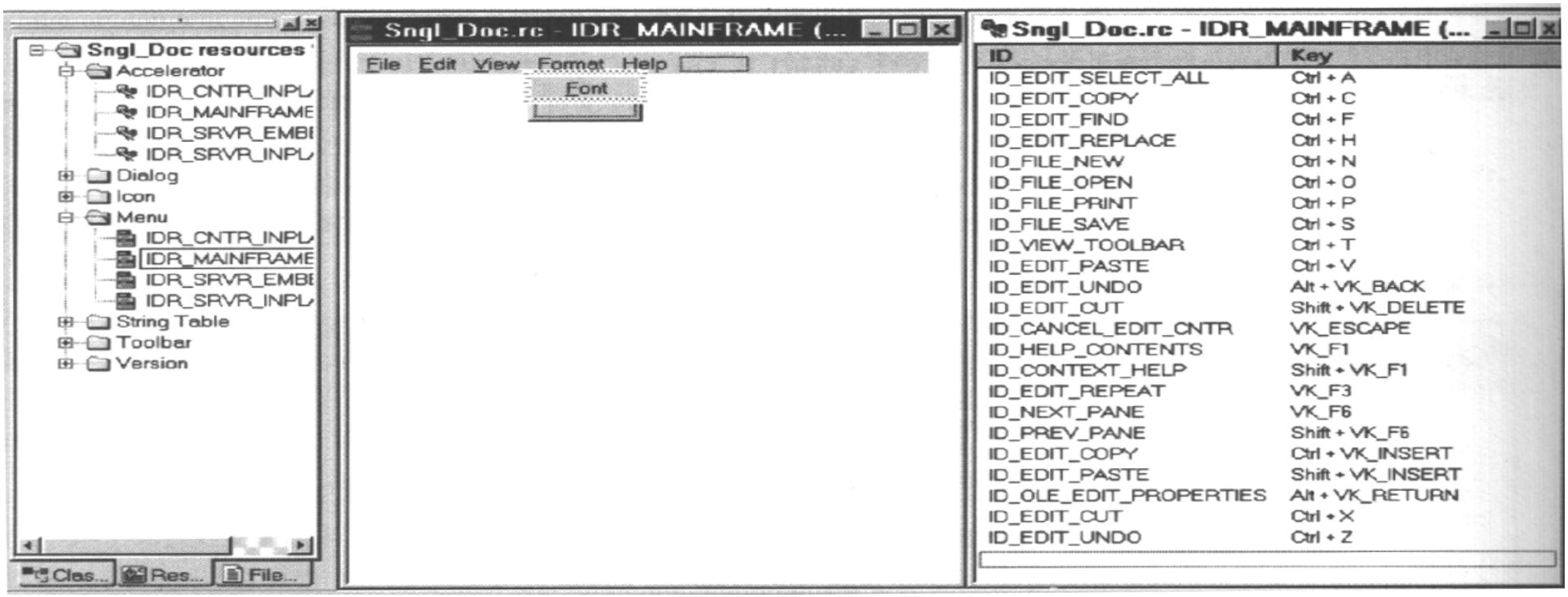


向菜单命令中添加新的文本，这一点很容易。打开 View（视图）菜单，然后右击 Toolbar 条目。从上下文菜单选择 Properties（属性），你会看到如下所示的 Menu Item Properties（菜单项属性）对话框。

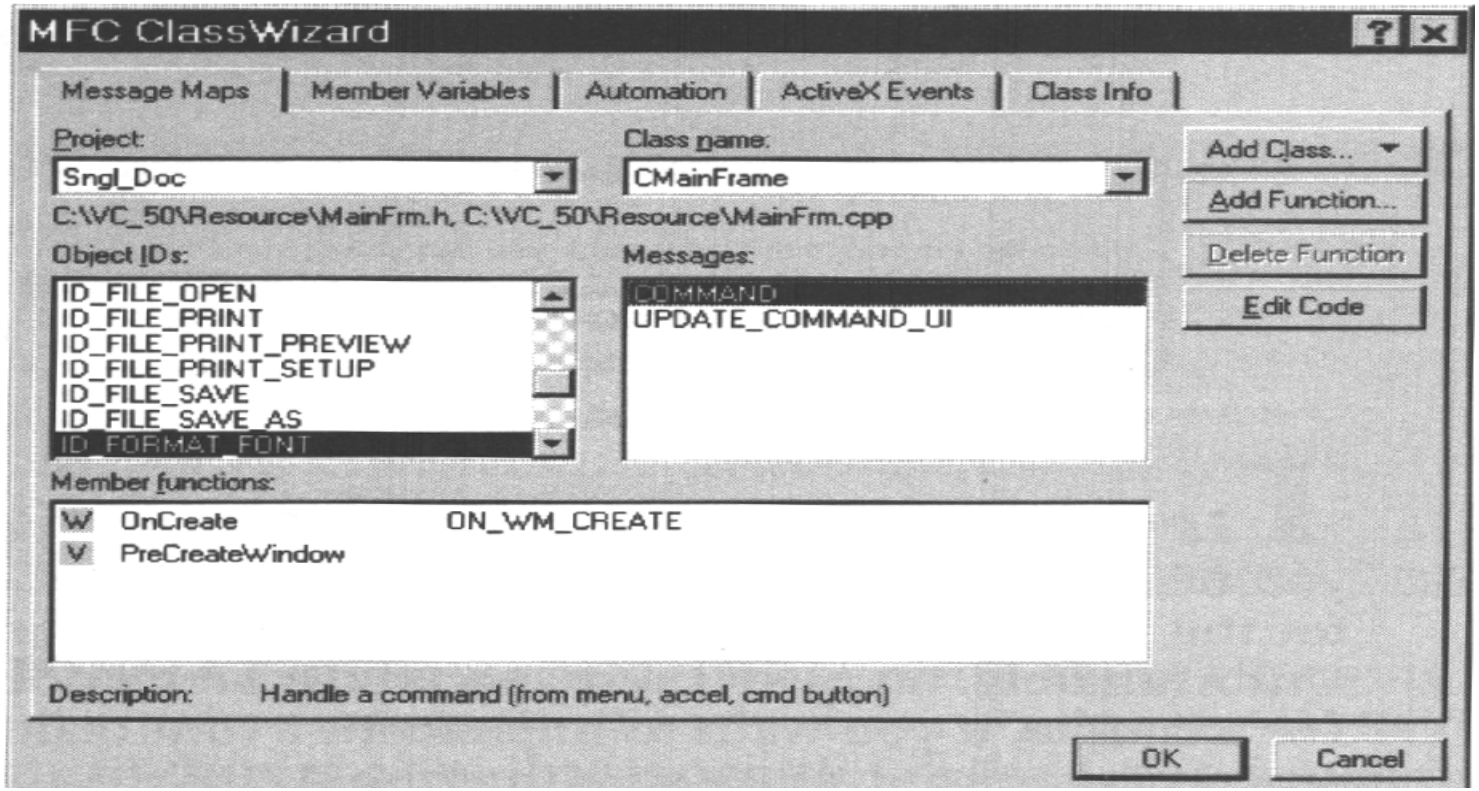
本例中，我们要做的是更改 Caption（标题）域以指明新的加速键。你可以使用通常用于格式化文本的 C 和 Windows 格式化字符。将 Caption 域修改成“&Toolbar\tCtrl-T”，告诉 Windows 你想看到单词“Toolbar”，字母"T"加上下划线，然后是一个空格，最后 CTRL-T 告诉用户该菜单命令的加速键是什么。

那么，如果想添加新的菜单该做些什么呢？只要选择菜单栏或现有菜单上的空白点，开始输入。系统会自动显示一个 Menu Item Properties（菜单项属性）对话框。在本实例中，我们将在工具条中添加一个 Format 菜单，该菜单只有一个菜单项 Font（请记住输入&Format 和&Font，这样每个菜单项的第一个字母才

能加上下划线）。添加了新的菜单项后，将 Format 移到 Help 菜单左侧。此时的菜单如下图所示。



现在我们要添加一些使这个新菜单项发挥作用的代码。右击 Font 菜单项，然后从上下文菜单中选择 ClassWizard，再选择 Message Maps 选项卡。你会看到 MFC ClassWizard 对话框，如下图所示。

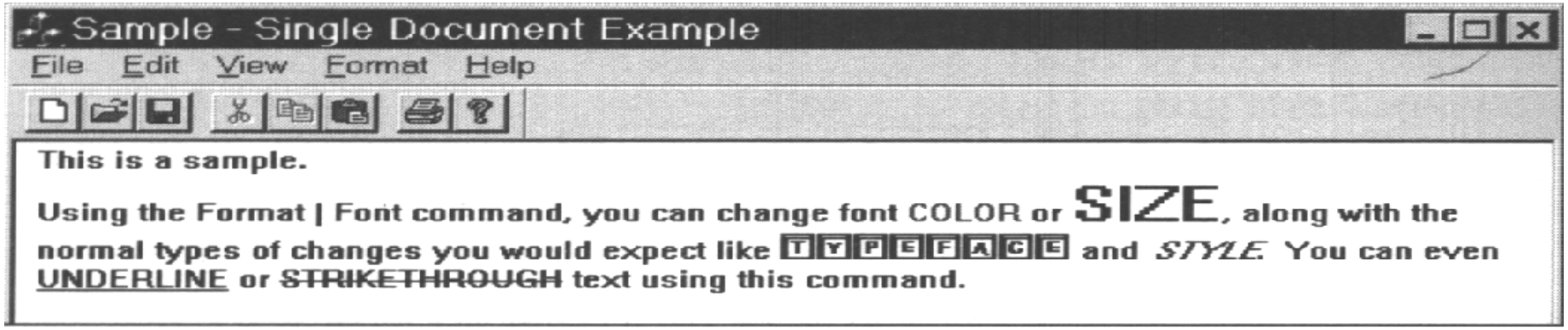


单击 Messages 列表中的 COMMAND。单击 Add Function（添加函数）按钮创建这个函数。你会看到 Add Member Function（添加成员函数）对话框，单击 OK 接受缺省的函数名。单击 Edit Code（编辑代码）按钮编辑这个新函数。程序列表 3.3 显示了向我们的程序中添加字体特性时所需的代码。虽然也可以将其添加到 CEdit 控件中，但只有富文本编辑屏幕（CRichEdit 控件）才原本就具有这种能力。

程序列表 3.3

```
void CMainFrame::OnFormatFont()
 {
      CFontDialog      oDialog;      //Create a font dialog.

      //Display the Font common dialog box.
      oDialog.DoModal();
 }
```

如你所见，在我们的示例中添加处理字体的能力简直太容易了。如果现在编译并运行这个程序，用户就能够改变缺省字体，或选择文本并改变所选文本的字体。下面是利用已经具备改变字体能力的这个示例程序修改字体方式的示例。

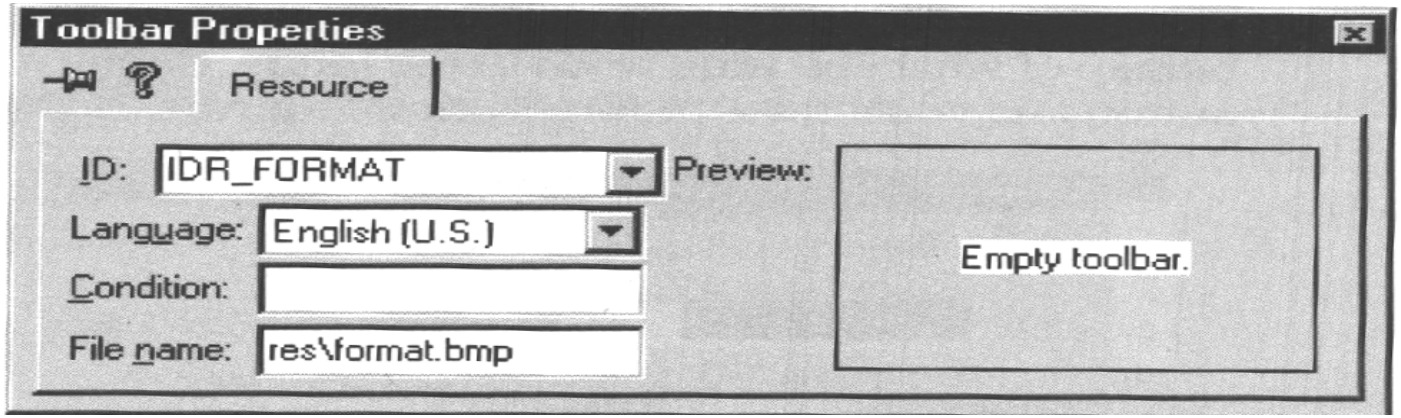


技巧 显示应用程序所需的大部分菜单并不需要做什么艰苦的努力。你真正需要完成的工作，只是运用各种菜单项的准确 ID 和工具条上的相关按钮。可惜的是，许多特殊的 ID 目前尚未列入文档。例如，如果想执行字体对话框而不做任何程序设计，要确保菜单项的 ID 是：ID_FORMAT_FONT。同样地，如果向应用程序中添加工具条按钮，工具条按钮应该与菜单项使用相同的 ID。可以在 MFC\INCLUDE 文件夹的 AFXRES.H 文件中找到所有特殊的 ID，无论它们是否已经列入文档。

3.3 使用工具条

如果说加速键是提高程序访问速度的键盘方法，那么工具条就和鼠标操作对应起来。你会看到，工具条不再仅仅是附属品了，它已经成为用户界面必不可少的一部分。但是，如果使用很多能够满足或者不能满足用户需要的按钮填满整个工具条的话，它很快就会变得过于臃肿。这个问题的一个解决办法是，创建多个工具条，允许用户自己决定当前需要哪些工具条。

使用工具条和使用菜单与加速键一样容易。不过在这里，必须创建工具条及其相关菜单命令之间的链接。缺省工具条 IDR_MAINFRAME 包括一些更为常见的按钮，比如打开文件或创建新文件时用到的按钮。



让我们以创建新的工具条来开始这个示例吧——这个工具条允许用户格式化文本。在 ResourceView 中右击 Toolbar 文件夹，然后从上下文相关菜单中选择 Insert Toolbar（插入工具条）。Visual C++会自动创建一个新的工具条。但是，它给出的名字(IDR_TOOLBAR1)并不具有清晰的意义。右击 IDR_TOOLBAR1 条目，然后从上下文菜单中选择 Properties（属性）。你会看到如下图所示的 Toolbar Properties（工具条属性）对话框。

在 ID 域中输入 IDR_FORMAT（不必担心要修改 File Name（文件名）域；修改 ID 域时它会自动修改）。完成修改后，单击 Close（关闭）框关闭 Toolbar Properties（工具条属性）对话框。

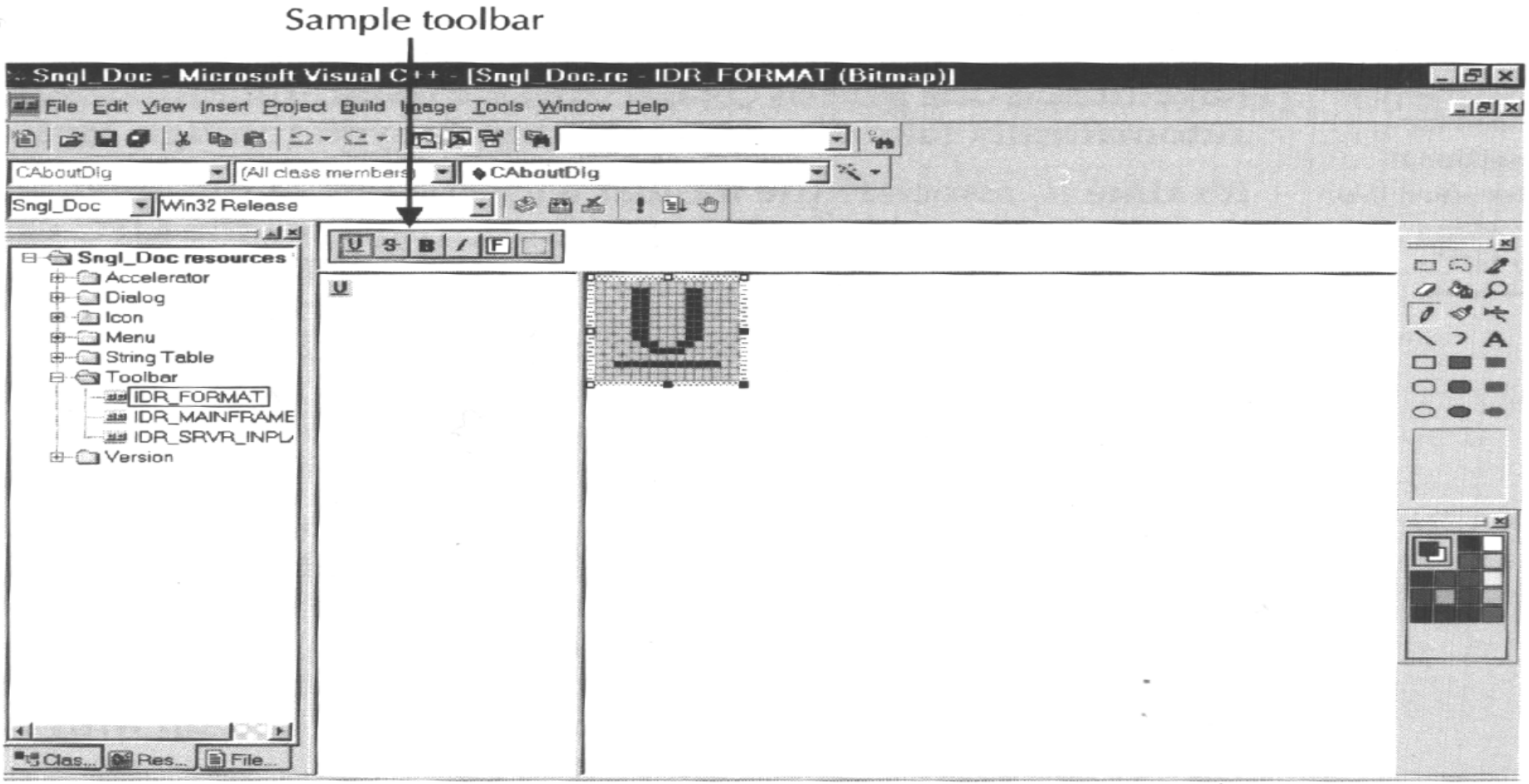
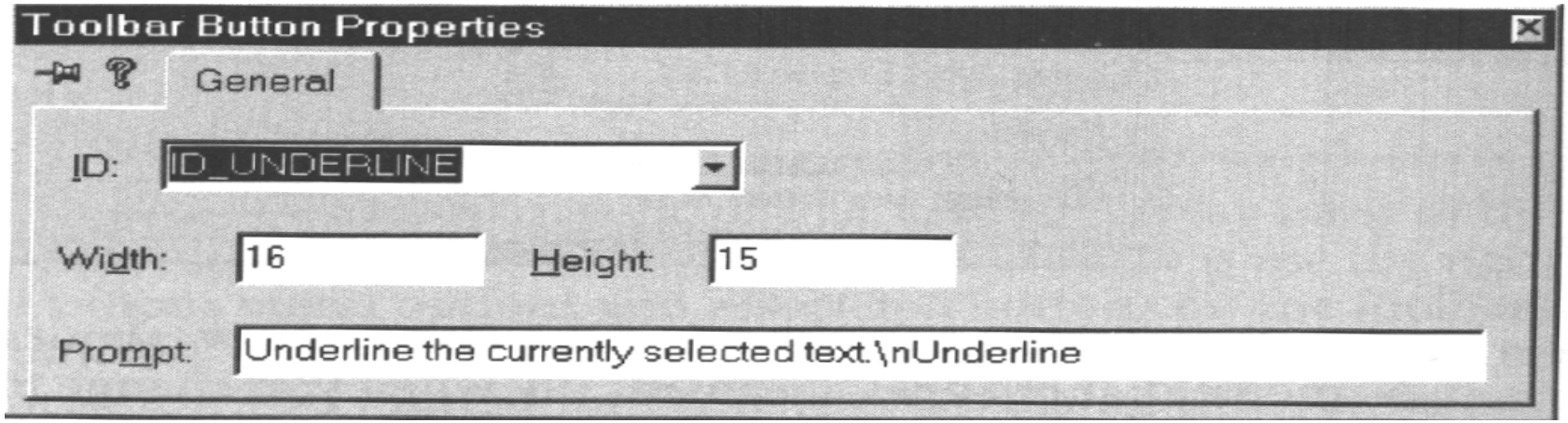


图 3.6　示例工具条上包括了五个按钮

现在我们要向这个工具条中添加一些按钮。这些按钮将允许用户执行各种各样的任务，而不必借助于使用键盘或在菜单系统间移动。图 3.6 显示了我们

将为本示例使用的样本工具条（这些按钮分别代表了加下划线、删除线、粗体、斜体和打开字体对话框）。

向工具条上添加按钮并不需要做太多的工作。你真正需要具备的东西是将来要进行操作的位图。双击下划线按钮，会看到 ToolBar Properties（工具条属性）对话框，如下所示。



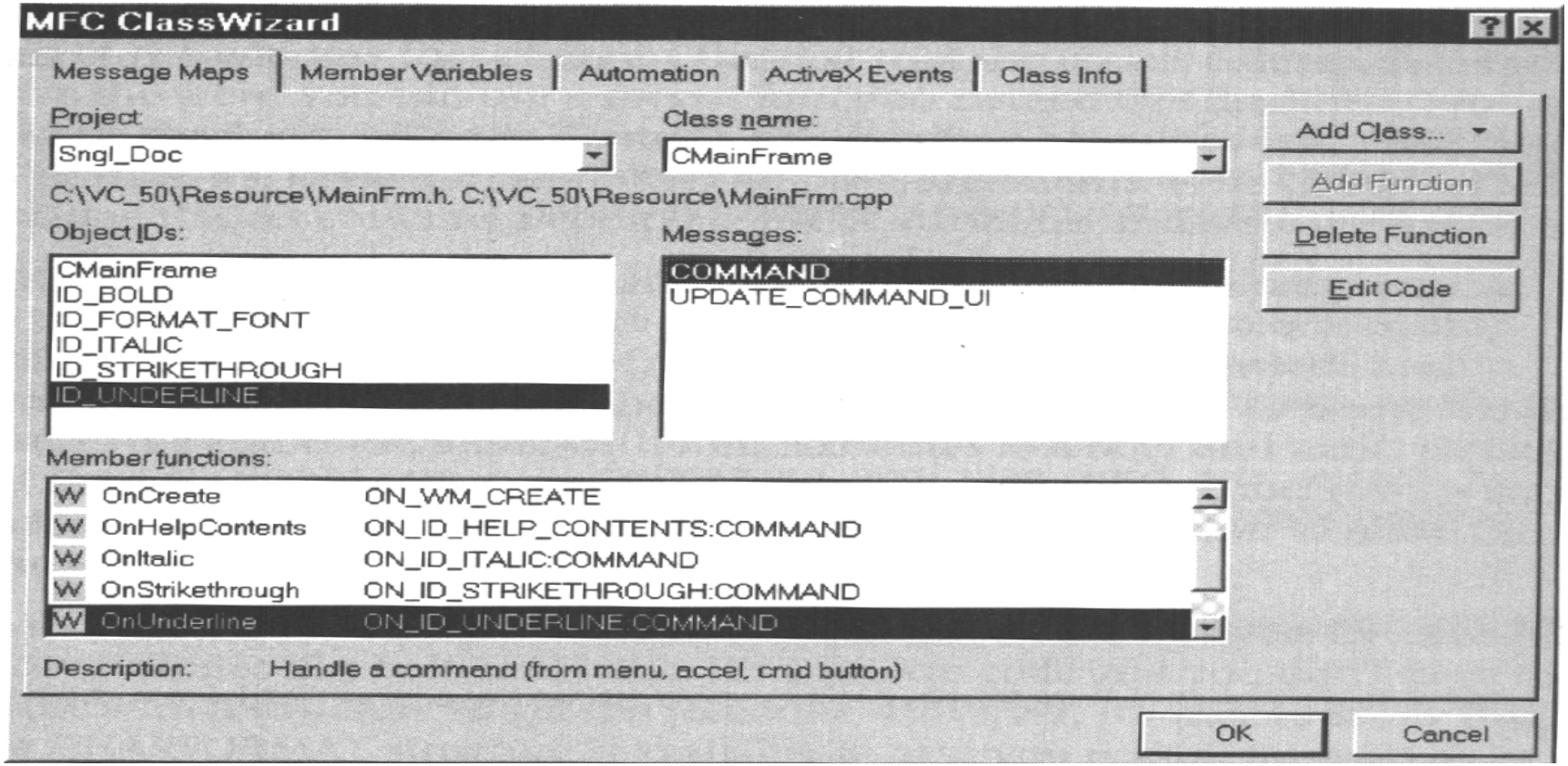
如你所见，我已经把 ID 域修改为 ID_UNDERLINE 了，将 Prompt（提示）域修改为 Underline（下划线）了。用相似的方法修改其它所有的按钮——ID_STRIKETHROUGH（删除线），ID_BOLD（粗体），ID_ITALIC（斜体）和 ID_FORMAT_FONT（字体对话框）。一定要仔细地输入所有的 ID，否则，在今后使用按钮时会出问题。我为最后一个按钮使用 ID_FORMAT_FONT 的原因是，减少所需的代码编写数量。使用这个 ID 意味着，无须添加一行代码就可以让这个按钮发挥作用，Visual C++会通过 MFC 自动处理这个按钮。

注 要了解 MFC 提供的其它标准 ID，请不要忘了浏览 MFC\INCLUDE 文件夹

中的 AFXRES.H 文件。

现在让我们把工具条与应用程序的其它部分联系起来。按住 CTRL 键并双击下划线按钮，你会看到 MFC ClassWizard（类向导），然后系统显示 Adding a Class（添加类）对话框。IDR_FORMAT 是一种新的资源，Visual C++不能确定你想用它做什么。你要把它和现有类联系起来，现在只要单击 OK，接受缺省设置。Visual C++会显示 Select Class（选择类）对话框。选择 CMainFrame 类，然后单击 Select（选择）。你现在已经把工具条和应用程序联系起来了。

创建和按钮相关的函数是很容易的。单击 MFC ClassWizard 的 Object IDS 列表中的按钮，比如 ID_UNDERLINE，然后单击 Messages（消息）列表中的 COMMAND。单击 Add Function（添加函数）按钮创建函数。Visual C++显示 Add Member Function（添加成员函数）对话框。只需单击 OK 接受缺省函数名。最后，你会看到一个成员函数列表，如下图所示。



现在我们可以向程序中添加使按钮发挥作用的代码了。单击任一成员函数名（比如 OnUnderline），然后单击 Edit Code 按钮。Visual C++会显示代码编辑区。你会看到我们刚刚创建的函数的框架。程序列表 3.4 显示了需要添加的代码。

程序列表 3.4

```
void CMainFrame::OnBold()
```

```
{
        CRichEditView*        poView;        // Create a variable to hold our view.

        // Get the active view from the current window.Typecast it as a
        // CRichEditView rather than a CView, the standard return value.
        poView = (CRichEditView*) GetActiveView();

      // Change the font settings as needed.
      poView->OnCharEffect(CFM_BOLD, CFE_BOLD);
}
void CMainFrame::OnItalic()
{
      CRichEditView*          poView;       // Create a variable to hold our view.
      CHARFORMAT              cfFont;       // Create a structure for the font data.

      // Get the active view from the current window. Typecast it as a
      // CRichEditView rather than a CView, the standard return value.
      poView = (CRichEditView*) GetActiveView();

      // Get the current font settings, then change them to Italic.
      cfFont = poView->GetCharFormatSelection();
```

```
        cfFont.dwMask |= CFM_ITALIC;
        cfFont.dwEffects |= CFE_ITALIC;

        // Change the font settings as needed.
        poView->SetCharFormat(cfFont);
}

void CMainFrame::OnStrikethrough()
{
        CRichEditView*        poView;        // Create a variable to hold our view.
        // Get the active view from the current window. Typecast it as a
        // CRichEditView rather than a CView, the standard return value.
        poView = (CRichEditView*) GetActiveView();

        // Change the font settings as needed.
        poView->OnCharEffect(CFM_STRIKEOUT, CFE_STRIKEOUT);
}
void CMainFrame::OnUnderline()
{
        CRichEditView*        poView;        // Create a variable to hold our view.
```

```
        // Get the active view from the current  window. Typecast it as a
        // CRichEditView rather than a CView, the standard return value.
        poView = (CRichEditView*) GetActiveView();

        // Change the font settings as needed.
        poView->OnCharEffect(CFM_UNDERLINE, CFE_UNDERLINE);
    }
```

正如从源代码中所看到的，对选定的一组字符，有两种不同的方法更改字符的字体属性。第一种方法要容易一些，只需得到活动视图——即包含用户正编辑文本的窗口部分。一旦拥有了这个视图，就可以用名为 OnCharEffect()的特殊函数更改字体属性。要使这个函数真正发挥功能，还要为两个参数提供相同的字体属性（CHARFORMAT 文档包含了一系列完整的属性及相关定义）。

第二种方法要求再做一些工作，但同时它也更灵活。这时，必须获得活动视图的一个拷贝。但是这一次，用它把当前字体特性填入 CHARFORMAT（特性格式）结构。该结构包括你需要知道的一切，比如字体名和颜色，以及字体属性，比如粗体和斜体。一旦获得已填好的 CHARFORMAT（特性格式）结构，就在屏幕上更改想修改的成员，然后用 SetCharFormat()函数做实际的更改。

在很多情况下，你会想用我演示的第一种方法改变字体属性，如粗体和斜体。这样做的话，编码要少得多，并且实现起来也不必非得与结构打交道。但是，在进行更详细的屏幕更改时可以使用 CHARFORMAT（字符格式）结构，了解这一点对编程工作将会起到重要作用。

我们已经有了工具条和一些使其发挥作用的代码。我们的示例程序仍然缺少一个重要的特征。如果现在就运行它，会看不到工具条。最后一步是添加菜单项和一些让工具条用起来更方便的代码。我们从菜单项开始。我要做的，只是用和讨论 Format（格式）菜单时一样的过程，向 View（视图）菜单中添加一个新的菜单项。在 Menu Item Properties（菜单项属性）对话框中，我把 ID_VIEW_FORMATTOOLBAR 用作 ID，把 &Format Toolbar 用作标题，把显示或隐藏格式工具条的\nToggle Format ToolBar 用作提示。由于我们缺省显示工具条，所以你还要选中 Checked 复选框。

创建程序框架相当容易。按下 CTRL 键并双击新的 Format Toolbar（格式工具条）菜单项。你会看到 MFC ClassWizard（MFC 类向导）对话框。Visual C++ 应该自动加亮 Object Ids（对象 ID）列表中的 ID_VIEW_FORMATTOOLBAR 项。加亮 Messages（消息）列表中的 COMMAND 选项，然后单击 Add Function（添加函数）。最后，单击 Edit Code（编辑代码）显示代码窗口。

有三个地方需要添加工具条代码。第一块代码出现在 MAINFRAME.H 文件中。要在初始工具条变量右下方的 Protected（受保护）节中添加新的变量。新的声明变量的代码如下所示：

```
CToolBar    m_wndToolBar2;
```

下一块代码出现在 MAINFRAME.CPP 文件中（见程序列表 3.5）。这块代码设置工具条的特性，使你在启动程序时能看见它。请注意，为了使工具条成为可“浮动”工具条（这样可以在应用程序中把它从一个地方移到另一个地方），要考虑进行一些特殊的编码。

程序列表 3.5

```
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    if (!m_wndToolBar.Create(this)  ||
        !m_wndToolBar.LoadToolBar(IDR-MAINFRAME))
    {
        TRACE0("Failed to create toolbar\n");
        return  -1;                    // fail to create
    }
    if  (!m_wndToolBar2.create(this) ||
         !m_wndToolBar2.LoadToolBar(IDR_FORMAT))
    {
         TRACE0(Failed to create toolbar\n");
         return -1;                    // fail to create
    }

    if (!m_wndStatusBar.Create(this) ||
        !m_wndStatusBar.SetIndicators(indicators,
```

```
        sizeof(indicators)/sizeof(UINT)))
{
    TRACE0("Failed to create status bar\n");
    return    -1;          // fail to create
}

// TODO:Remove this if you don't want tool tips or a resizeable toolbar
m_wndToolBar.SetBarStyle(m_wndToolBar.GetBarStyle() |
    CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC);
m_wndToolBar2.SetBarStyle(m_wndToolBar2.GetBarStyle() |
    CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC);

// TODO: Delete these three lines if you don't want the toolbar to
// be dockable
m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
m_wndToolBar2.EnableDocking(CBRS_ALIGN_ANY);
EnableDocking(CBRS_ALIGN_ANY);
DockControlBar(&m_wndToolBar);
DockControlBar(&m_wndToolBar2);

return 0;
```

```
}
```

有三个需要使用工具条代码的主要区域。第一个主区创建工具条，然后将IDR_FORMAT 工具条加载进去。如果这一过程失败，那么在应用程序启动前，会得到“创建失败”的消息。第二个主区定义工具条风格。我使用缺省设置，允许用户调整工具条的大小并将其四处移动。鼠标停留在按钮上时，还会显示工具提示。代码的最后一节使工具条拼接化（Docking），定义用户可以在哪里停放工具条，并实际停放我们已创建的工具条。此时，工具条是可见的；用户可以将其四处移动，并用 Close（关闭）框将其从视线清除。

使菜单命令 View | Format Toolbar（视图|格式工具条）发挥作用相当容易。程序列表 3.6 显示了使这一部分程序发挥作用所需的代码。

程序列表 3.6

```
void CMainFrame::OnViewFormattoolbar()
{
        CMenu*      poMenu;          // Create a pointer to the current menu.

     poMenu = GetMenu();          // Get the menu.

    // Determine if the View | Format Toolbar option is checked. If it
    // is, then hide the format toolbar and uncheck the option. Otherwise,
    // display the toolbar and check the menu item.
```

```
        if (poMenu->GetMenuState(ID_VIEW_FORMATTOOLBAR, MF_CHECKED))
        {
            showControlBar(&m_wndToolBar2, FALSE, FALSE);
            poMenu->CheckMenuItem(ID_VIEW_FORMATTOOLBAR,
MF_UNCHECKED);
        }
        else
        {
            ShowControlBar(&m_wndToolBar2, TRUE, FALSE);
            poMenu->CheckMenuItem(ID_VIEW_FORMATTOOLBAR,
MF_CHECKED);
        }
    }
```

如你所见，我们是从获得 CWnd 类菜单对象的拷贝开始的。当我们得到了这个菜单对象后，要判断当前用户是否选择了 Format Toolbar（格式工具条）选项就变得很容易了。如果选择了这个选项，那么工具条是可见的。把 ShowControlBar()函数的第二个和第三个参数设成 false，可以使工具条不可见。CheckMenuItem()允许你从 View | Format Toolbar（视图|格式工具条）菜单选项去掉复选标记。反过来，用相反的过程使工具条可见并再次选择菜单选项。

继续下去，再编译一次应用程序，这样你才能熟悉我们刚刚添加的各种特

征。一定要试验所有的格式化选项和连接及隐藏工具条的功能。很明显，这个应用程序并不像你现在看到的一些程序那样复杂，但它确实可以充分地使用资源。下面是我的程序版本的外观，其中工具条尚未连接在一起。