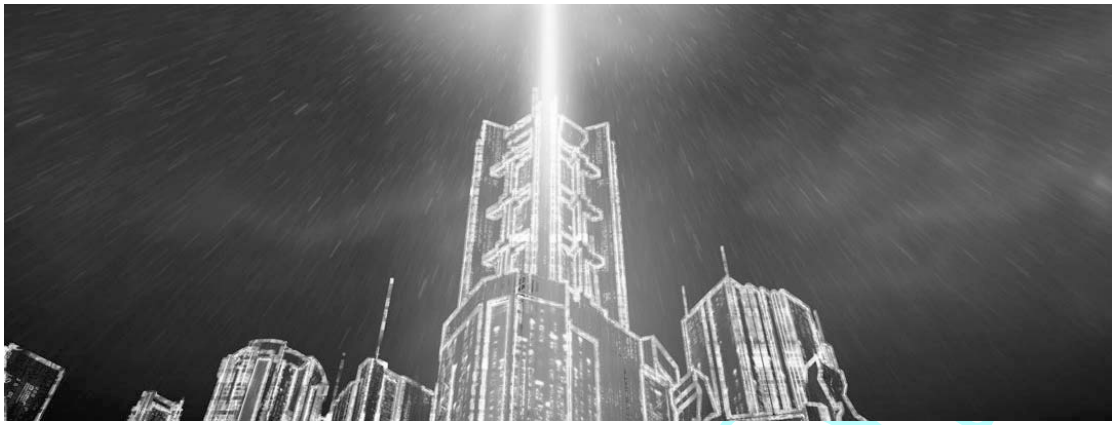
学习 MicroStation VBA

Jerry Winters 著

奔特力学院出版
Exton,PA
2006 年



前　言

(奔特力系统公司，John Gooding)

MicroStation VBA 是 MicroStation V8 对 Microsoft Visual Basic for Application(VBA)引擎的实现，它为 MicroStation 用户和开发者提供了大量功能— 从简单的自定义到与其它 Windows 应用的紧密集成。借助于 VBA，用户能定义 MicroStation 自动完成用户所需的重复任务。

许多用户已经发现，用 VBA 来使工作更容易其实很简单。使用标准的用户界面，采用世界上最流行的编程语言，轻松与 Microsoft Office 以及其它应用程序连接和通讯，所有这些仅仅是 VBA 所提供的许多功能的一部分。

虽然 MicroStation VBA 很流行，但您可能属于那些踌躇于甚或是猜疑于探索 MicroStation 编程的人们。您大可不必这样！VBA 是最容易掌握的编程环境，《学习 MicroStation VBA》这本书是帮助您学习的最佳指南。

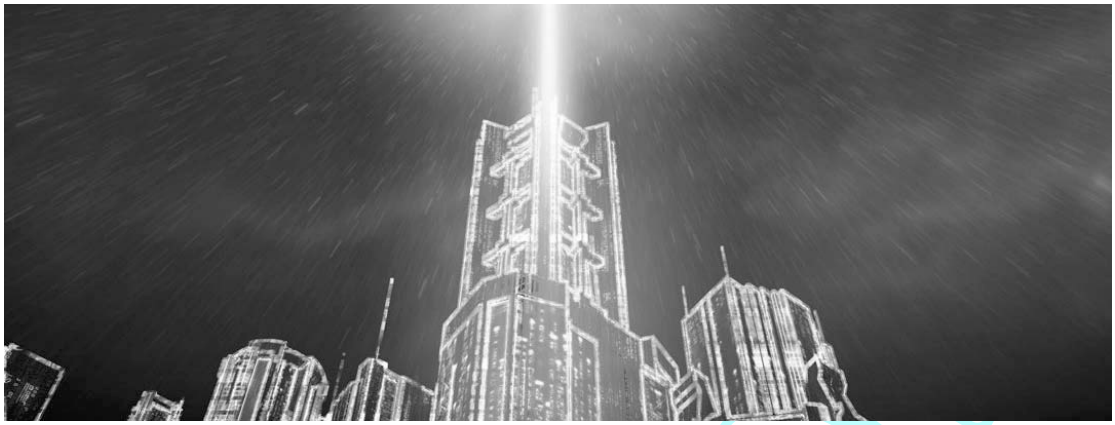
一本想适合所有读者的书经常是要么缺少初学者所需的基础资料，要么缺少专家所需的深层次技术信息。然而，《学习 MicroStation VBA》是一部同时服务于初学者和专家的少见的杰出作品。这本全面的书籍集 Jerry Winters 先生广博的 VBA 专业技能和 MicroStation 基础知识于一身。

介绍性章节仔细而彻底地带领新用户遍历了 Visual Basic for Application 的基础。从其对 VBA 集成开发环境的详细回顾开始，经过其对如何和怎样使用窗体和类模块的细心描述，再到其对对象模型的全面解释而终，Jerry 先生确保初学者不仅具有一本如何使用 VBA 的指南，而且还使初学者从其深厚的洞察力中获益。Jerry 先生告诉了我们，在为 MicroStation 建立专业级的应用时如何更好地使用 VBA 工具。

专家级的 VBA 程序员同样会发现 Jerry 先生的这本书是一本无价的参考工具书，这本书能帮助他们开拓 VBA 所提供的功能。例如，Jerry 先生对 XML 的概述以及在 MicroStation VBA 中如何调用 Windows API 能提高每位程序员的知识水平。

但对于大多数的 MicroStation 用户来说，这本书应当能帮助您最终摆脱对编写程序的恐惧。这本书中含有开始编写 MicroStation 程序和自动化您最通常任务的所有资料。

China BDN



目　录

1. VBA介绍 1

什么是VBA？ 1

为什么要学习VBA？ 1

什么时候应该使用VBA？ 2

怎样使用VBA？ 2

VBA是什么样的？ 6

本章回顾 7

2. VBA工程管理器 8

VBA工程管理器的功能 9

宏对话框 10

本章回顾 11

3. VBA IDE 12

菜单 13

File(文件)菜单 13

Edit(编辑)菜单 14

View(视图)菜单 15

Insert(插入)菜单 15

Format(格式)菜单 16

Debug(调试)菜单 16

Run(运行)菜单 16

Tool(工具)菜单 17

China BD

Add-Ins(外接程序)菜单 17
Window(窗口)菜单 18
Help(帮助)菜单 18
工具栏 18
标准工具栏 19
编辑工具栏 19
调试工具栏 19
用户窗体工具栏 19
窗口 19
工程资源管理器 20
对象浏览器 21
属性窗口 21
监视窗口 22
本地窗口 22
立即窗口 23
调用堆栈窗口 23
工具箱窗口 24
其它窗口 25
本章回顾 26
4. 找到帮助 27
术语 27
帮助文件 29
内容选项卡 30
索引选项卡 31
搜索选项卡 31
书签选项卡 33
MicroStation VBA帮助文件 34
网络 35
对象浏览器 37

China BDN

本章回顾 39
5. 模块、窗体和类模块 40
模块 40
窗体 44
类 46
过程和函数 50
返回一个数组 55
返回“类型” 57
返回对象 58
ByVal和ByRef(按值和按引用传递) 59
声明变量 61
Optional Explicit 62
本章回顾 62
6. 变 量 64
标准VBA变量类型 64
Integer(整型) 65
Long(长整型) 65
Double(双精度型) 65
Boolean(布尔型) 66
Date(日期型) 67
String(字符串型) 67
Object(对象型) 67
Variant 67
MicroStation特有的变量类型 67
Application(应用) 68
DesignFile(设计文件) 68
ModelReference(模型参考) 68
Level(层) 69
LineElement(线元素) 69

China BDN

EllipseElement(椭圆元素) 69
ArcElement(弧元素) 70
TextElement(文本元素) 70
赋值和设置对象 71
数组(Array) 71
常数 72
变量名 72
区分大小写 73
Optional Explicit 74
使用变量 75
本章回顾 75
7. 操作文本 76
VBA字符串函数 77
UCase(大写转换) 77
LCase(小写转换) 77
StrConv(字符串转换) 78
WeekDayName(星期名)、WeekDay(日期转星期) 78
MonthName(月名) 79
LTrim(左截取)、RTrim(右截取)、Trim(两端截取) 80
StrComp(字符串比较) 80
Len(取字串长度) 84
Left(左取子串) 84
Right(右取子串) 85
Mid(取中串) 85
Replace(替换字串) 86
InStr(子串位置) 87
InStrRev(反向子串位置) 89
Split(分割)和Join(联接) 89
Asc(字符到ASCII码)和Chr(ASCII码到字符) 91

China BDN

FormatCurrency(格式化货币) 92
FormatNumber(格式化数字) 93
FormatDateTime(格式化日期) 94
Format(格式化) 94
&(连接符) 95
VbCr(回车符) 95
VbTab(制表符) 95
本章回顾 95
8. 操作数字 96
数字函数 96
加 96
减 98
乘 98
除 99
平方与指数 99
平方根 100
正弦、余弦、正切 100
反正切 103
绝对值 104
转换为整数、长整数、双精度数和值 104
CLng 105
Fix 105
CDbl 106
Val 106
IsNumeric 106
Round 107
Mod—求余 108
Sgn—求数的正负号 108
Rnd与Randomize 108

China BDN

End Sub 操作符的优先级 109
本章回顾 109
9. 标准VBA调用 110
消息框 110
输入框 113
Now 115
DateAdd 115
DateDiff 116
Timer 116
FileDateTime 117
FileLen 117
MkDir 117
RmDir 118
Dir 118
Kill 120
Beep 121
SaveSetting 121
GetSetting 121
用GetSetting能取得注册表中的设置。 121
DeleteSetting 122
GetAllSettings 122
读写ASCII文件 123
FreeFile 124
从ASCII文件中读 127
控制代码的执行 129
For … Next 129
While … Wend 131
Do … Loop 132
For Each … Next 135
If … Then 135

China BDN

Select Case 136
错误处理 137
本章回顾 144
10. 可视界面 145
属性、方法和事件 146
属性 146
控件事件 147
通用控件属性 149
Name(名称) 149
Left(左)、Top(顶) 149
Width(宽)、Height(高) 150
Visible(可见) 150
Enabled(有效) 150
TabStop(制表符站) 150
TabIndex(制表符索引) 150
Tag(附加信息) 150
ControlTipText(控件提示文本) 151

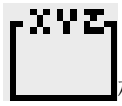
标签 151

文本框 151
属性 151
事件 152

复合框 152
属性 152
方法 153
事件 153

列表框 153

China BDN

属性 153
方法 154
事件 154
复选框 154
属性 154
事件 154
选项按钮 154
属性 155
事件 155
切换按钮 155
属性 155
事件 155
框架 156
属性 156
命令按钮 156
属性 156
事件 156
TabStrip(选项卡条) 156
属性 156
方法 157
事件 157
多页 157

China BDN

属性 157
方法 157
事件 157
滚动条 157
属性 157
事件 158
旋转按钮 158
属性 158
事件 158
图像 158
属性 159
用户界面练习 159
点列表读取器 165
写出到文件 169
缩放和平移 173
本章回顾 177
11. MicroStation对象模型—对象 178
对象浏览器 178
自动列出成员 180
MicroStation VBA帮助文件 181
添加监控 182
MicroStation对象模型 183
Application对象 183
本章回顾 238
12. MicroStation对象模型—枚举 239
MsdDesignFileFormat 239

China BDN

枚举列表 240
本章回顾 277
13. MicroStation对象模型—类型 278
本章回顾 282
14. MicroStation对象模型—事件 283
OnDesignFileOpened和OnDesignFileClosed 284
本章回顾 285
15. 向文档中添加内容 286
图形元素 286
线 286
建立形 292
建立圆 295
建立椭圆 299
建立弧 300
建立文本 303
建立单元 304
建立新文档 308
建立数据的安全性问题 310
本章回顾 310
16. 在文件中查找 311
在文件中查找的基础知识 311
使用ScanCriteria 317
多个准则的组合 322
回顾三种集合方法 326
扫描准则的方法 326
本章回顾 328
17. 交互式修改 329
向用户提供反馈和信息 329
操作选择集 331

China BDN

取得用户输入 333
一些实际应用程序 337
使用SendCommand 349
非模态对话框 355
frmMatchProperties.frm 355
向用户提供反馈和信息 361
用户窗体初始化(UserForm Initialize) 362
frmAlignText.frm 364
frmExportElements.frm 380
frmDFAV.frm 384
与MDL应用交互 391
本章回顾 394
18. 接口的本质 396
接口基础 397
类模块回顾 398
ILocateCommandEvents 401
LocateFilter事件 403
Accept事件 403
LocateReset事件 403
LocateFailed事件 403
Start事件 404
Cleanup事件 404
Dynamics事件 404
LocateCriteria(定位准则) 409
IPrimitiveCommandEvents 411
优化Dynamics(动态)事件 432
本章回顾 435
19. 使用MicroStation内置用户窗体 436
声明MicroStation用户窗体函数 436

China BDN

mdlDialog_fileOpen函数 437
mdlDialog_fileCreate函数 444
mdlDialog_fileCreateFromSeed函数 446
mdlDialog_openAlert函数 447
mdlDialog_openInfoBox函数 448
本章回顾 449
20. 类模块 450
封装相似功能 450
建立带有属性、方法和事件的对象 469
在集合中使用类模块 476
存取集合中的对象 478
从集合中移除对象 480
使用自定义类模块 481
本章回顾 484
21. CAD管理员使用的VBA 486
用VBA来维护标准 486
跨公司的标准 493
跟踪时间 498
绘图员 498
经理 499
会计师 499
自动装载和自动运行 502
MS_VBA_OPEN_IN_MEMORY 504
保护工程 506
分发VBA工程 508
在高安全模式下工作 509
本章回顾 510
22. MicroStation中基于文件的事件 511
OnDesignFileOpened 511
OnDesignFileClosed 512

China BDN

ISaveAsEvents接口 515
本章回顾 522
23. 响应MicroStation附件事件 523
IAttachmentEvents接口 523
AfterAttach 524
AfterDetach 528
AttachmentModified事件 528
BeforeAttach事件 528
BeforeDetach事件 529
本章回顾 530
24. 模型事件 531
本章回顾 534
25. 层事件 536
Active事件 540
AfterCreate事件 540
AfterDelete事件 540
BeforeChangeActive事件 541
BeforeDelete事件 541
ChangeAttribute事件 541
本章回顾 541
26. 改变跟踪事件 542
BeginUndoRedo事件 542
ElementChanged事件 544
例 1 546
例 2 548
例 3 553
例 4 554
激活ChangeTrackEvents接口 556
本章回顾 556

China BDN

27. 非图形信息—数据库 557
MicroStation是如何将元素“链接”到数据库的 557
从头开始建立数据库 559
使用UDL文件 568
链接MicroStation元素到数据库记录 569
用SQL建立数据库记录 571
建立一个查看数据库信息的用户界面 571
本章回顾 574
28. 标签 576
从基于选择集的标签中取得信息 576
取得文件中的所有标签 579
使用标签集 580
取得一个文件夹下所有文件中的所有标签 581
改变标签的值 583
改变多个文件中的多个标签值 584
把标签信息导出到一个文件中 586
本章回顾 590
29. XML 592
XML是什么？ 592
XML文件结构 592
读取XML文件 594
本章回顾 606
30. 批处理 607
处理列在一个ASCII文件中的文件 607
处理一个文件夹下的所有文件 611
处理一个文件夹及其子文件夹下的所有文件 615
建立选择文件的用户界面 617
记录批文件处理过程 622
使用记录文件 622

China BDN

用数据库跟踪活动 624
在注册表中保存信息 624
通过Internet记录活动 625
电子邮件处理记录 628
本章回顾 631
31. 标准检查器 632
标准检查器的基本实现 632
标准检查A 634
标准检查器的设置 639
检查标准 641
标准检查报告 647
自动装载自定义标准检查器外接程序 656
本章回顾 656
32. 使用Windows API 657
声明API调用 657
声明类型 658
利用API调用 659
GetLogicalDrives 659
GetDriveType 660
GetComputerName 662
GetVersionEx 662
Sleep 664
FindExecutable 665
GetDiskFreeSpace 666
GetSystemMetrics 667
GetTickCount 669
GetUserName 669
GetWindowsDirectory 670
LogonUser 670
MessageBeep 672

China BDN

PlaySound 673
ShellExecute 674
SHGetFileInfo 675
本章回顾 676
33. 使用第三方ActiveX控件和DLL 677
使用ActiveX控件 677
使用现有的DLL 679
Microsoft Scripting Runtime 683
Microsoft Speech Object Library 689
Microsoft CDO for Windows 2000 Library 689
DSO OLE Document Properties Reader 2.0 691
本章回顾 698
34. 操作Excel 699
连接到Excel 699
GetObject 699
CreateObject 701
New 702
工作簿、工作表、范围和单元格 702
单元格和范围的地址 707
操作工作表 712
标签提取 719
本章回顾 724
35. 操作数据库(ActiveX数据对象) 725
ActiveX数据对象入门 725
UDL文件基础 727
连接、记录集以及更多内容 731
记录集 736
Find方法 740
GetString方法 741

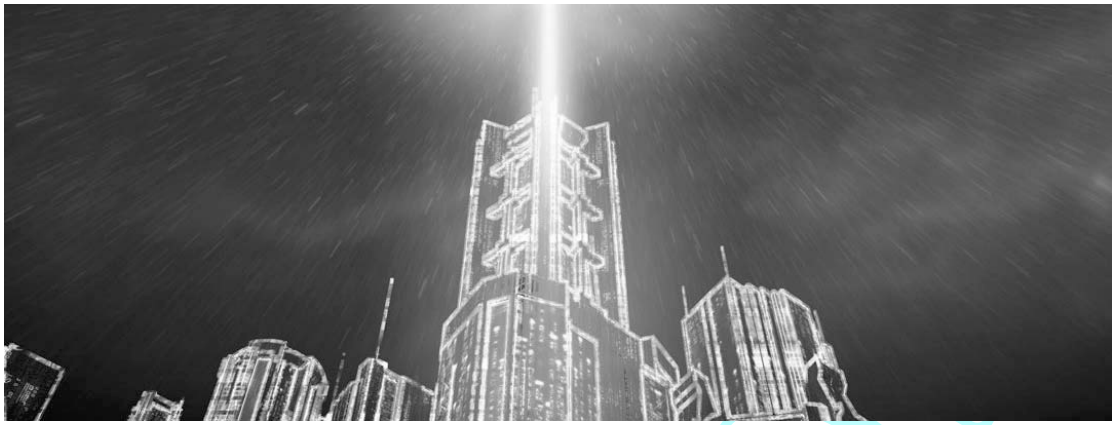
China BDN

AddNew和Update方法.....742
SQL要素.....743
Select语句.....744
Where.....745
Order By.....745
扩展ActiveX数据对象.....754
分析数据库架构.....759
Excel文件作为数据库.....762
本章回顾.....766
36. MicroStation用VBA操作Mathcad.....767
Mathcad的简要介绍.....767
添加引用以及使用对象浏览器.....768
与Mathcad通讯的基本宏.....775
区域对象 — 所有计算的基础.....778
Mathcad对象模型.....781
Application.....782
IMathcadApplication2.....782
从Mathcad中驱动MicroStation几何图形.....782
本章回顾.....795
37. 从外部程序存取MicroStation数据.....796
ActiveX/COM基础.....796
引用、前期绑定和后期绑定.....798
GetObject、SetObject和New.....800
什么时候使用GetObject、CreateObject和New.....803
“WithEvents”能干什么？.....804
从Excel还是从MicroStation中运行宏？.....805
从Excel中控制MicroStation.....806
运行Excel宏.....809
本章回顾.....817

China BDN

38. 编写VB6 应用程序 818
VBA与VB6 之间的不同 818
VB6 工程的结构 819
用VB6 控制MicroStation 826
在VB6 中建立ActiveX控件 833
调试ActiveX控件工程 838
编译ActiveX控件 840
建立ActiveX DLL 842
编译和分发应用程序 848
编译应用程序 848
分发VB6 应用程序 849
本章回顾 858
39. 使用VB.NET 859
VB.NET介绍 859
在VB.NET可以完成的工作 865
DGN浏览器应用程序 872
VBA到VB.NET引用 878
一切皆对象 878
重载 879
过程和函数 879
存取ASCII文件 880
遍历文件夹及其子文件夹 883
取得一个路径中的所有文件 883
返回函数值 884
Windows API调用 884
分发VB.NET应用程序 890
本章回顾 893
附加资源.. 894

China BDN



1. VBA 介绍

“让我们从最初级开始。这是最好的起点。”

- 什么是VBA?
- 我们为什么要学习它?
- 什么时候应该使用它?
- 怎样使用它?
- 它是什么样的?

这是五个非常好的问题，很值得花时间来回答。

什么是 VBA?

VBA 是 Visual Basic for Application 的缩写。微软把 VBA 授权给像奔特力系统这样的公司，所以被授权者就能够定制其公司软件。这样，开发世界级软件(如 MicroStation)的公司就能给其客户以最优秀的工具集。这些公司知道，实现软件定制这个目标的一种途径就是能让客户自己按需修改和个人化它们的软件。

为什么要学习 VBA?

为了能快速开发满足个人需要的程序而学习 VBA。在 MicroStation VBA 环境中所学到的大多数知识也能用在其它的 VBA 环境中。VBA 的前三分之二缩写是“VB”。Visual Basic 包含可视化 Basic 编程语言及其编程环境。“A”完成了 VBA 的最后三分之一。“A”就是特定应用的对象和应用编程接口(API)。

如果把 VBA 看作是三分之二的 Visual Basic 和三分之一的应用，那么您在本书中学到的三分之二的知识都能直接用到其它的 VBA 环境中。例如，如果学

会了 MicroStation VBA，您将懂得 Microsoft Excel VBA 的 2/3。这一切并不遥远。所以，除了能定制 MicroStation 满足您的需要以外，学习 VBA 还能使您与内嵌了 VBA 的其它应用程序打交道。

什么时候应该使用 VBA?

仅在成人直接监督下?

在上午八时至下午五时之间?

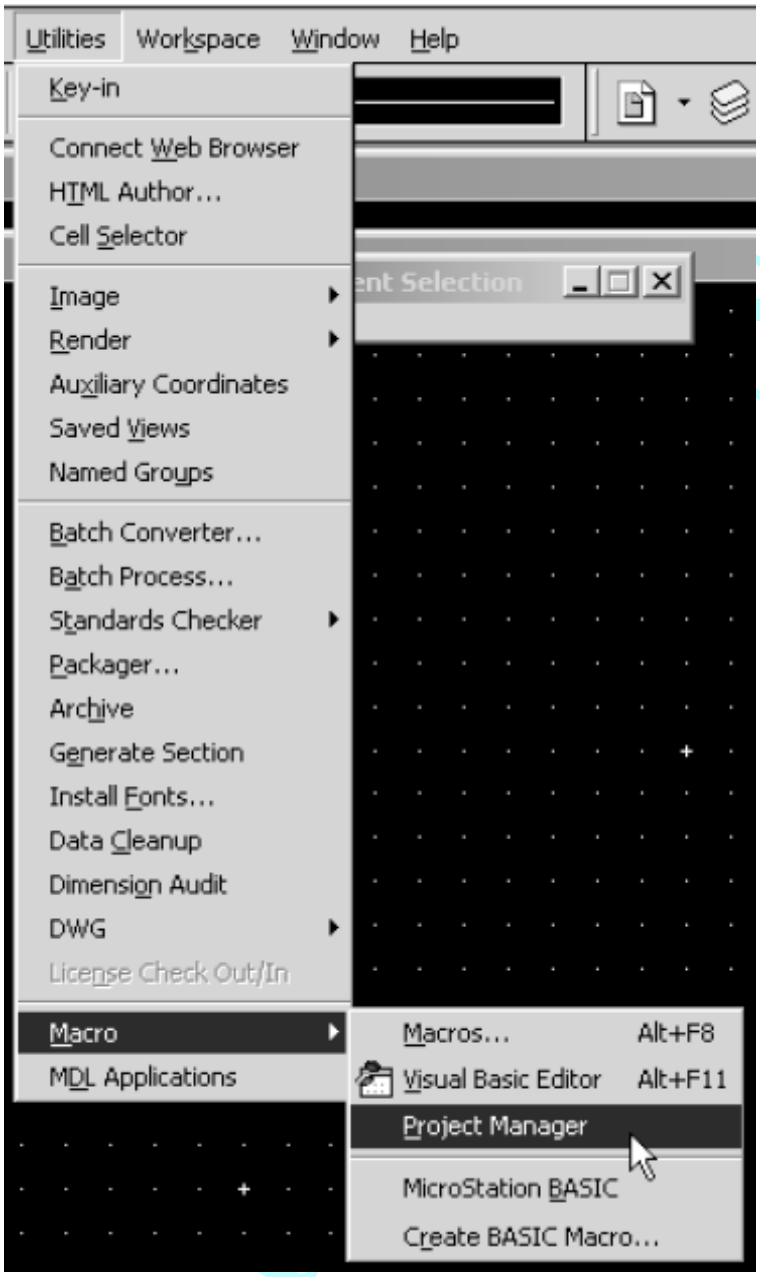
假期? 周末?

应该问你自己的真正的问题是:“我要写的程序能用 VBA 写吗? ”如果对这个问题的回答是“是”，那么大概就应该用 VBA 写。随着您对 VBA 的深入学习，对这个问题的回答将会更多的是“是”。

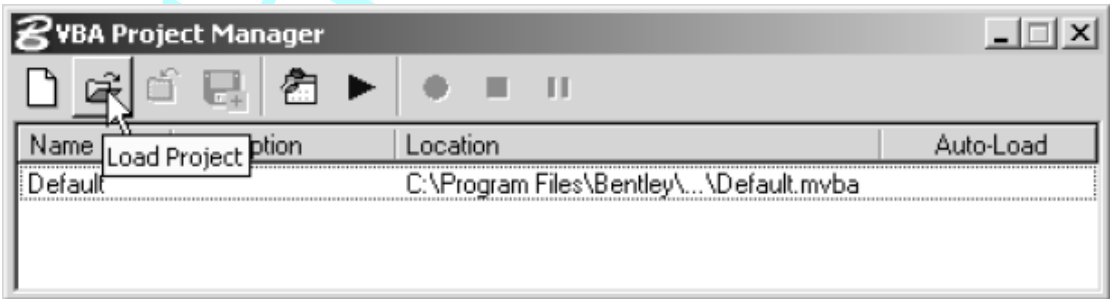
怎样使用 VBA?

MicroStation VBA 程序存储在以.mvba 为扩展名的文件中。要想运行这些 MicroStation VBA 文件中的代码，必须首先装载该文件。在进一步讨论之前，让我们建立一个新的名字为 **Introduction.dgn** 的图形文件。进入该文件后定位到 MicroStation 菜单 **Utilities > Macro > Project Manager(实用程序 > 宏 > 工程管理器)**打开工程管理器。

China BDN

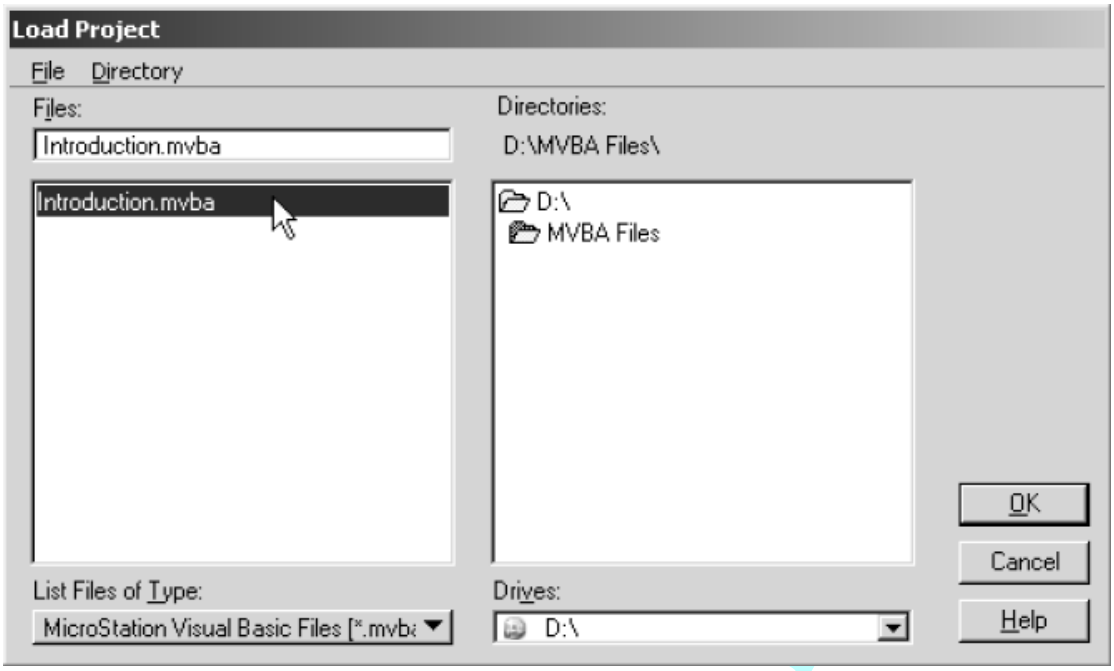


工程管理器用来建立新的 VBA 工程或打开现有的 VBA 工程。

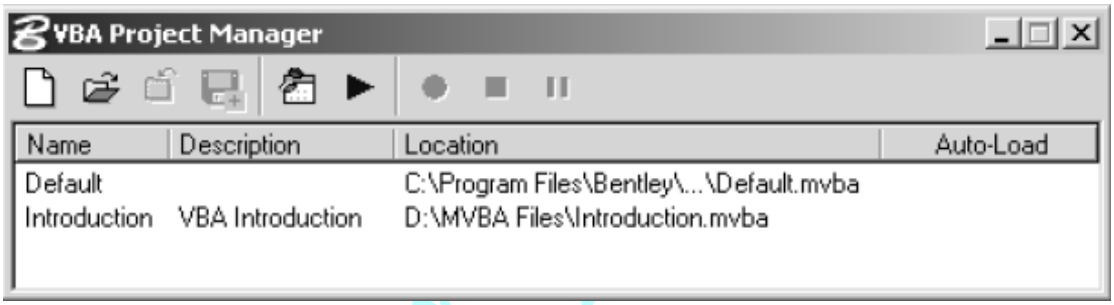


在以上对话框中点击 Load Project(装载工程)按钮，浏览到本书所含 CD 盘的“MVBA Files”文件夹。在该文件夹中有一个名字叫 **Introduction.mvba** 的

文件，选择这个文件并点击 OK 按钮。

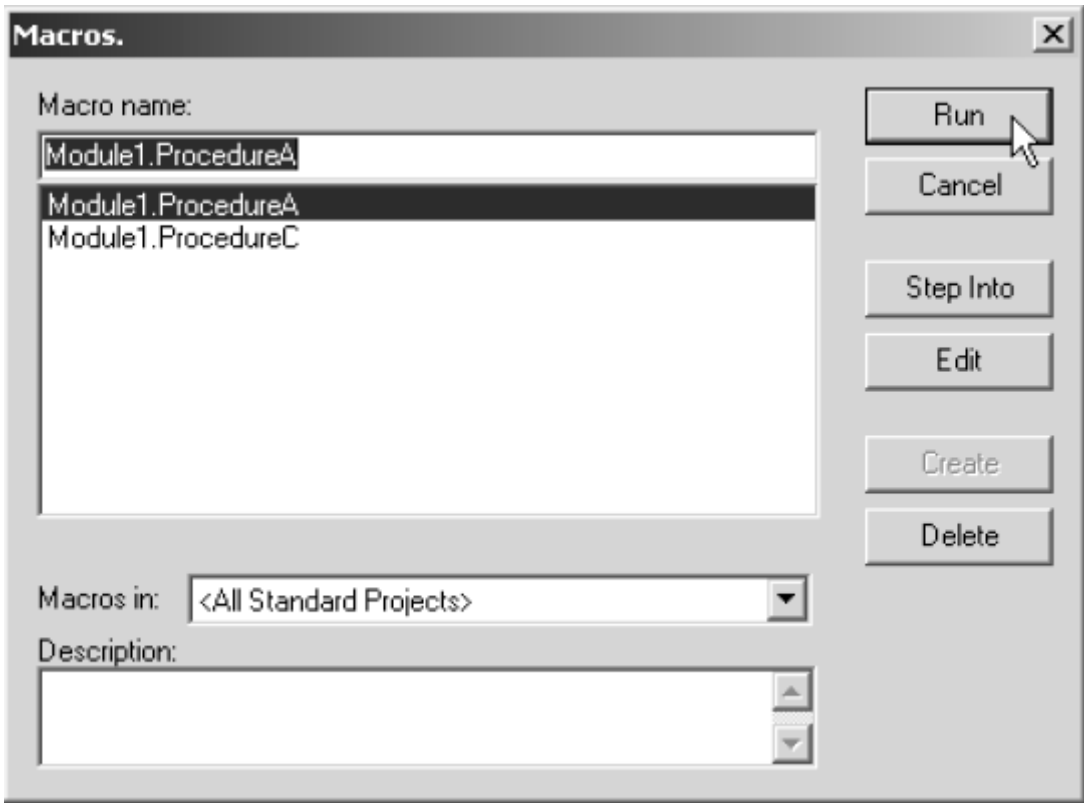


以上动作装载.mvba 文件到 MicroStation 中并在 VBA 工程管理器里显示之。



打开一个 MVBA 文件并不关闭工程管理器。工程管理器一直处于打开状态除非你点击对话框右上角的关闭按钮。

既然已经装载了.mvba 文件就能运行其中的一些代码。如何做到这一点呢？其实有几种方式。先让我们从 VBA 工程管理器中运行代码。如果 VBA 工程管理器(简称为 VBAPM)未打开，按照上面讲过的指令重新打开它。确认装载了 **Introduction.mbva** 文件，在 VBAPM 中选中 VBA 工程 **Introduction.mvba**。找一下 VBAPM 上部的一个像录像机的播放键一样的三角形，它就是运行宏的按钮。点击该按钮后会打开 Macros(宏)对话框，允许选择要运行的宏(过程或函数)。



从宏列表中选择 ProcedureA 并点击 Run(运行)按钮，此时宏对话框关闭，当前模型中绘制出一条对角线。ProcedureA 在当前文件中从(0,0,0)到(10,10,0)画一条线。假如运行宏后没有看到这条线，则使用 Fit View(充满视图)按钮缩放当前视图以显示所有的文件内容。

请记住，运行一个 MVBA 宏的步骤如下：

1. 用 VBAPM(VBA 工程管理器)装载 MVBA 文件；
2. 在工程列表中选择工程；
3. 点击 VBAPM 中的运行宏按钮，或点击 MicroStation 菜单 **Utilities > Macro > Macros(实用程序 > 宏 > 宏)**，或按下键盘上的**<Alt>**键的同时再按**<F8>**键。

这是一个三步处理的过程。当然，如果.mvba 文件已经装载，每次运行宏时就不需要再装载了。可以用以上第 3 步中描述的三种方法中的任何一种来运行某个宏。

您刚用 VBAPM 运行了一个宏。现在再用**<Alt+F8>**快捷键运行一个宏。按下**<Alt>**键然后再按**<F8>**键显示出 Macros(宏)对话框。从列表中选择 ProcedureC 并点击 Run(运行)按钮。ProcedureC 绘制出一个用线组成的正方形，

该正方形从(0,0,0)到(10,0,0)，再到(10,10,0)，再到(0,10,0)，最后返回(0,0,0)。

以上就是我们在这里介绍的运行一个 MicroStation VBA 宏的全部内容。装载并运行它吧。

VBA 是什么样的？

下面就是宏背后的 VBA 代码：

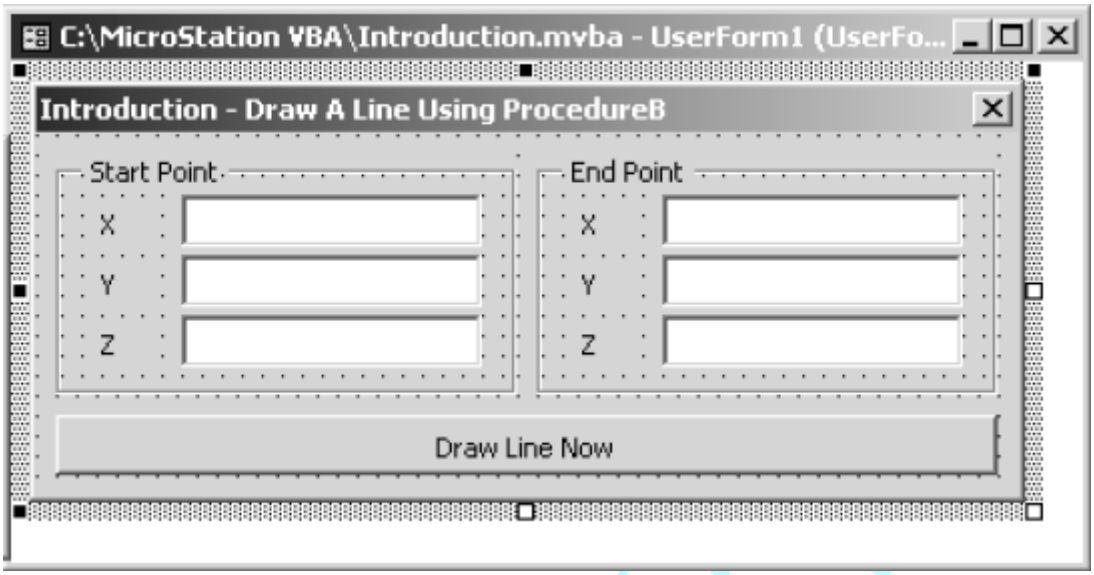
```
Sub ProcedureA()
    '********
    '* 该过程绘制一条从(0, 0, 0)到(10, 10, 0)的线
    '********
    Dim StartPoint As Point3d
    Dim EndPoint As Point3d
    Dim MyLine As LineElement
    StartPoint.X = 0
    StartPoint.Y = 0: StartPoint.Z = 0
    EndPoint.X = 10: EndPoint.Y = 10: EndPoint.Z = 0
    Set MyLine = CreateLineElement2(Nothing, StartPoint, EndPoint)
    ActiveModelReference.AddElement MyLine
End Sub
```

VBA 工程被分成不同的代码块，这些代码块有过程、函数和事件。每个代码块都有一个名字。上面显示的过程被命名为 ProcedureA。代码中的注释以一个单引号开始，单引号后面的任何内容都属于注释部分。用“Dim”语句声明变量并对变量赋值或设置给对象。过程、函数和事件中的代码由上而下运行。在学习 MicroStation VBA 的过程中，我们将一起编写大量的函数和过程。

再回头看 ProcedureA 的代码，它没有图形用户界面——仅仅是代码。书写代码是 VBA 开发的一部分。

VBA 开发的另一部分是图形用户界面(GUI)，例如按钮、文本框和标签。

China BDN



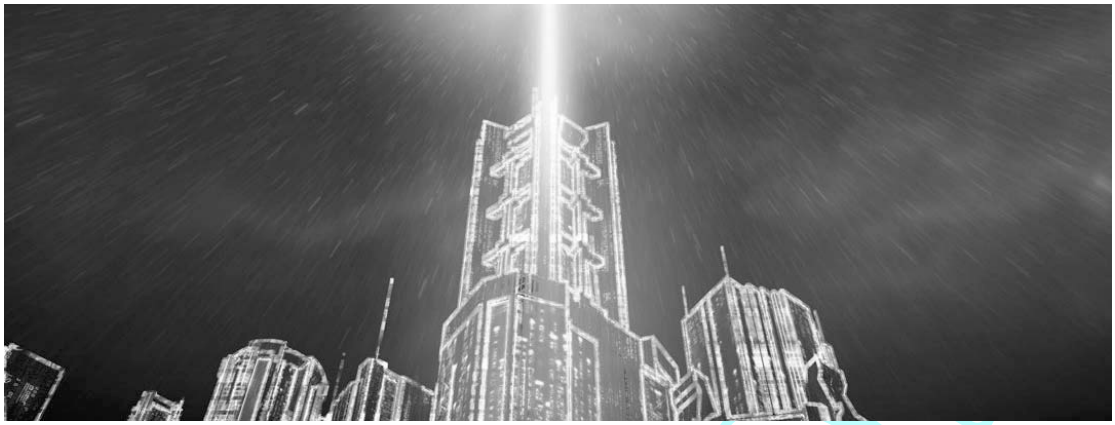
我们编写的应用有些没有 GUI，但我们也会探索 Visual Basic“可视”的一面。(译者注：Visual 即“可视化”的意思)

本章回顾

VBA 工程包含在.mvba 文件中。每个文件中含有代码也能含有图形用户界面。用 VBA 工程管理器来装载和卸载 VBA 工程。代码编写完后，用 MicroStation 菜单 **Utilities > Macro > Macros...(实用程序 > 宏 > 宏...)**或按键盘上的**<Alt+F8>**来运行 VBA 工程及其代码。

学习 VBA 很像在学习一种新的语言，这需要耐心和时间。在我们继续共同学习的时候请牢记这一点。

China B



2. VBA 工程管理器

您已经了解了如何显示出 VBA 工程管理器。请记住，是从 MicroStation 菜单 **Utilities > Macro > Project Manager(实用程序 > 宏 > 工程管理器)**。我们使用工程管理器装载 VBA 工程并运行其中含有的宏。在此，让我们更全面地认识一下工程管理器到底能为我们做什么事情：

- 工程管理器能装载现有的VBA工程；
- 工程管理器允许运行已装载的工程中的过程和函数。
- 用工程管理器能建立新的VBA工程；
- 卸载已装载的VBA工程；
- 保存装载的VBA工程到一个新的位置和/或一个不同的文件名中；
- 从工程管理器进入VB编辑器；
- 从工程管理器中录制宏；
- 自动装载 VBA 工程，每当 MicroStation 启动时工程就被装载。

我们能用 MicroStation 菜单显示出 VBA 工程管理器，也能在按住**<Alt>**键的同时按**<F8>**键来显示出已装载的宏并使其运行。

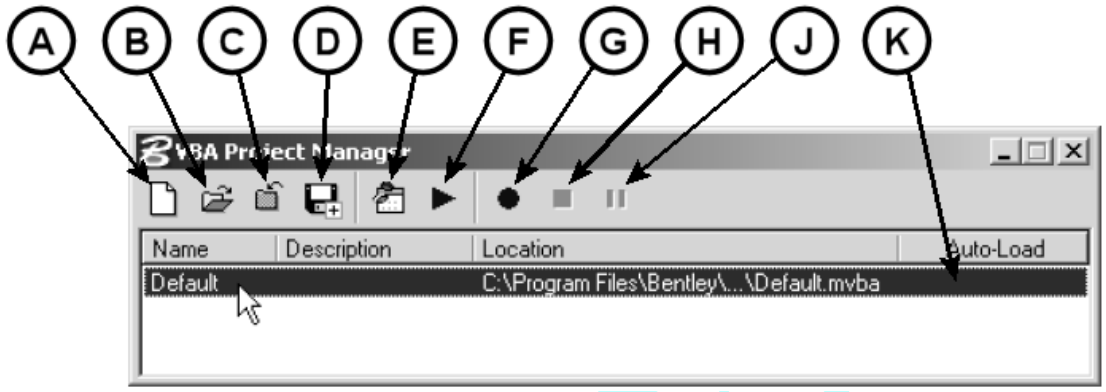
在讨论 VBA 工程管理器时需要细心一些。术语“工程管理器”太通用了，容易与其它的产品或功能混淆。为简短起见，本书从头至尾将会时不时地把 VBA 工程管理器说成 VBAPM。

既然已经从总体上说明了 VBAPM 的功能，下面就更详细地对它加以考察。

Chinaz RD

VBA 工程管理器的功能

下图显示了 VBAPM，每个元素都有一条引出线来标识。本章的剩余部分将经常提及该图。



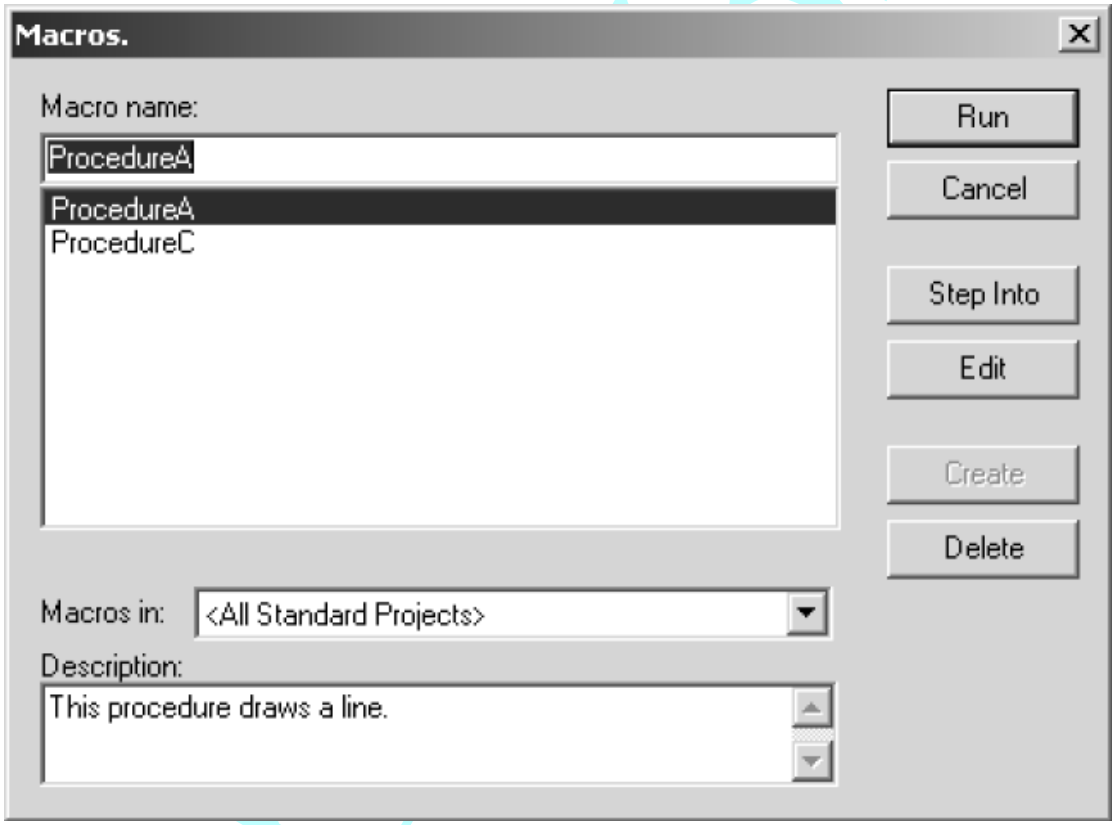
A	建立一个新的 VBA 工程	提示输入新.mvba 文件的位置和名字。
B	打开一现有的 VBA 工程	选择一现有的.MVBA 文件装载到当前会话中。
C	关闭 VBA 工程	该按钮仅当选择了一个现有的工程时才可用。
D	保存工程到	当选择了一个现有的工程时，可以把它保存到一个新的位置和/或一个不同的文件名中。
E	显示 VBA 编辑器	所有新旧工程都是在 VBA 编辑器中编辑的。
F	运行宏	点击该按钮弹出运行宏对话框，可在该对话框中选择并执行某个宏。
G	记录宏	仅当选择了一个现有的工程时才可用。该按钮被激活后，MicroStation 中的所有活动都被记录到选定工程的一个宏中。
H	停止记录宏	仅当激活了记录宏时才可用。当停止记录时，宏被放入选定的 VBA 工程中，命名为 Macro1、Macro2、Macro3 等等。宏记录器自动命名宏。可在 VBA 编辑器中改名这些记录宏。
J	暂停记录宏	暂停记录一个宏并在暂停和恢复间切换。
K	自动装载	设置每当打开 MicroStation 时特定的 VBA 工程被自动装载。在该栏点击后，一个对勾表明该文件被设置为自动装载。



我们刚说明了能够直接从 VBA 工程管理器中做的十件事情。其中之一是“运行宏”，它并不实际运行宏而是显示出宏对话框。

宏对话框

用宏对话框能选择一个宏来运行，但该对话框决不仅限于运行一个宏，它还能“单步运行(Step Into)”它。



Step Into(逐语句)是在调试模式下一次一行地执行宏，这样就能看清代码是如何执行的，变量中保存的是什么值等等。对编程新手或老练的开发者来说这都是 VBA 最好的特性之一。

Edit(编辑)按钮带我们进入 VBA 编辑器窗口，光标停在选定宏的最顶行上。

Delete(删除)按钮从 VBA 工程中删除选择的宏。这是一个非常危险的按钮。毕竟该对话框中没有一个 Undo(恢复)按钮。有吗？所以一定要小心使用 Delete。

Macro in(宏的位置)列出 VBA 工程。如果选择<All Standard Projects>(<所

有标准工程>)，宏列表中会显示所有已装载的 VBA 工程中的所有可执行宏。选择某一个工程将过滤宏列表只显示选定工程中的那些宏。

Description(描述)区允许用户在其中为选定的宏输入一个描述。这是一个好的特性，因为描述能提供比宏名字更多的信息。

例如，不需要给宏起名为 Draw_A_Line_From_0_0_0_to_10_10_0。可以起名为 ProcedureA 并在宏对话框中输入一个描述。

我们除了一个按钮外已经讨论了宏对话框中的每一个按钮，这个按钮就是在上图中被灰色掉的那个 Create(创建)按钮，该按钮建立新的宏。它非常简单，下面就是它的用法：

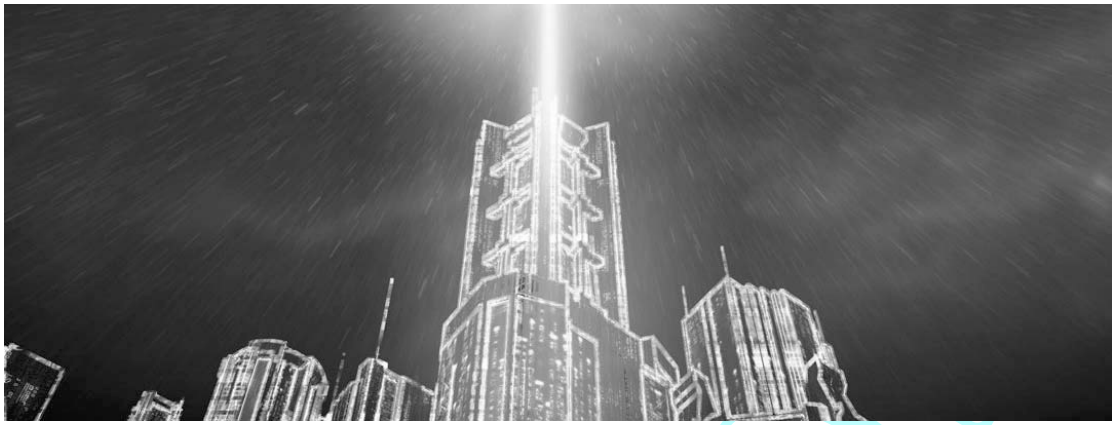
当在宏列表中选择宏时，列表框顶上的文本框中会显示出选择的宏名。如果改变文本框中的文本为不在宏列表中显示的一个宏名(过程名)，那么 Create(创建)按钮将被激活并能用指定的名字建立新宏。所以，如果在文本框中键入 ProcedureA_1 并点击 Create(创建)按钮，一个新的 ProcedureA_1 过程将被建立。当然，刚建立的过程中没有代码，输入代码是我们的工作。现在可以从列表框中选择 ProcedureA_1 并点击 Edit(编辑)按钮，这样就以新过程进入 VBA 编辑器开始编写代码。(译者注：建立新宏时，Macro in 中必须选定一个具体的工程，否则 Create 按钮不会激活。)

本章回顾

VBA 工程管理器在执行一系列任务时是很有用的。这些任务包括：

- 装载和卸载 VBA 工程(MVBA 文件)；
- 把现有的 VBA 工程保存到新的文件和新的位置；
- 建立新的 VBA 工程；
- 记录宏到现有的 VBA 工程中；
- 在 VBAPM 中用“Auto-Load(自动装载)”来自动装载 VBA 工程；
- 从 VBAPM 中进入 VBA 编辑器。

ChinaBDN



3. VBA IDE

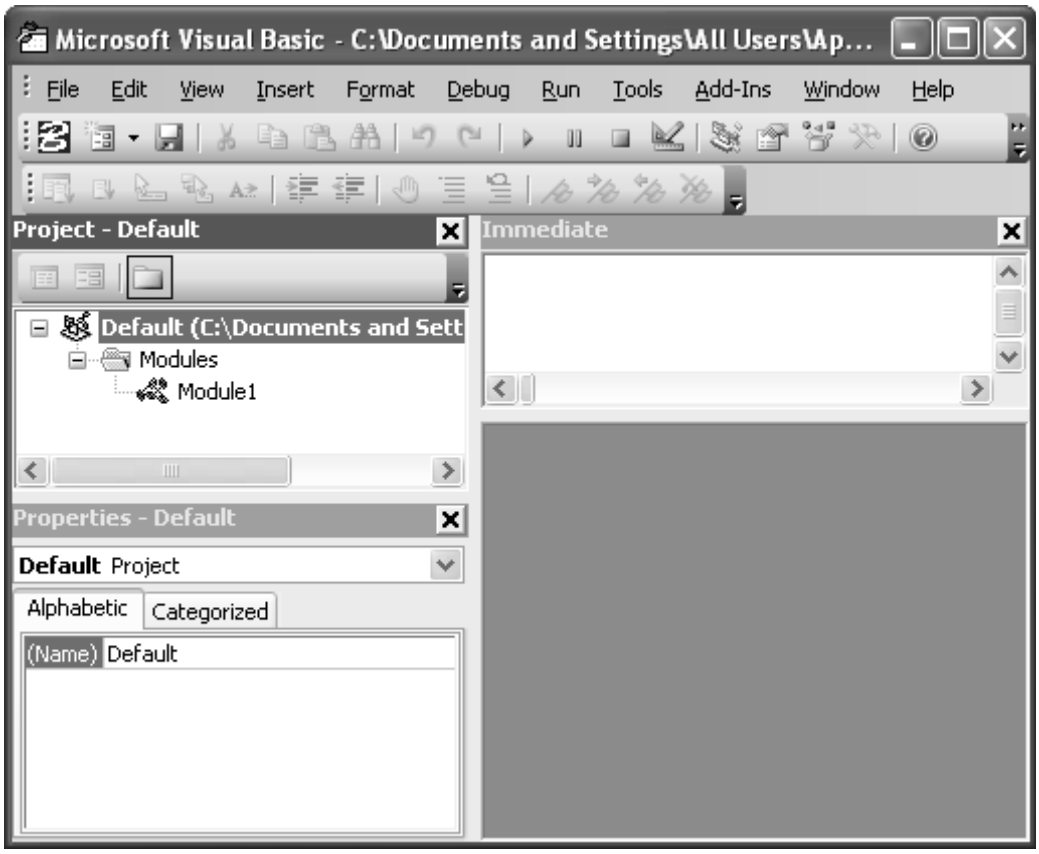
打开 VBA IDE PDQ!!!是的，IDE 是 WYSIWYG。GM?

翻译：赶快打开 Visual Basic for Applications 的集成开发环境吧!!!是的，集成开发环境是“所见即所得”的。您找到窍门了吗？

VBA IDE 是我们进行 VBA 编程的地方。就像大多数 Windows 程序一样，VBA IDE 也由三种元素组成：

- 菜单
- 工具栏
- 窗口

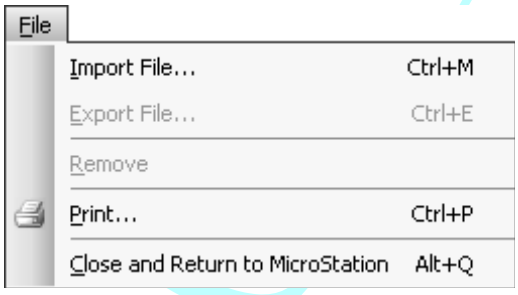
China BDT



菜单

几乎所有的微软视窗应用都利用菜单发出命令。许多菜单项具有快捷键。例如，按住**<Ctrl>**键的同时按**<P>**键与从菜单中选择 **File > Print (文件 > 打印)**完成同样的事情。无论是点鼠标还是用相关的快捷键，了解清菜单项的功能都是有用的。

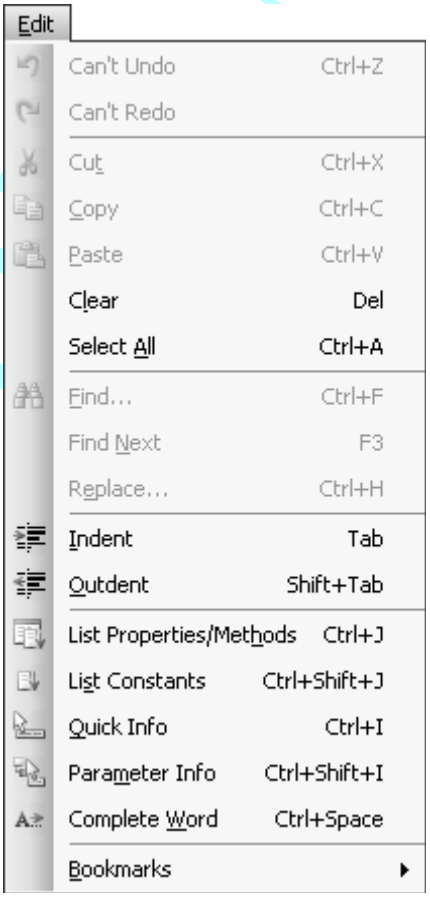
File(文件)菜单



- **Import File(导入文件)**导入现有的窗体(.frm)、模块(.bas)和类(.cls)文件到当前工程中；
- **Exprot File(导出文件)**从当前工程中导出窗体、模块和类到它们自己的.frm、.bas 和.cls 文件中。这些文件被导出后，又可以导入到其它的工程中。
- **Remove(移除)**从当前工程中移除窗体、模块和类。当试图从工程中移除一个元素时，将会被提示是否想先导出(保存)它。

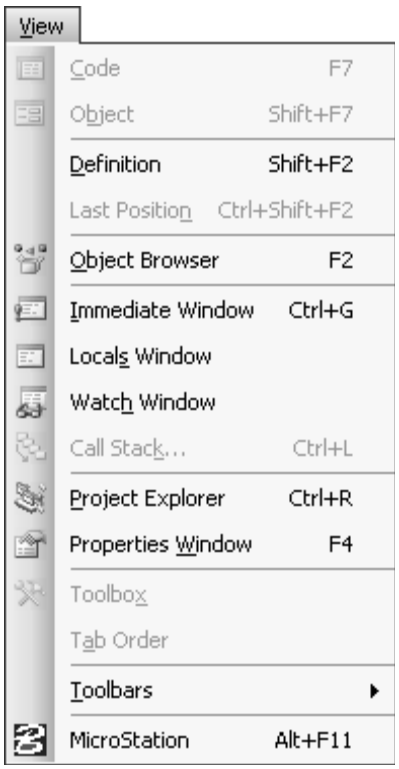
- **Print(打印)**允许打印代码和/或窗体。
- **Close and Return to MicroStation(关闭并返回到 MicroStation)**隐藏 VBA 环境，聚焦当前操作到 MicroStation。

Edit(编辑)菜单



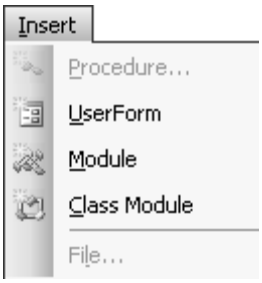
- **Undo(撤销)**和 **Redo(重复)**是标准的 Windows 菜单项；
- **Cut(剪切)**、**Copy(复制)**和 **Paste(粘贴)**执行标准的 Windows 剪切板操作；
- **Clear(清除)**删除选定的文本或对象；
- **Select All(全选)**在代码窗口情况下选择所有的文本，在用户窗体情况下选择所有的控件；
- **Find(查找)**、**Find Next(查找下一个)**和 **Replace(替换)**菜单项执行标准的查找和替换功能；
- **Indent(缩进)**把选择的代码向右缩进一个制表符。**Outdent(凸出)**把选择的代码向左移动一个制表符；
- **List Properties/Method(属性/方法列表)**显示属性/方法列表；
- **List Constants(常数列表)**工作于使用常数的 API 调用。可用的常数表被显示出来；
- 要使用 **Quick Info(快速信息)**，先把光标放到变量、对象或对象的属性或方法上，然后点击 Quick Info 来显示光标所在对象或变量的类型；
- **Parameter Info(参数信息)**显示光标所在的方法的信息；
- **Complete Word(自动完成关键字)**在 VBA 中显示常数和方法的列表以便从中选取；
- **Bookmark(书签)**在代码中设置和移除书签，将光标从一个书签移到另一个书签。书签就是一个让您能快速跳转到某行代码的标记。书签不随.mvba 工程存储。

View(视图)菜单



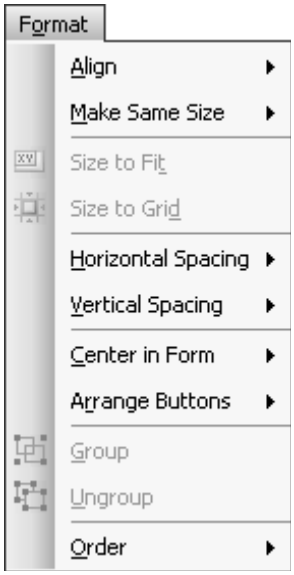
- 在考察一个用户窗体时，点击 **Code(代码)**能跳转到窗体后面的代码；
- **Object(对象)**显示与正考察的代码相关联的窗体；
- 当光标在一个正考察的条目上时点击**Definition(定义)**能快速显示变量的声明处或方法的定义处；
- **Last Position(最后位置)**把光标移到上一个代码行；
- **Object Browser(对象浏览器)**、**Immediate Window(立即窗口)**、**Locals Window(本地窗口)**、**Watch Window(监视窗口)**、**Call Stack(调用堆栈)**、**Project Explorer(工程资源管理器)**、**Properties Window(属性窗口)**和 **Toolbox(工具箱)**显示与它们的名字相同的窗口；
- **Tab Order(Tab 键顺序)**显示控件的 Tab 键顺序属性，这样能看出当用户在窗体中按 Tab 键时控件接收聚焦的顺序；
- **Toolbars(工具栏)**打开或关闭调试、编辑、标准和用户窗体工具栏；
- 点击 **MicroStation** 使 MicroStation 视窗切换到前台。

Insert(插入)菜单



- **Procedure(过程)**显示添加过程对话框以建立新过程。该对话框在为类模块建立新属性时最有用；
- **UserForm(用户窗体)**、**Module(模块)**和 **Class Module(类模块)**插入这些新对象到当前工程中；
- 提高开发速度的一种方法就是重用已经写好的代码。如果把代码放入 ASCII 文本文件中，就能通过点击 **File(文件)**并选择插入文件来向当前工程中插入程序片段。

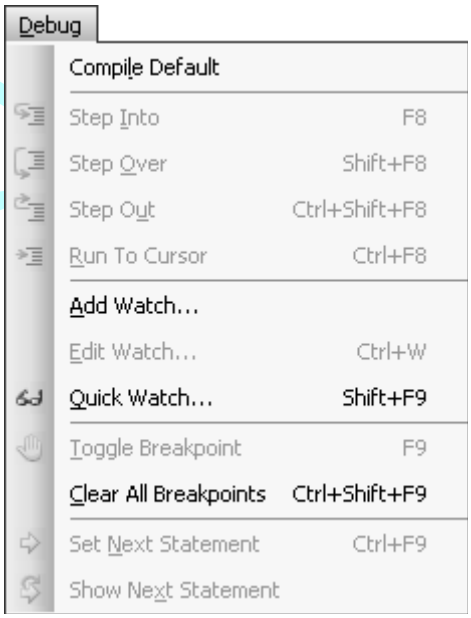
Format(格式)菜单



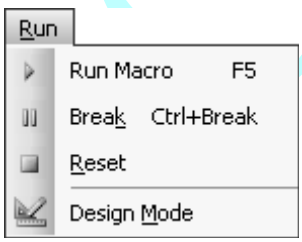
在编辑用户窗体时，用格式菜单执行标准的格式化功能。

Debug(调试)菜单

调试菜单能让我们对代码执行调试操作。将在第 9 章“标准 VBA 调用”中描述这项功能。

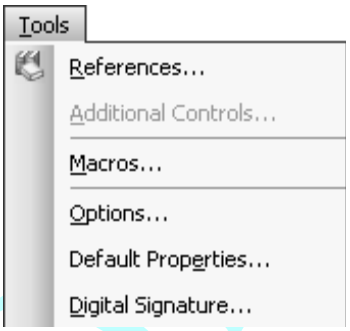


Run(运行)菜单



- **Run Macro(运行子过程/用户窗体)**、**Break(中断)**和 **Reset(重新设置)**代码的执行都是通过运行菜单中的这些菜单项来完成的；
- **Design Mode(设计模式)**是一个标准的 VBA 按钮，在 MicroStation 对 VBA 的实现中没有实际的含义。

Tool(工具)菜单



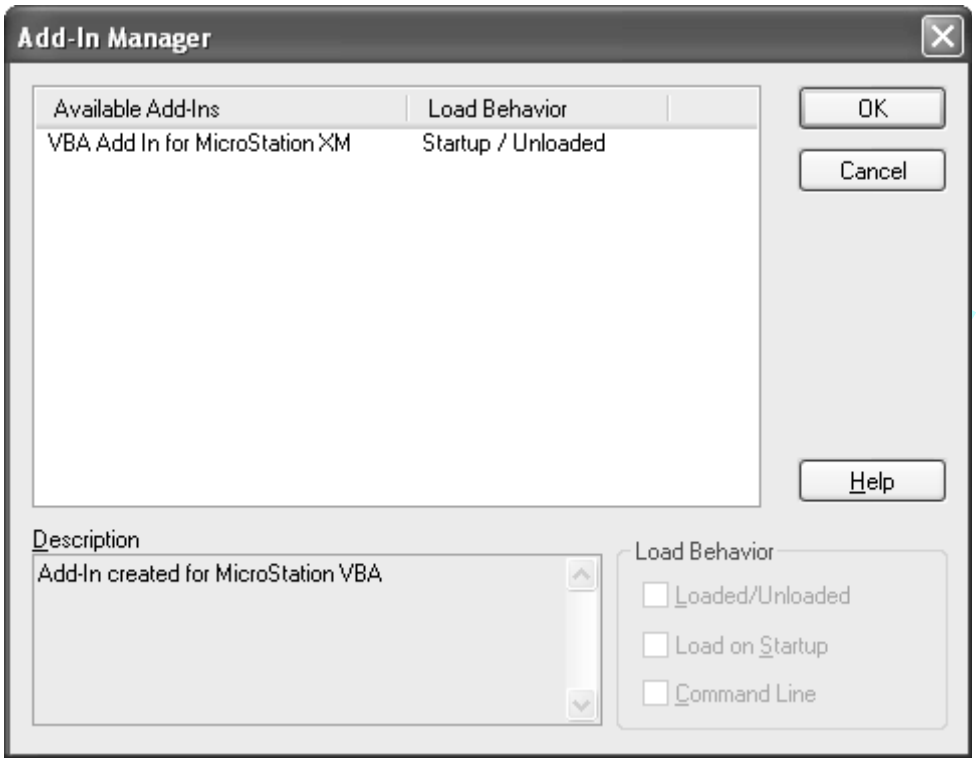
- **References(引用)**能让我们添加一个对现有的 DLL 和类型库的引用。例如，如果想操作 Microsoft Excel，可以添加一个对“Microsoft Excel Object Library”的引用。这样做了以后使得操作 Excel 非常容易；
- 缺省情况下，工具箱中显示有 14 个控件可用于窗体中。通过选择 **Additional Controls(附加控件)**菜单项来添加更多的控件；
- **Macros(宏)**显示宏对话框，可以用它建立、编辑、运行和调试工程中的宏；
- 点击 **Options(选项)**能改变诸如字体大小、制表符宽度以及网格设置等这些预选项；
- **Properties(属性)**菜单项用于显示当前 VBA 工程的属性。在以上显示的图像中，工程名称为 Default(缺省)；
- **Digital Signature(数字签名)**能对 VBA 工程签名。这确保最终用户所运行的代码是由特定的公司创建的。

Add-Ins(外接程序)菜单

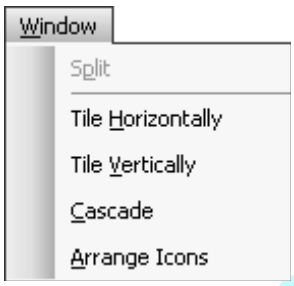


- 第三方开发商能建立 VBA 外接程序(也叫插件)。**Add-In Manager(外接程序管理器)**显示外接程序对话框，在那里能为外接程序设置其属性。

本书中其它地方不会再讨论外接程序，下图是装载了一个外接程序的管理器。外接程序能基于“装载行为”的设置被装载。

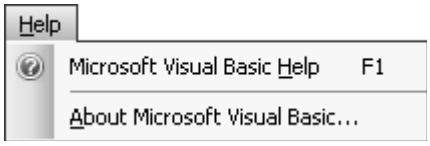


Window(窗口)菜单



这些是几乎每个微软视窗程序都有的标准菜单项。

Help(帮助)菜单



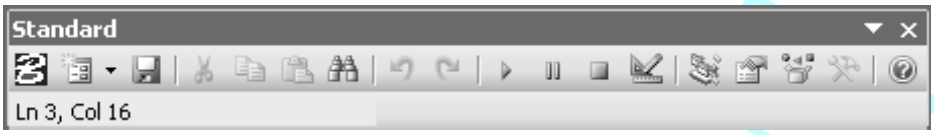
- 下一章将讨论帮助。取得帮助的一种方法是点击 **Microsoft Visual Basic Help** 菜单项；
- **About Microsoft Visual Basic** 显示关于对话框。

工具栏

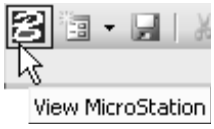
工具栏能非常快速地发出命令，启动一项任务通常只需单击即可完成。与用菜单发出同样的命令至少要双击相比，使用工具栏能立即提升一倍的 CIP(命令发出性能)。一般规则是，点击工具栏图标所发出的所有命令都能从菜单中发出。

熟悉工具栏图标需要一点时间。将光标放在图标上待一会就能看到该图标的功能显示，直到了解了每个图标的功能。

标准工具栏



标准工具栏非常非常重要。为什么呢？因为保存 VBA 工程中对 VBA 的修改的唯一途径就是点击保存按钮。用菜单不能保存修改，必须用标准工具栏中的保存图标。恳请朋友们经常保存您的工程。没有几件事能比如下情况更糟糕的了：你在 VBA 工程中花费了几小时，但由于某些愚蠢的原因，如断电或致命性错误，而丢失了您所有的工作。



请注意光标在图标上停留时所显示出的图标工具提示。

我们能给出每个工具栏上的每个按钮的工具提示，但这会浪费一些时间，因为你自己就可以把光标放在图标上来看工具提示。

编辑工具栏

编辑工具栏显示在编辑菜单中能找到的那些功能。



调试工具栏

调试工具栏显示在调试菜单中能找到的那些功能。



用户窗体工具栏

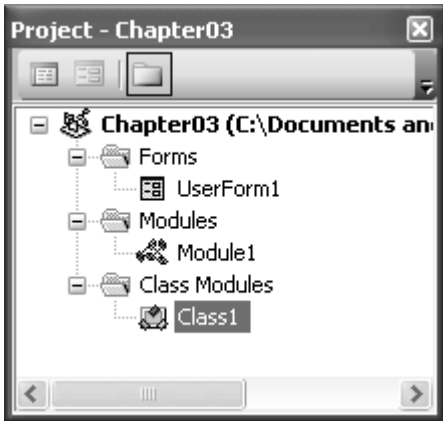


用户窗体工具栏列出在格式菜单中能找到的那些功能。

窗口

用工具栏和菜单项能显示和隐藏 VBA 窗口。让我们看一下今后将会经常用到的 VBA 窗口吧。

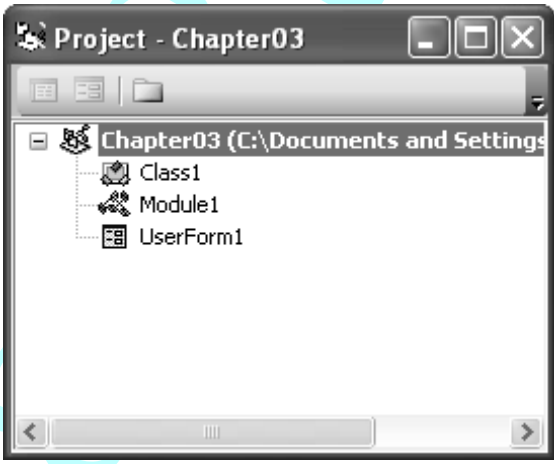
工程资源管理器



工程资源管理器显示装载工程中的顶级对象。

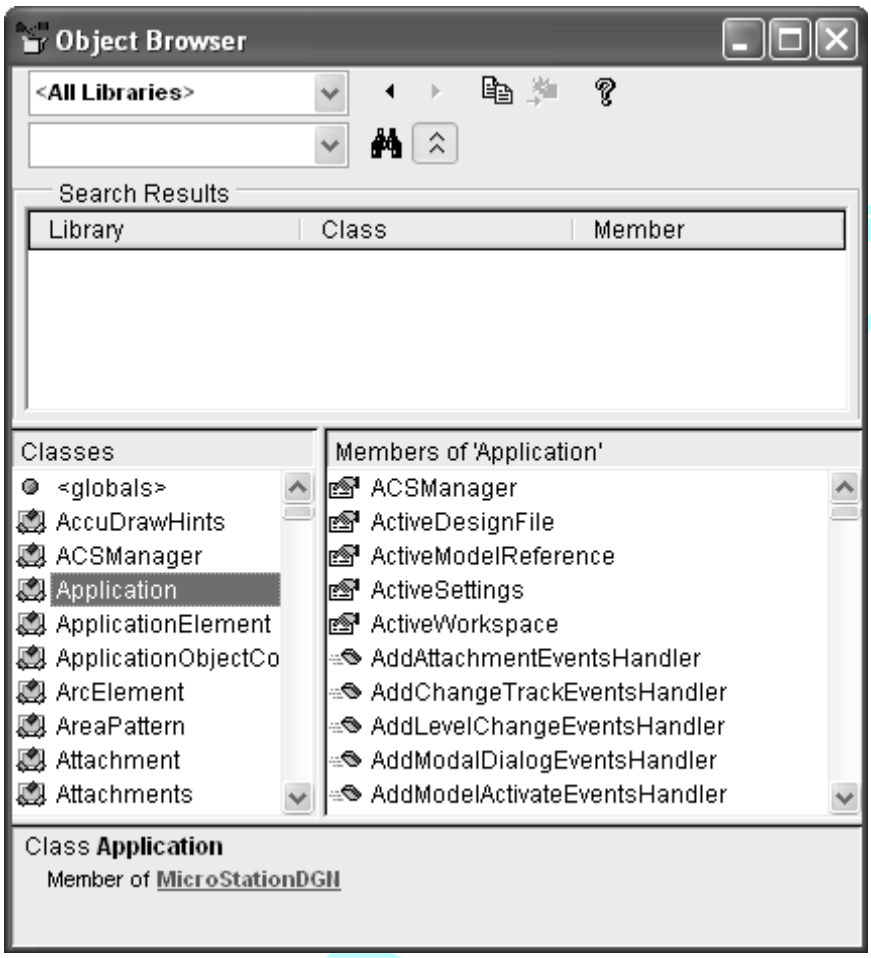
在该工程中有一个叫做 **UserForm1** 的窗体、**Module1** 的模块和 **Class1** 的类模块。左侧所显示的工程资源管理器视图采用文件夹形式按对象的类型来分组。

点击工程资源管理器顶上的文件夹图标能关闭文件夹而以字母顺序显示对象。



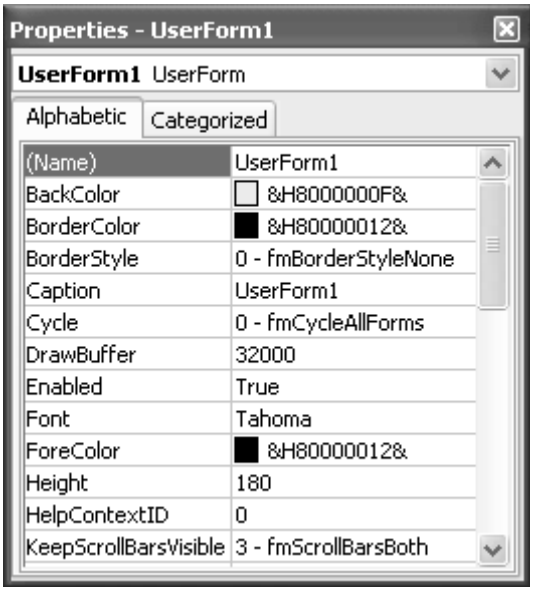
ChinaPDN

对象浏览器



对象浏览器提供了一种探究当前 VBA 工程中所装载的对象(类)的途径。除对象本身外，在“成员”列表下还有看到与每个对象相关的属性、方法和事件列表。窗口底部的灰色区域是所选方法、属性或事件的声明。能显示所有库或选定某个库，也能在装载的库中进行查询。

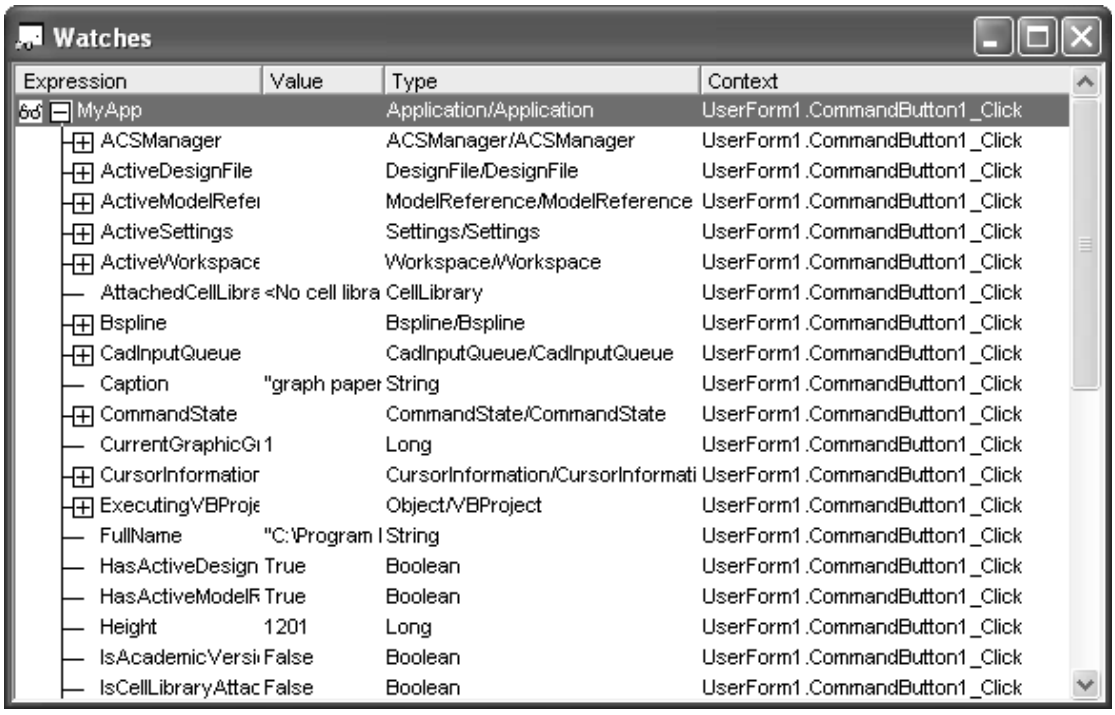
下一章将更详细地讨论对象浏览器。



属性窗口

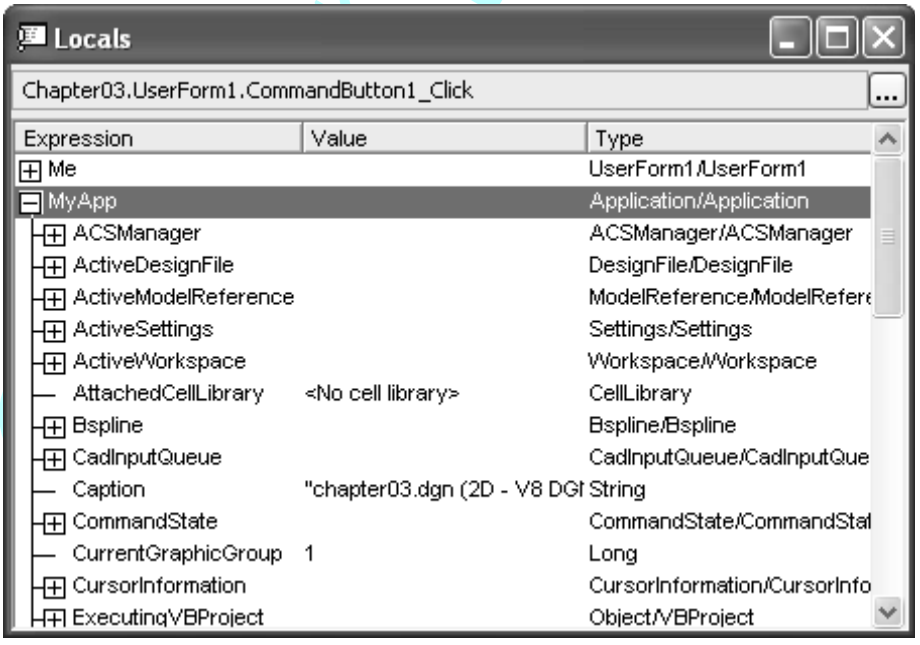
VBA 中的对象都有名称。例如，这个用户窗体的名称是 **UserForm1**。能用属性窗口改变这个窗体的名称、颜色和其它属性。在设计窗体和窗体上的控件时会广泛地使用属性窗口。

监视窗口



监视窗口是 VBA 开发者最喜欢的窗口了。它能让我们监视变量、对象或属性的值。正如上图所看到的，就是在监视一个叫做 **MyApp** 的变量。该变量指向 MicroStation 应用，看一下所能利用的所有属性。

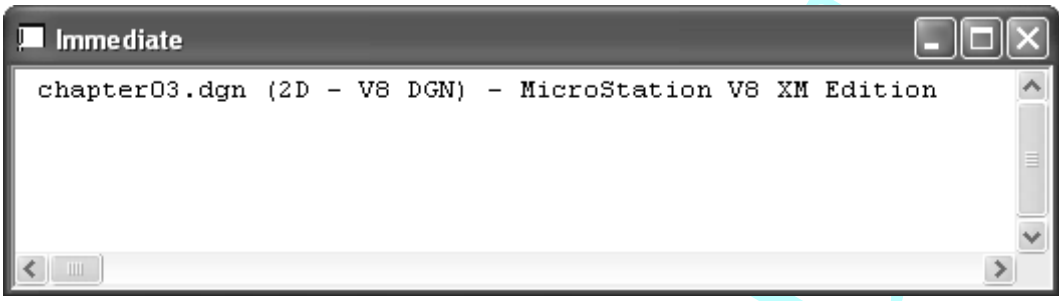
本地窗口



本地窗口看起来非常像监视窗口，然而却有本质的不同。看一看监视窗口中

的条目，你必须添加一个监视条目才行。本地窗口按照每个变量的类型和值自动显示当前过程或函数中声明的变量。

立即窗口



立即窗口能做好几件事情。首先，在代码执行时它能显示文本。当使用如下代码时，

```
Dim MyApp As Application
Set MyApp = Application
Debug.Print MyApp.Caption
```

MicroStation 应用的标题将会输出在立即窗口中。缘于此，立即窗口也被称之为调试窗口。

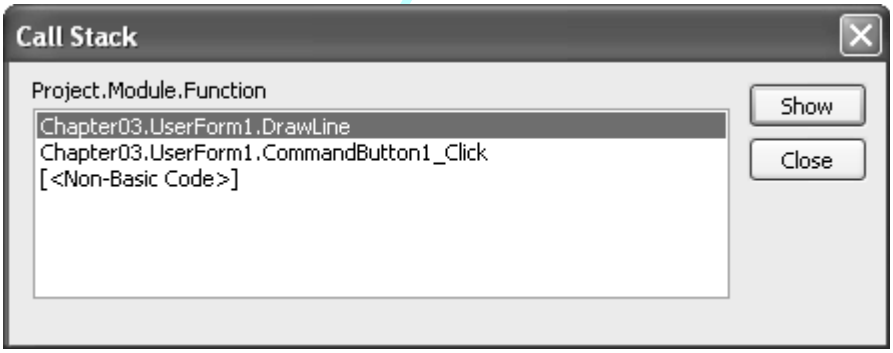
立即窗口能做的另一件事情是允许立即执行代码。例如，在立即窗口中键入



MsgBox “Learning MicroStation VBA”

并按**<Enter>**键，就会显示一个消息框。

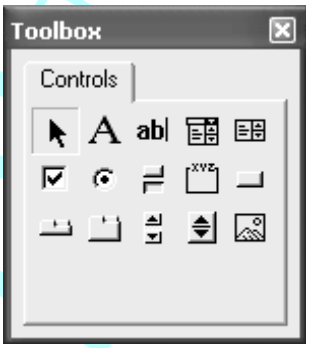
调用堆栈窗口



当我们一行行逐语句执行代码或在代码执行时将其中断，调用堆栈窗口都能显示出我们当前在哪里(最顶行)、从哪里开始的(最底行)以及我们怎样去到那里的(中间的所有行)。当一个过程调用了另一个过程，那另一个过程又调用了一个

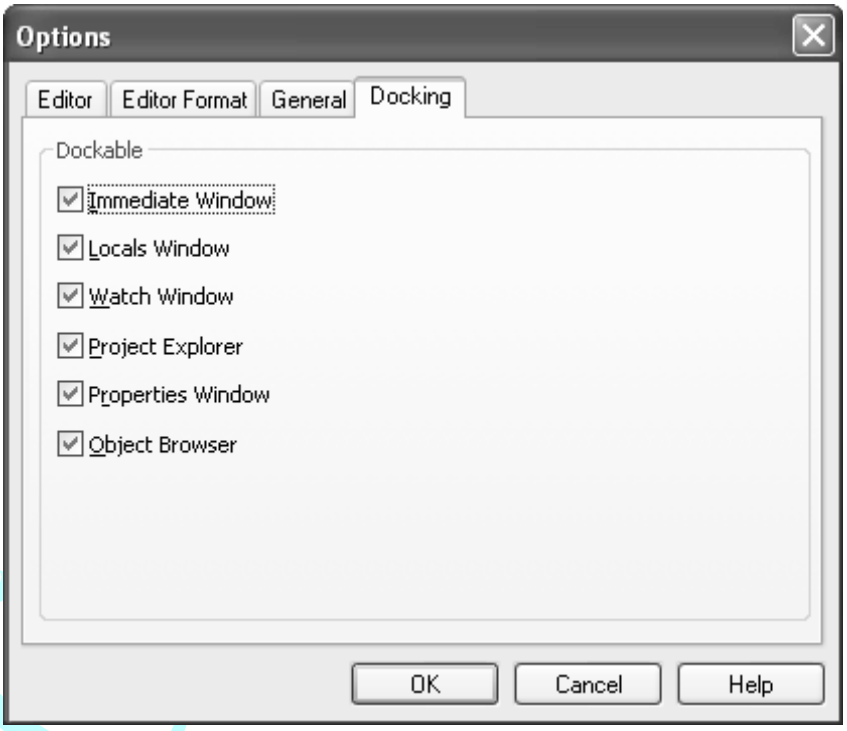
函数，这个函数再次调用了一个过程时，知道如何达到一个过程或函数对调试程序是有帮助的。

工具箱窗口



工具箱窗口显示能放入用户窗体中的标准控件。只有在用户窗体是 VBA 的当前窗口时它才显示出来。假如操作用户窗体时工具箱没有出现，可以点击标准工具栏中的工具箱图标或点击 **View > Toolbox (视图 > 工具箱)**菜单来显示它。

除了工具箱窗口外，我们至今所讨论过的所有窗口都是可贴边的(dockable—译者注：微软 VBA 集成环境中把该词翻译为“可连接的”)。“可贴边的”意思是它们能被吸到 VBA IDE 窗口的底部、顶部、右侧或左侧。这些窗口的可贴边属性在选项对话框的可贴边选项卡中设置。要查看这个选项，请在 VBA 的 **Tools(工具)**菜单下选择 **Options(选项)**。

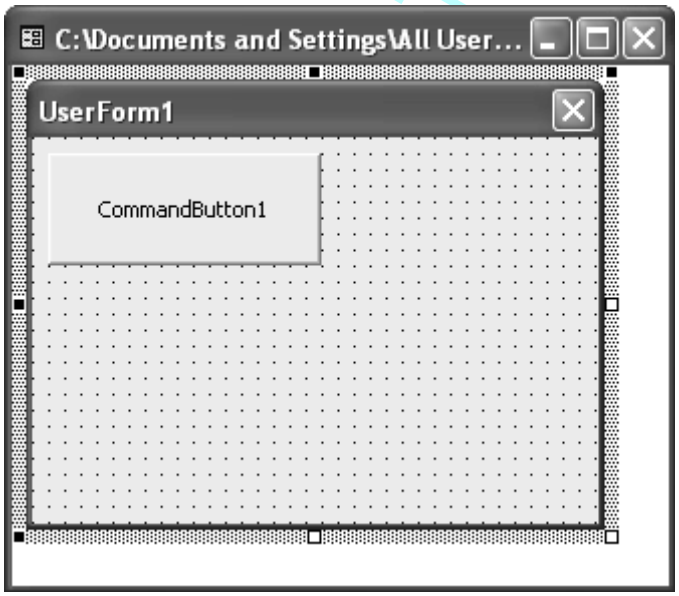


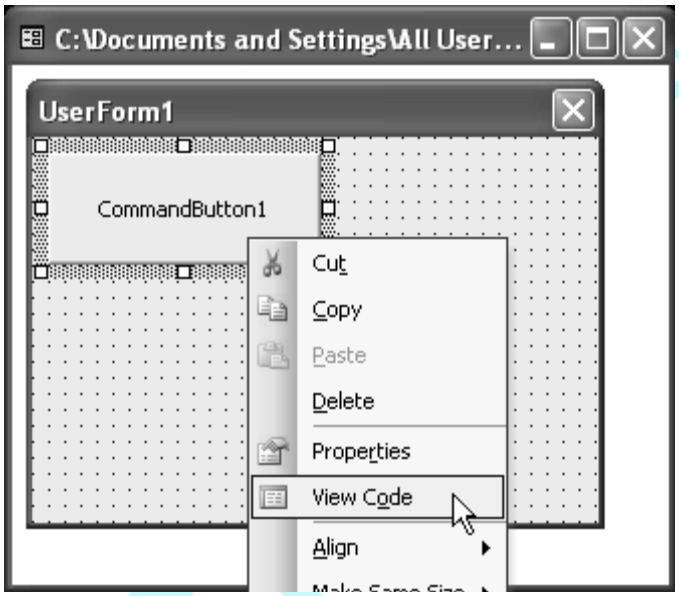
如果贴边放置一个窗口时失败，请查看选项对话框中该窗口的可贴边属性，勾选你想贴边的那个窗口。就像大多应用程序那样，通过把这些窗口拖拽到所希望的某一边，它们就能被贴边。通过把这些窗口从所贴处拖拽出来就能实现窗口的浮动。

其它窗口

在 VBA 中还有几个要讨论的窗口。正如我们所知，VBA 工程由窗体、模块和类组成。这些元素中的每一个都有其自身的窗口。

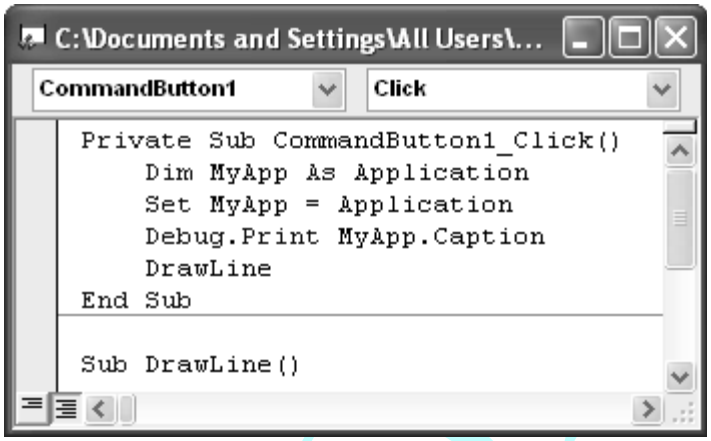
这里是一个窗体。



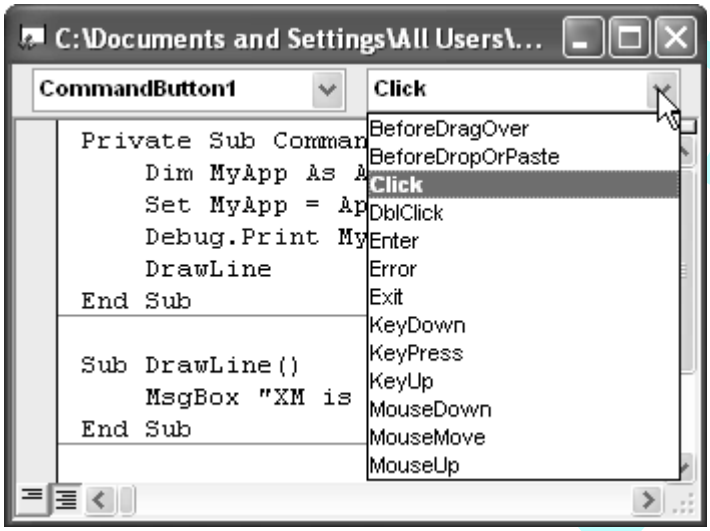


该窗体中仅加入了一个命令按钮，还能再往这个窗体中放入控件。记住，控件是从工具箱中放入窗体的。

运行窗体并点击这个按钮看发生了什么？代码被执行了。双击这个按钮能调出窗体背后的代码窗口。查看代码的另一种方式是右击该按钮并选择 **View Code(查看代码)**。



右侧窗口所示是 **CommandButton1** 的单击事件。**View Code(查看代码)** 能调出右击的那个控件的缺省事件代码。所以，此处看到的是单击事件。还能处理其它的事件吗？怎样看到其它事件并在其中书写代码？



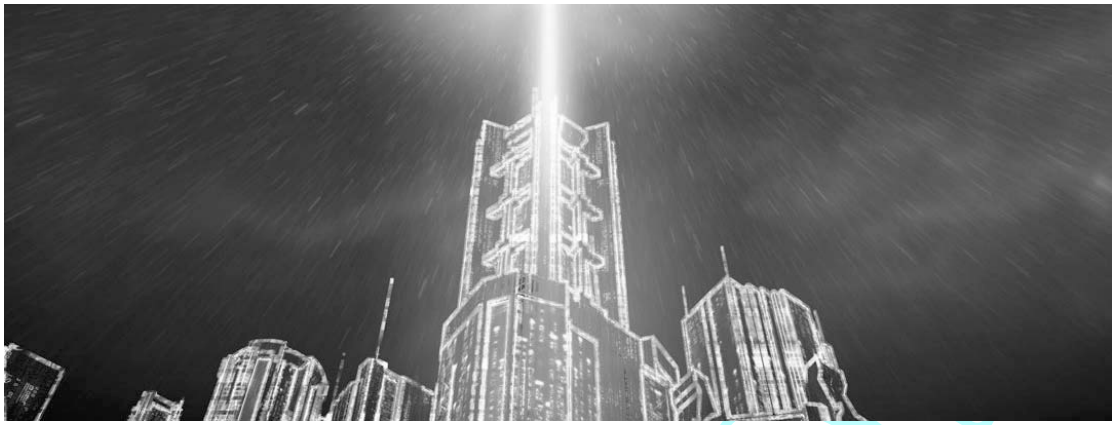
看一下代码窗口的顶部，那里有两个复合框。左边的那个含有当前代码窗口中的控件和对象。右边的那个含有可用的事件。从右侧复合框中选择一个事件将调出所选事件的代码。

在工程资源管理器中双击模块(代码模块)和类模块图标或右击图标后选 **View Code(查看代码)**就能打开模块窗口和类模块窗口。它们看起来非常像上面所示的代码窗口。

本章花了数页篇幅来讨论 VBA IDE。在我们继续学习 MicroStation VBA 的过程中，您将发现 IDE 的其它方面并能得到更加详细的知识。

本章回顾

VBA IDE(Visual Basic for Application 集成开发环境)是我们编写程序的地方，例如书写代码和创建用户界面。随着对这一环境的熟悉，您将能更快速地开发程序。



4. 找到帮助

如何找到帮助是学习一种新编程语言最困难的事情之一。为什么呢？假如您有一个关于某些人的问题，可以与他们交谈直到把问题澄清，这很容易。但当学习 VBA 时，你并不总是知道去问什么。例如，你可能会问：“我如何在窗体中放上一个强制用户输入数值的东西？”如果拿这个问题去问一位 VBA 权威，将能得到简单易懂的回答，但通过帮助文件找答案就比较困难了。对于初学者，即使知道问什么但不懂正确的术语仍然找不到答案。

烦恼吗？是的。灰心吗？的确。我们知道文明的终结吗？不。

本书的定位就是帮助你学习 MicroStation VBA 的。其中充满了代码样例和解释，但它没有包含对每个可能出现的问题的答案。下面的一些内容能提供帮助。

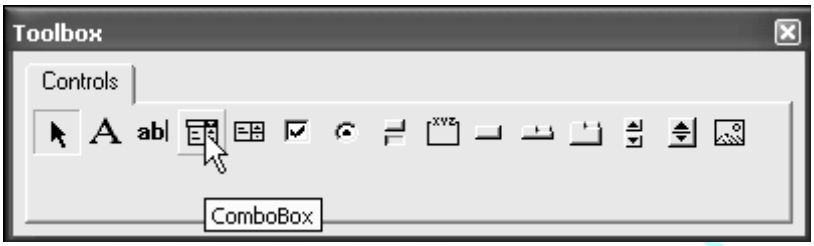
这一章中包括：

- 术语
- 帮助文件
- 网络
- 对象浏览器

术语

Thingy、Dilly Whopper、Whatchamacallit、Gizmo。无论去问人或问计算机，都无法得到这些词的合适答案。那么如何能得到正确的关键字呢？

让我们从“探望”VBA 开始吧。



把光标放在一些对象上将会显示出一个工具提示。在工具箱中有一个复合框，如果问有关复合框的问题将比问有关下拉的问题更容易找到答案。把复合框和下拉这两个词组合在一起会更有意义，但是使用控件的正确名称比用不正确的术语要更接近于问题的答案。

一开始您会觉得 VBA 中使用的术语很陌生，当遇到想记住的词或短语，或者今后想快速找到某个地方时，加个注记或涂上颜色都是好主意。

比如别人让您弄来一根绳子(string)，你可能会缠一条柔软的东西在食指上以提醒你别忘了这件事情。对于我，我刚刚忘记绳子缠在手指上就像我忘记绳子为何要缠在手指上一样。这段话是什么意思？因为不久以后我们的手指就失去了知觉。到底是解开了手指？还是没有手指了？

在 VBA 中字符串(英文中也叫 string)又是什么呢？它是保存文本、数字和其它字符的一种变量。“这是一个测试”就是一个字符串的例子。词 string 在 VBA 中的含义与本杰明·富兰克林电流实验中的含义无法相比较，在书中强调一下字符串类型变量的定义是很有用的。

除了把书中重要的文字涂色强调外，还要拿出一支钢笔或铅笔在书的边上写下心得。

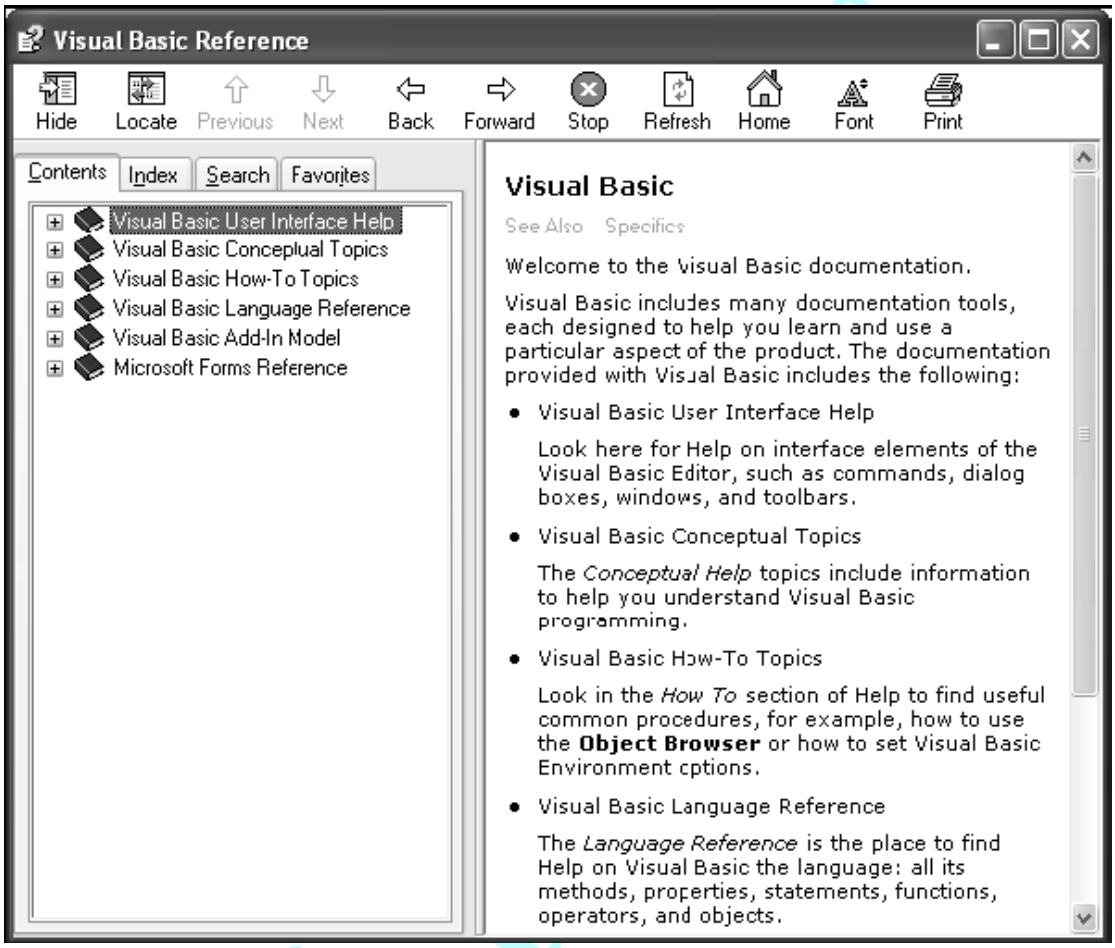
“VBA 程序员的鸡汤”、“流行的 VBA 词语”、“VBA 程序员日历”。这些产品也许不存在但却很有用。为什么呢？因为经常性地接触能增强记忆。

若想拥有黑褐色的皮肤就要经常晒太阳。若想学会烤面包就要经常下厨房。“实践造就完美”、“没有付出就没有回报”、“如果第一次不成功，尝试、再尝试”我们一次又一次听过这些话，这每一句话都在提醒我们经常实践是成功的必要条件。然而为了学习 VBA 术语并不需要每天编写一千行代码，每天写 20 行代码会比只在每周一的早晨写 100 行更有效。

如果频繁地和 VBA 打交道，术语就会变得熟悉，你也将能更好地找到所需要的帮助。

帮助文件

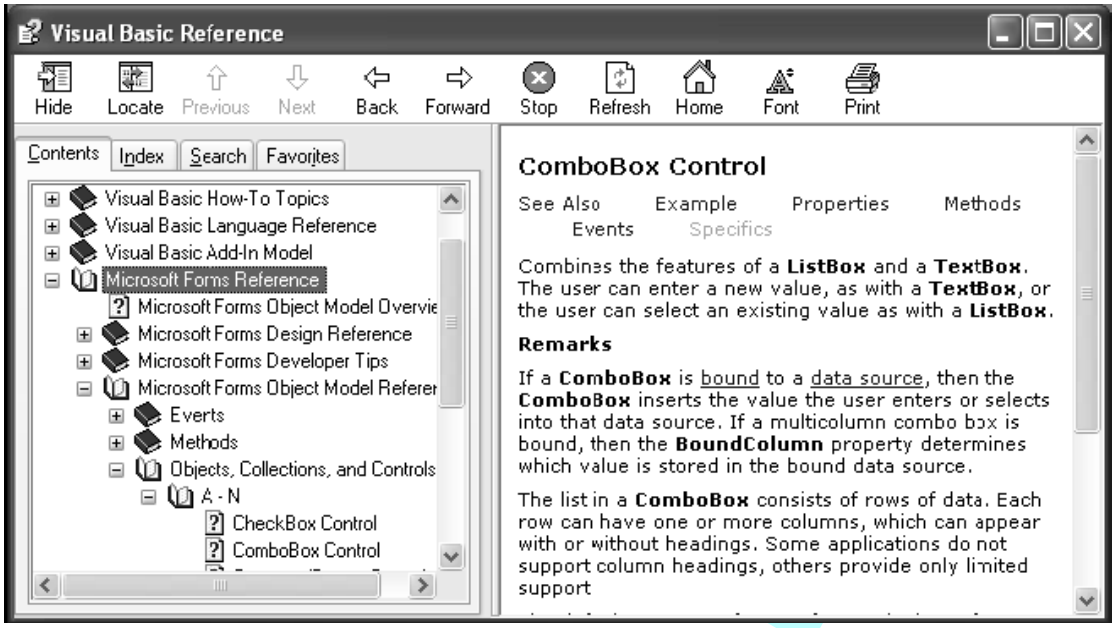
显示 VBA 帮助文件的一种方法就是在 **Help(帮助)**菜单中选 **Microsoft Visual Basic Help**。



该文件中含有大量的信息，也能让我们用书签选项卡组织自己的帮助文件。

你想要一个关于复合框的小型帮助吗？请切换到内容选项卡展开 Microsoft Forms Reference(Microsoft 窗体参考)下的 ComboBox(复合框)。

内容选项卡



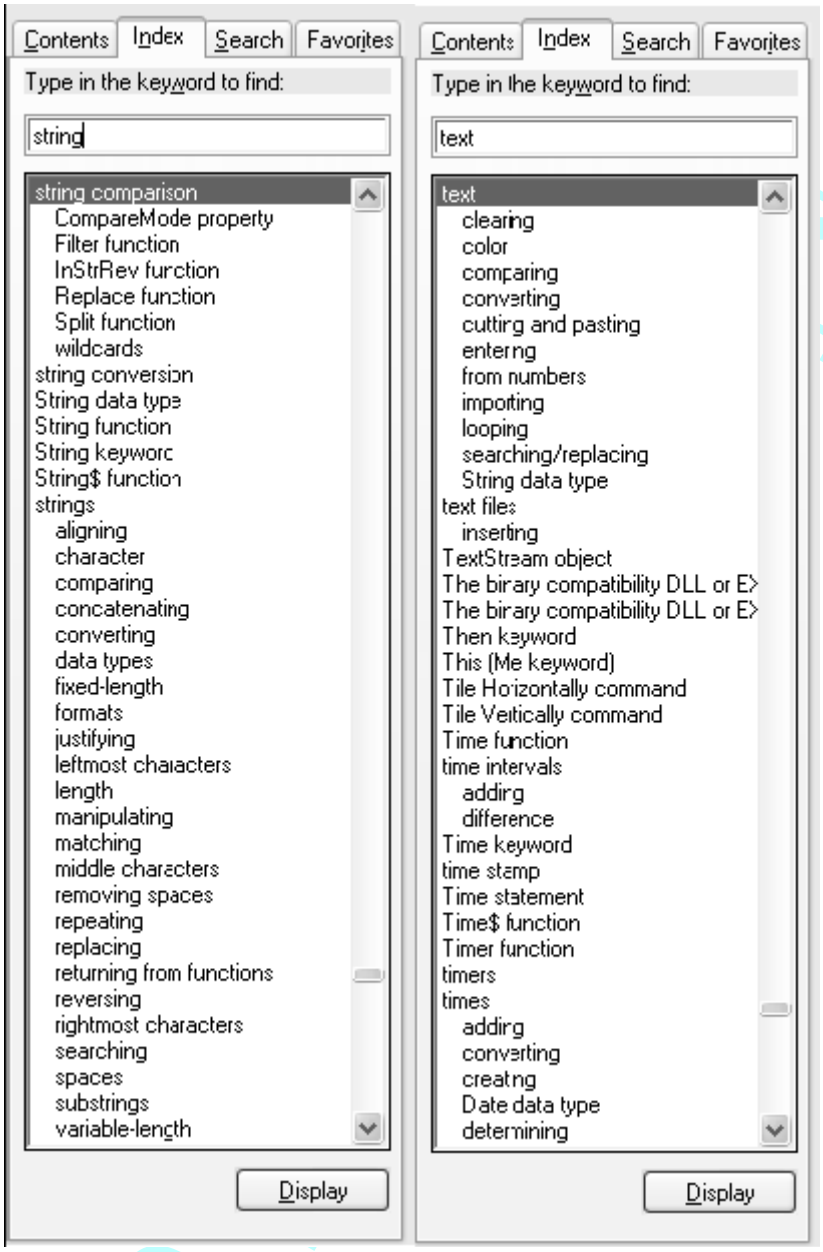
用内容选项卡查找信息需要费些力气，但是知道正确的术语帮助很大。在许多帮助主题的顶部都有“See Also(参阅)”、“Example(例子)”、“Properties(属性)”、“Method(方法)”、“Events(事件)”和“Specifics(特定)”等链接。如果想查找更多的解释，“See Also(参阅)”很有用。如果想查找代码来复制和粘贴，“Example(例子)”就是你所需要的。有关特定的属性、方法和事件的信息请点击对应的链接。

帮助内容中经常含有到其它主题的超链接和对高亮文字更详细解释的弹出框。

通过点击帮助文件顶部的打印图标能打印出帮助主题。

China

索引选项卡

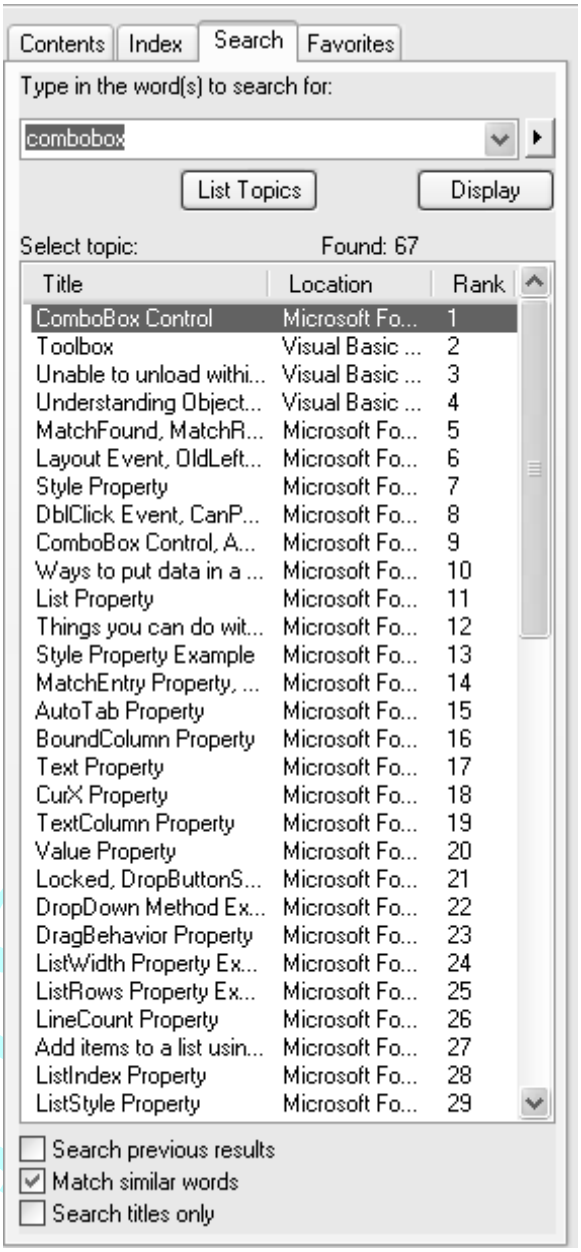


索引选项卡以不同的方式来组织帮助主题。它很像一本书的索引，在这里也需要使用正确术语。如果在关键字文本框中输入“string”，将能得到大量的链接主题。(假如要查找有关字符串变量类型的信息，)输入“text”将得到一些无关的主题，但也有一个有用的“String data type(字符串数据类型)”链接。所以，即使没有准确的术语，接近正确的词也能帮你链接到正确的主题。

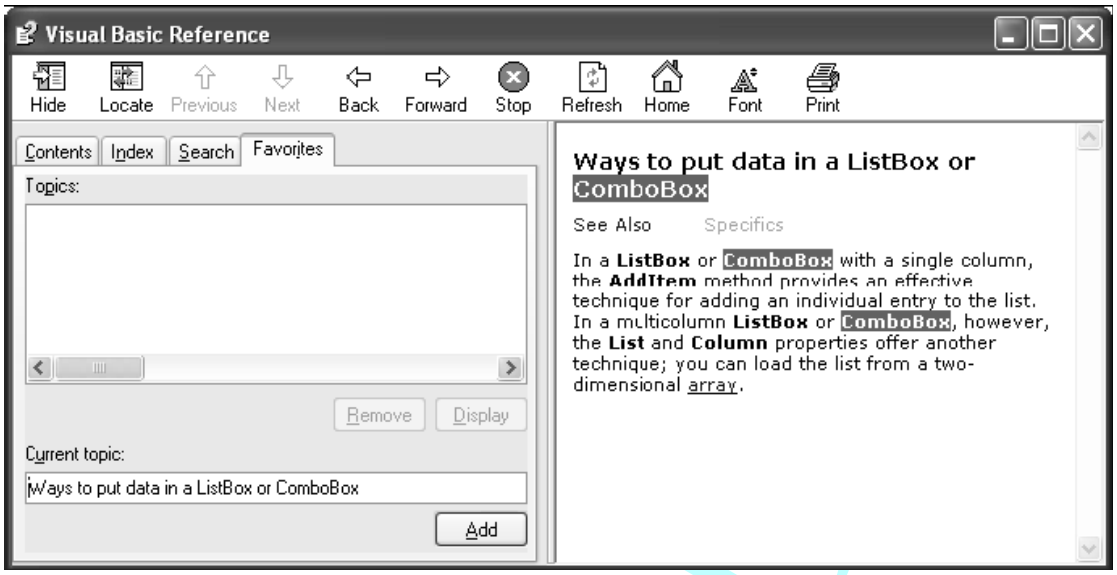
搜索选项卡

在搜索选项卡中输入一个或几个词来搜索帮助主题。它返回所有含有输入词

的帮助主题。例如，输入“combobox”将能返回含有 67 个主题的列表。有些主题是属性，比如“List Property(列出属性)”。另一些主题是使用说明，如：“Ways to put data in a ListBox or ComboBox(在列表框或复合框中放置数据的方法)”。请记住，大多数帮助主题都能链接到相关的主题。所以，以“combobox”开始搜索，通过跳转到其它的帮助主题就能读到远多于 67 个的主题。



书签选项卡



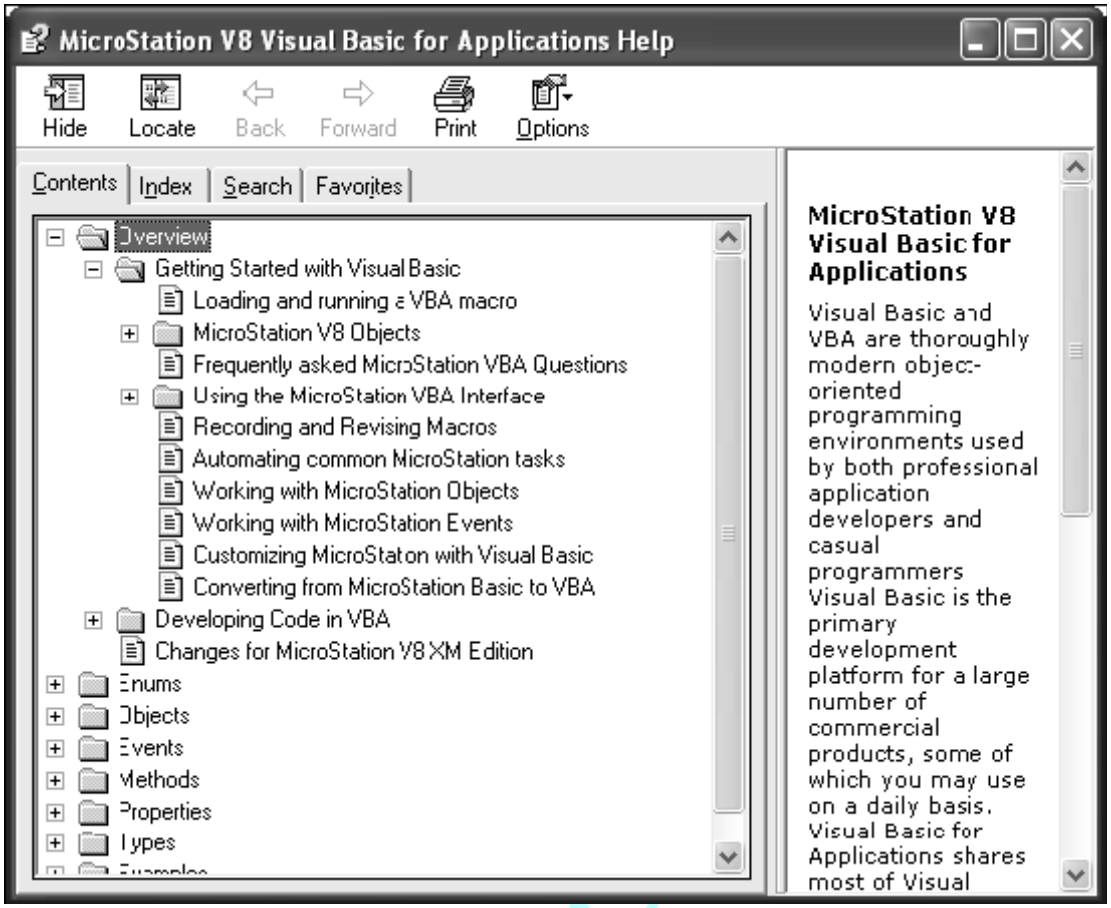
若找到了一个特别有用或很难找到的帮助主题，就把它添加到书签中。Current topic(当前主题)文本框中显示缺省的主题标题。很幸运，能把这个描述改成您想要的任何内容。例如，能把它改成“ComboBox—Adding To the List(复合框—添加到列表)”，然后点击 Add(添加)按钮。

书签是个性化标准 Visual Basic 帮助文件的最佳方式之一。当然了，您可以把帮助文件一页页打印出来，用记号笔对重点内容作标记并装订成册。这是一种分类所学知识并建立附加参考材料的好方法，但是书签能快速找到“标记”主题并能跳转到链接的主题。书签还能让您复制和粘贴内容。

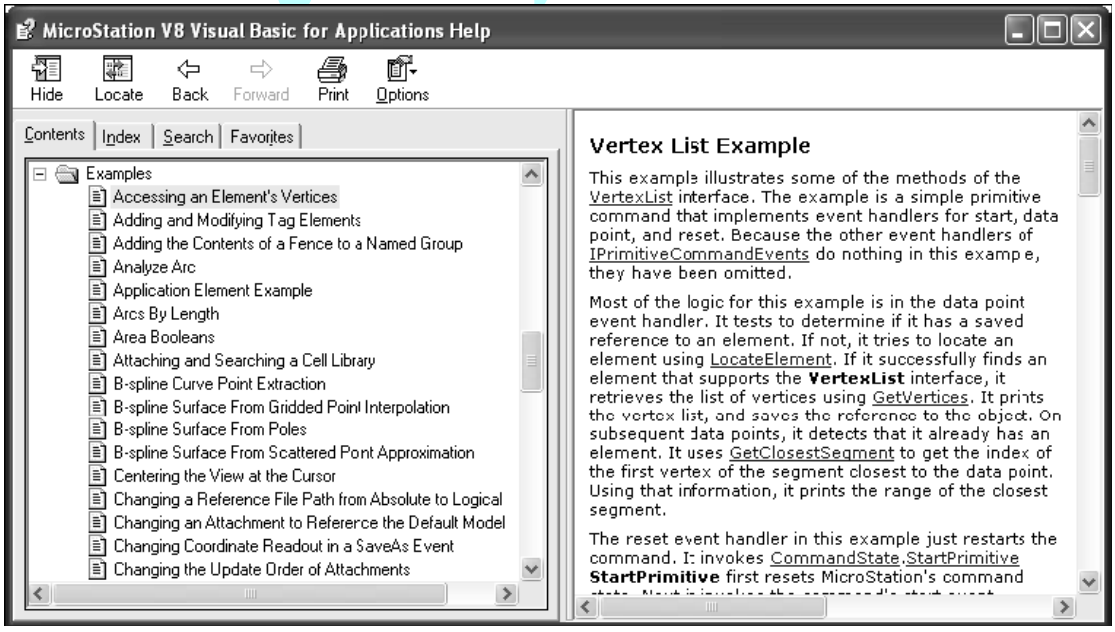
总而言之，VBA 代表 Visual Basic for Applications。本章到目前为止讨论了如何在 VBA 中的“VB”部分找到帮助。那个“A”的帮助又是什么呢？应用？好问题！

学习 VBA 编程语言的各个方面会给我们编程提供良好的背景，但不会对如何与 MicroStation 打交道有非常大的帮助。

MicroStation VBA 帮助文件



MicroStation 有其自己的 VBA 帮助文件，该文件含有 MicroStation 所特有的 VBA 帮助。在您的电脑中搜索文件“microstationvba.chm”将能找到它在什么位置，现在就让我们打开它吧。

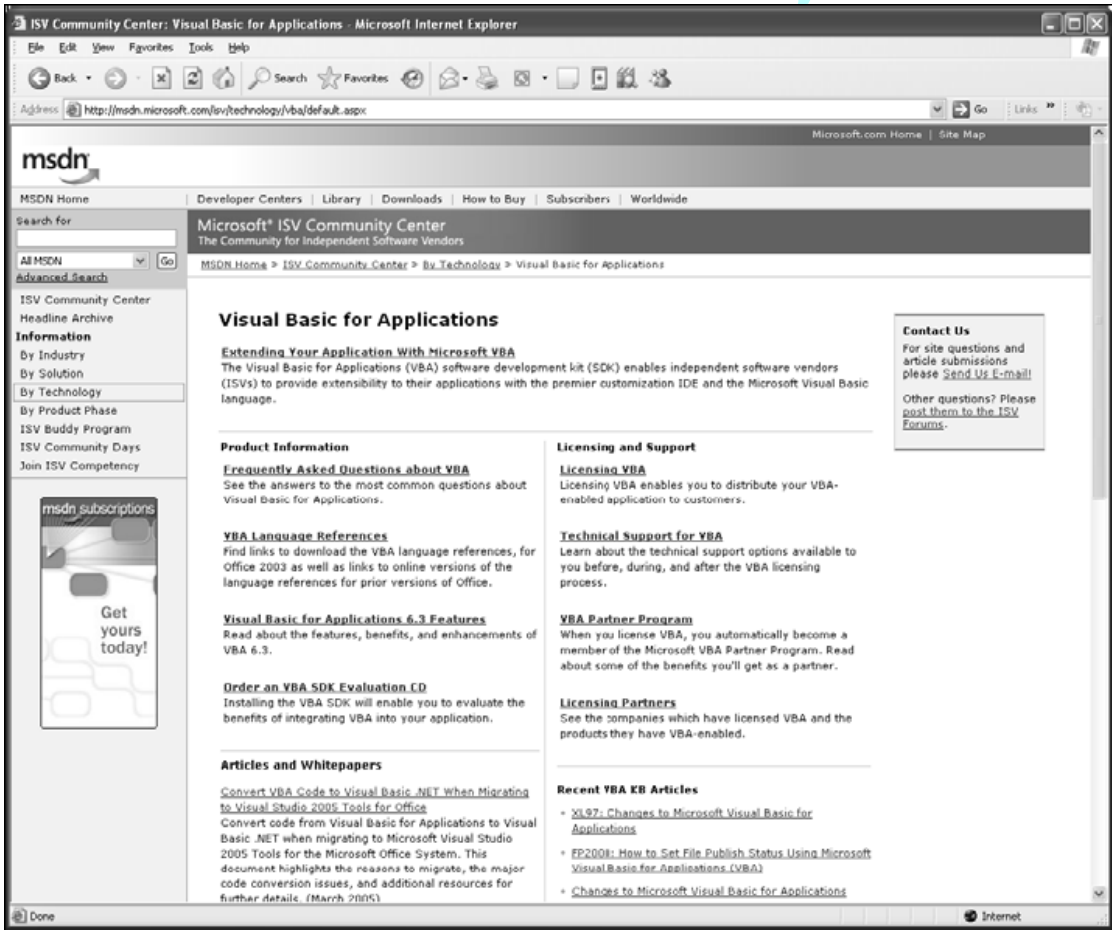


MicroStation VBA 帮助文件中最有用的一节就是 Examples(例子)文件夹，那里含有极好的解释和执行特定任务的代码。

另外一个打开 MicroStation VBA 帮助文件的好方法是在对象浏览器或代码中选择一个 MicroStation 的 API 调用并按键盘上的**<F1>**键。

网络

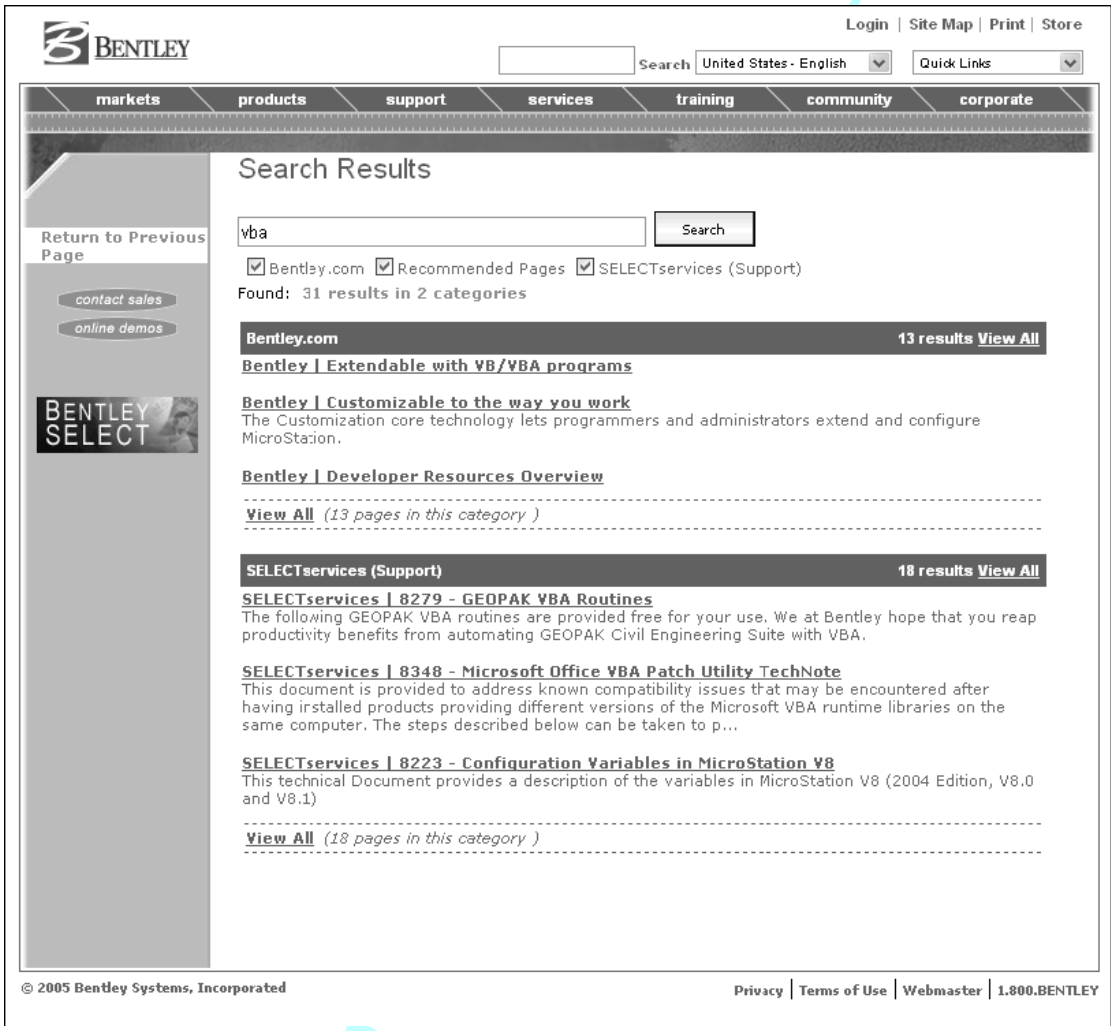
您能上网吗？ http://msdn.microsoft.com/isv/technology/vba/default.aspx 能转到微软的 VBA 站点。如果这个 URL 太难记，还能用 http://msdn.microsoft.com/vba，这个站点有 VBA 白皮书、知识库文章以及其它参考材料，它们构成了 VBA 编程及其相关主题的丰富信息。虽然在这里找不到有关 MicroStation VBA 特有的实现，但将能找到许多其它的在 VBA 环境中完成特定任务的例子。



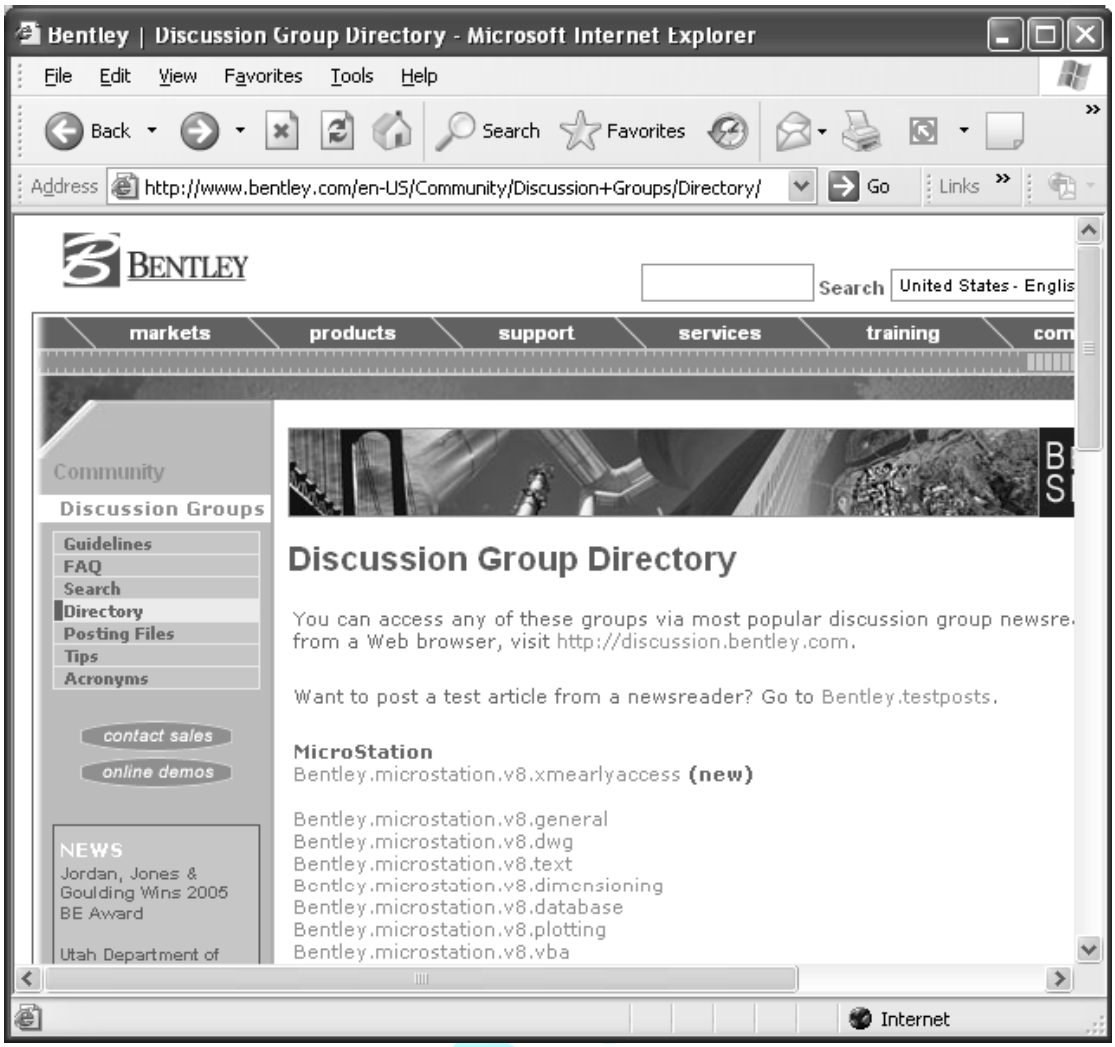
Google、Yahoo 和其它一些搜索引擎能找到其它一些在互联网上的 VBA 知识。请记住，在互联网上有多种编程语言。对“Message Box(消息框)”的搜索将会返回大量的网页，除了有关 VBA 的以外还会有 C#、C++、Java、JavaScript、

Fortran、Pascal 和其它语言的。

到奔特力网站查找“vba”也能得到一些有价值的信息。为什么不直接到 MicroStation VBA 的源头上来呢？



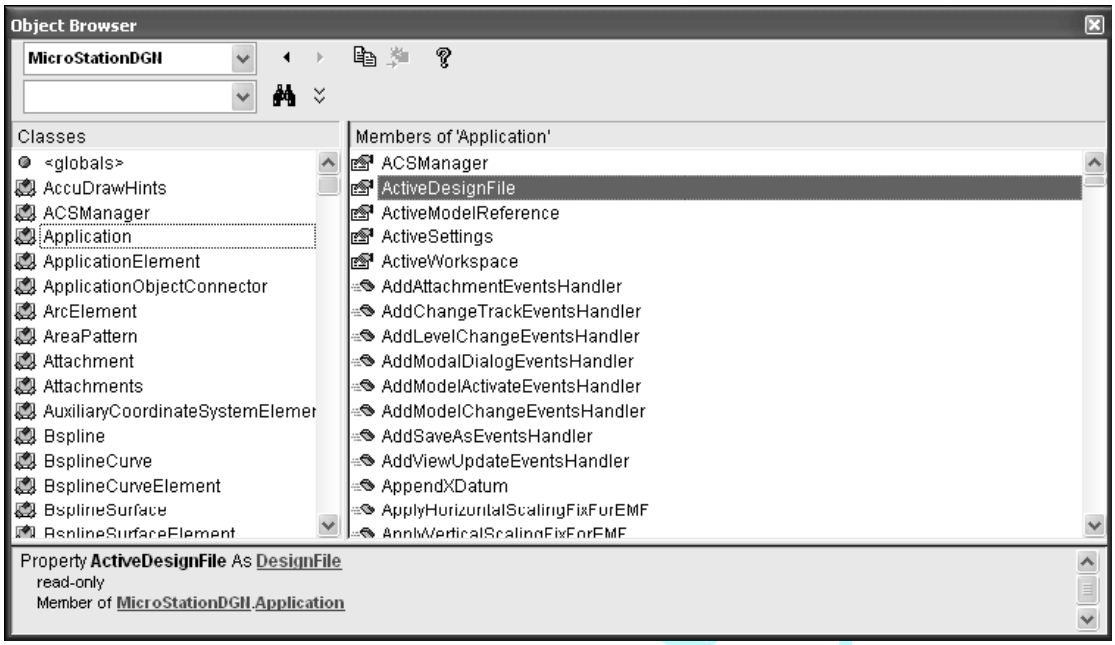
在 Bentley Discussion Groups(奔特力讨论组)中有 **Bentley.microstation.v8.vba**。这是问问题的一个好地方。谁知道答案呢？当你成为 VBA 权威时，你就能在这个讨论组中回答问题来帮助其他的人。



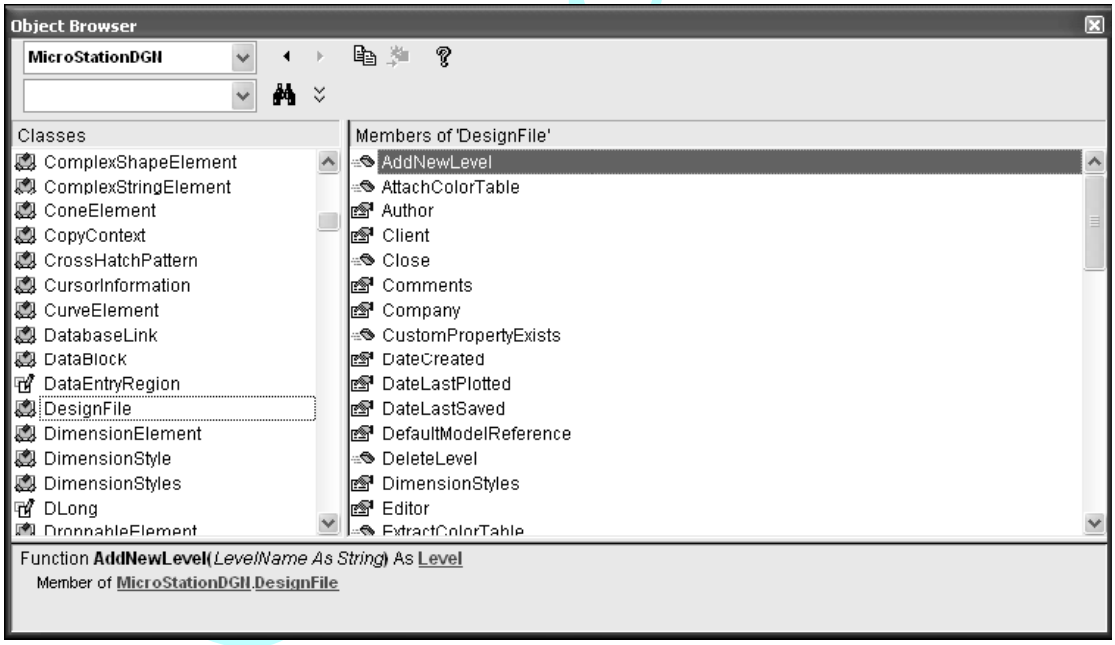
对象浏览器

我们通过 MicroStation 的对象模型与之交流。MicroStation 对象模型的最顶级是 Application 对象。当你想找到对象模型中的某些东西时使用 VBA 的对象浏览器是最有帮助的。如下图所示，可以把浏览器限定到 MicroStationDGN 库，这样所有 MicroStation 的类都在左侧列表框中显示出来。在左侧列表框中选择“Application”，右侧窗口中将显示其属性、方法和事件。

Cnjn



例如，您在左侧列表框中选择了 Application 并想在 MicroStation 中对当前设计文件做点什么，那就可以在右侧列表框中点击 ActiveDesignFile。对象浏览器底部的描述告诉我们 Application 对象的属性 ActiveDesignFile 返回一个 DesignFile 对象。现在在类列表(左侧的列表框)中选择 DesignFile，右侧的成员列表中就能看到 DesignFile 的属性、方法和事件。



在成员列表中选择 AddNewLevel 会在对象浏览器底部显示出使用 AddNewLevel 时需要提供一个 LevelName，也显示出了 AddNewLevel 返回一个 Level 对象。

对象浏览器很有用的一个原因是：它让我们从最通用的级别(Application)开始，通过对象模型从上至下到达要找的对象、属性、方法或事件。使用对象浏览器能逐个逐个地浏览可用的对象和 API。

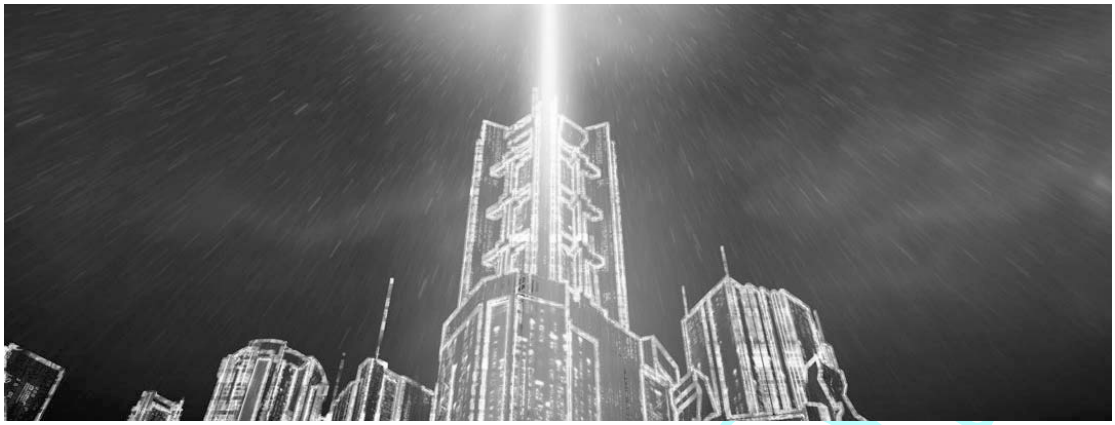
还有其它的方法来找到/获得帮助吗？上一个奔特力认可的 VBA 培训班或参加每年一度的奔特力开发者大会。

本章回顾

找到帮助并不总是一件容易的事情。首先要知道到哪里去找；其次，使用正确的术语能使我们沿着正确的路径去找。学会使用像对象浏览器这样的工具，它们能提供更多的答案。

请记住，即使最简单的主题仍然需要努力去学习和掌握。VBA 对于每个人来说都是一样的。如果您在学习过程中灰心了，成功的机会就会减少。你能学会 MicroStation VBA，你真的能！

China BDN



5. 模块、窗体和类模块

我们已经讨论了 MicroStation VBA 工程的基本内容。用模块、窗体和类创建程序能提高生产力和准确性。我们目前就知道这么多。现在是学习何时该用这些设计元素以及它们如何协同工作的时候了。

这一章中包括：

- 模块
- 窗体
- 类
- 过程和函数

模块

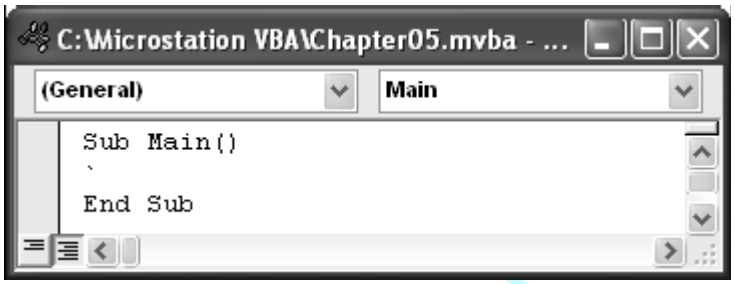
代码模块是每个 VBA 工程的基础。在代码模块中声明的变量能在其本身使用，也能由其它的代码模块、窗体甚至类模块使用。Windows API 函数在模块中声明，所以，在整个工程中都可使用 API 调用(本书后续章节有更多关于 Windows API 函数的内容)。模块内的过程和函数能从 VBA 工程管理器中运行。实际上，代码模块是最基本的，每次建立新的 VBA 工程后都要建立一个初始化代码模块。代码模块中写的过程是运行代码和显示窗体的起始点。

谈得够多的了，让我们写一些代码看一看。

建立一个叫做 Chapter 05 的新 VBA 工程，把它保存到文件夹 C:\MicroStation VBA 中。该新工程建立之后，能看到一个叫做“Module 1”的代码模块被自动建立。把这个模块改名为 modCh05。

继续建立叫做 Main 的新过程。在代码模块中键入

Sub Main()



键入以上代码后按**<Enter>**键，VBA 会为我们输入“End Sub”来完成这个新过程。此时的模块应该像右图那样。

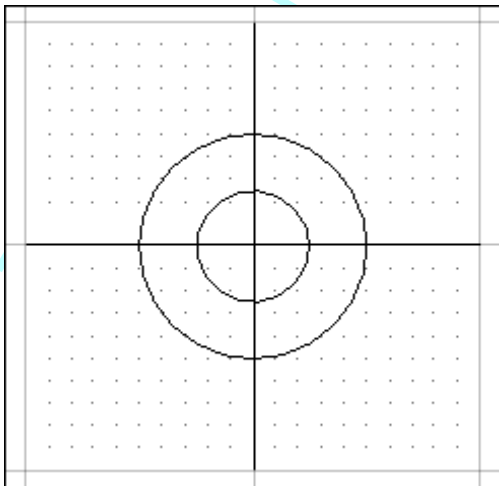
将要做的下一件事情是在新过程“Main”中输入一些代码。

```
Sub Main()
    '声明变量
    Dim MyLine As LineElement
    Dim MyCir As EllipseElement
    Dim CenPt As Point3d
    Dim LineSt As Point3d
    Dim LineEn As Point3d
    Dim RotMatrix As Matrix3d
    '生成水平线
    LineSt.X = -1
    LineEn.X = 1
    Set MyLine = Application.CreateLineElement2(Nothing, LineSt, LineEn)
    Application.ActiveModelReference.AddElement MyLine
    '生成垂直线
    LineSt.X = 0: LineSt.Y = 1
    LineEn.X = 0: LineEn.Y = -1
    Set MyLine = Application.CreateLineElement2(Nothing, LineSt, LineEn)
    Application.ActiveModelReference.AddElement MyLine
    '生成圆
    Set MyCir = Application.CreateEllipseElement2(Nothing, _
    CenPt, 0.25, 0.25, RotMatrix)
    Application.ActiveModelReference.AddElement MyCir
```

```
    Set MyCir = Application.CreateEllipseElement2(Nothing, _
    CenPt, 0.5, 0.5, RotMatrix)
    Application.ActiveModelReference.AddElement MyCir
End Sub
```

以上代码现在看起来可能感觉一头雾水，但对继续学习 VBA 非常有意义。注意插入在代码中的注释。请记住，注释以一个单引号(')开头。

运行这段代码会在 MicroStation 中绘出两条线和两个圆，它们生成了一个如右所示的靶形图。



这段代码非常棒。从现在开始，只要想绘制中心点为(0,0,0)的这样大的靶形图时，就可以用这段代码。无论什么时候想要这个靶形图，只要通过运行宏 Main 就能在几毫秒内绘出。

> 代码写得越死板就越少有机会运行它！

如果想绘制中心在(4,5,0)的靶形图，可以复制、粘贴代码并把该过程改名为 Main2。同时我们还可以建立具有不同坐标的 Main3、 Main4 和 Main5 等等。这样做正确吗？可以，我们能这样做，但还有更好的方法。

让我们稍稍改变一下做事情的方法。建立一个新的在 X、Y、Z 坐标绘制靶形图的过程，以取代在(0,0,0)点绘制靶形图的 Main 过程，其中的 X、Y、Z 坐标由用户来指定。通过为新的过程建立参数来实现这一点。

```
Sub DrawTarget(CenX As Double, CenY As Double, CenZ As Double)
    '声明变量
    Dim MyLine As LineElement
    Dim MyCir As EllipseElement
    Dim CenPt As Point3d
    Dim LineSt As Point3d
    Dim LineEn As Point3d
    Dim RotMatrix As Matrix3d
```

```
    '生成水平线
    LineSt.X = CenX - 1
    LineSt.Y = CenY
    LineSt.Z = CenZ
    LineEn.X = CenX + 1
    LineEn.Y = CenY
    LineEn.Z = CenZ
    Set MyLine = Application.CreateLineElement2(Nothing, LineSt, LineEn)
    Application.ActiveModelReference.AddElement MyLine
    '生成垂直线
    LineSt.X = CenX
    LineSt.Y = CenY + 1
    LineSt.Z = CenZ
    LineEn.X = CenX
    LineEn.Y = CenY - 1
    LineEn.Z = CenZ
    Set MyLine = Application.CreateLineElement2(Nothing, LineSt, LineEn)
    Application.ActiveModelReference.AddElement MyLine
    '生成圆
    CenPt.X = CenX
    CenPt.Y = CenY
    CenPt.Z = CenZ
    Set MyCir = Application.CreateEllipseElement2(Nothing, CenPt, _
    0.25, 0.25, RotMatrix)
    Application.ActiveModelReference.AddElement MyCir
    Set MyCir = Application.CreateEllipseElement2(Nothing, CenPt, _
    0.5, 0.5, RotMatrix)
    Application.ActiveModelReference.AddElement MyCir
End Sub
```

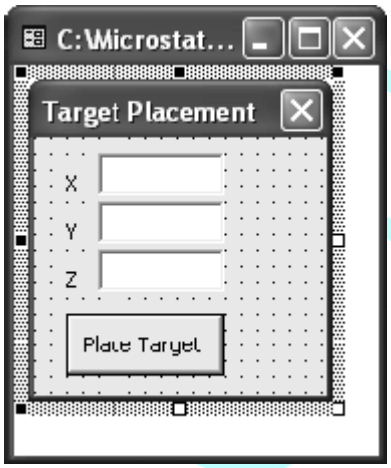
Qilin BDN

在同一代码模块复制并粘贴 Main 中的代码，将新粘贴的过程改名为 DrawTarget。我们的目的是使代码更灵活，可以在指定的任何位置绘制靶形图。新过程 DrawTarget 要求指定三个参数：CenX、CenY 和 CenZ。这三个参数被声明为双精度(非常精确的数字)。让我们看一看如何使用这个带参数的过程。

```
Sub Main()
    '绘制靶形图
    DrawTarget 0, 0, 0
    DrawTarget 3, 0, 0
    DrawTarget -3, 0, 0
    DrawTarget 0, 3, 0
    DrawTarget 0, -3, 0
End Sub
```

过程 Main 现在绘制了五个靶形图。新过程是不是更灵活了？更强大了？绝对是！但坐标仍然是硬编码的(hard-coded，译者注：硬编码就是指在程序中写入了具体的值而不是在运行时由用户输入值)，硬编码只是在 MicroStation 中设置打印页时才用。但是，怎样才能由用户来指定坐标呢？通过下面要介绍的用户界面就能把我们的程序再扩展一些。

窗体



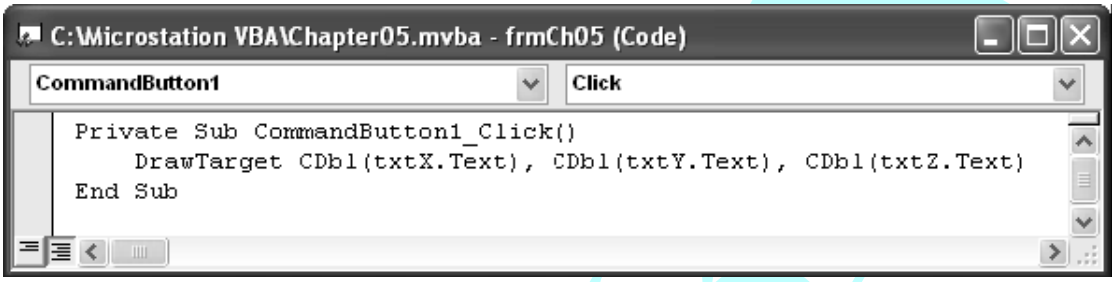
用户窗体给用户提供了一个图形用户界面(GUI)。我们开始插入一个新的用户窗体。

该窗体含有三个标签、三个文本框和一个命令按钮。除文本框外，保持其它窗体元素的缺省名称。改文本框名为 txtX、txtY 和 txtZ，这是在 VBA 中的属性窗口中完成的。

当选中一个控件时，能在属性窗口中改变控件的属性。就像 VBA 中其它的窗口一样，假如属性窗口没有显示，通过操作 VBA 菜单 **View > Properties Window (视图 > 属性窗口)**来打开它。在改变了文本框的名称后，再改变标签和命令按钮的标题(caption)属性使它们与上图所示一致。窗体上控件的属性都修改完后，把用户窗

体的名称改为“frmCh05”。

我们将要在文本框中输入一些坐标，当用户点击“Place Target(放置靶形图)”按钮时，以用户输入的坐标为参数调用刚刚建立的 DrawTarget 过程。这需要在命令按钮的 Click(单击) 事件中写代码。在工程的设计状态双击按钮进入命令按钮的 Click 事件。



还能通过右击按钮在弹出菜单中选择 View Code(查看代码)进入这里。

仅需输入一行代码就能同时使用 DrawTarget 过程和窗体中文本框里的值。

这个小的工程几乎就要完成了。我们有了一个绘制靶形图的过程，有了一个让用户输入坐标的窗体。现在需要给用户一个显示窗体的途径。我们不想让用户进入 VBA 环境来运行这个程序，所以，再次对过程 Main 加以修改。

```
Sub Main()
    '显示窗体
    frmCh05.show
End Sub
```

现在保存一下 VBA 程序，丢失刚做的工作会很遗憾。在 VBA 中点击保存按钮。

保存了吧？好。运行你的程序看一看它工作的是多么的好。在 MicroStation 中按下键盘上的**<Alt>**键的同时按**<F8>**键。

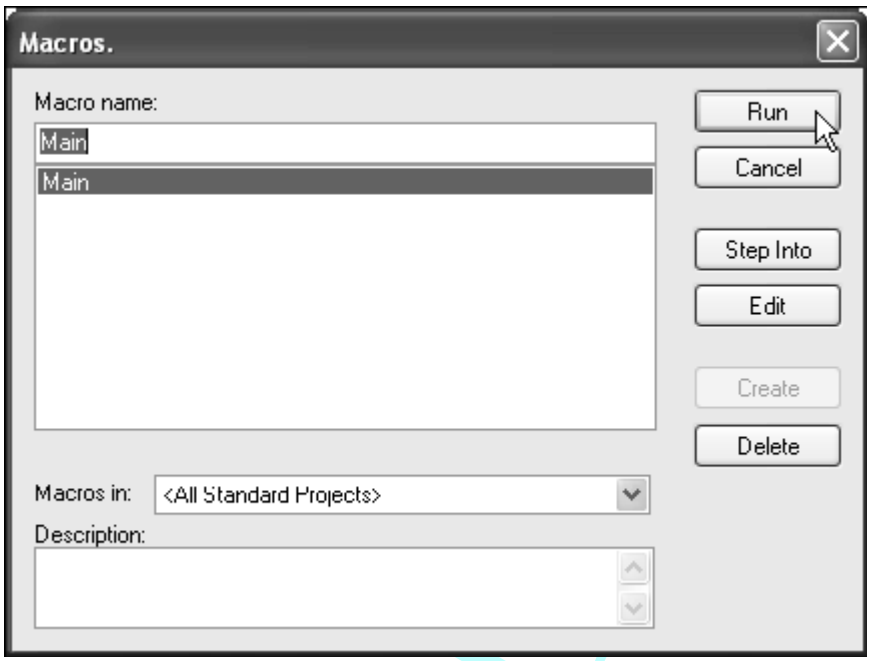
如下图所示的叫做 Main 的宏就是 Main 过程。选中它并点击 Run(运行)按钮。

此处出现窗体，我们可以输入 X、Y 和 Z 坐标来放置靶形图。

在继续之前，让我们回顾一下都做了什么事情。

我们想让用户在 MicroStation 中绘制一个靶形符号。第一件事情是需要写一些代码来正确地绘出靶的图形。这样我们就把代码放入了一个叫做 Main 的过程中，通过硬编码得到了绘制靶形的基本代码。当代码能正确工作以后，我们通过

把代码从 Main 里拿出并建立成一个新过程 DrawTarget 来使代码更灵活和有效。提供了三个参数用于指定靶的位置。所有这些工作完成后，通过修改 Main 代码来对 DrawTarget 过程进行测试。



下一步是建立用户界面(第 10 章会更多地使用窗体)。使用来自文本框的值作为 DrawTarget 的参数(使用标准 VBA 函数 Cdbl 将值从字符串转换为双精度)。最后一步是修改 Main 中的代码以显示用户窗体。

现在我们有了一个代码模块和一个窗体协同工作，使用 Main 来显示窗体。当用户点击按钮时，来自窗体的值就是 DrawTarget 的参数，DrawTarget 位于代码模块中。那么，DrawTarget 为什么要在代码模块中呢？就不能在窗体的代码区吗？能，它能位于窗体的代码区并且代码仍能工作。然而，这样做造成的后果就是其它模块和其它窗体将不能很容易地使用 DrawTarget。代码模块之所以这样重要的一个原因就是，代码模块中声明的过程和函数能容易地被工程中其它地方所利用。例如，有三个窗体在使用一个代码模块中的代码，也在使用另一个代码模块中的过程和函数。

现在，我们的代码工作得非常好，但是还能否使我们的工程更灵活和强大呢？这是一个好问题。

类

您可能知道用类来创建对象，但您是否知道把少许想法应用到创建类模块中，这些类就能在未来数年一直发挥作用。怎么能做到这样的呢？

当前工程能在指定的任一坐标处绘制靶形图。功能很强大，也能满足目前的

需要。然而，一年以后我们发现又想改变靶形图的大小，这时候又该怎么办呢？过程 DrawTarget 只允许三个参数项(X、Y 和 Z)，我们将修改这个过程，增加第四个参数也就是最后一个参数来指定靶的大小。但这样做会破坏原来的部分代码而且还需要使新参数为可选参数。下面是一个更好的解决方法。

建立一个含有 X、Y 和 Z 属性的新类，它还有一个 Draw 方法。这一切就绪后，再加入一个 Scale 属性。我们还能添加 Level 属性、Color 属性、NumberOfCircles 属性等等。这些属性可以在今天添加，也可以在明天甚至明年添加，什么时候添加都没有关系。我们只要保证添加新属性后原来使用该类的那些代码不需要修改就能继续正常工作就行了。

现在是写代码的时间了。

用 VBA 菜单 **Insert > Class Model(插入 > 类模块)** 在当前工程中添加一个新的类模块，命名为 clsTarget(用属性窗口来命名)。该类将含有三个属性和一个方法。为类实现属性的最基本的做法就是在类模块的通用声明区声明 Public 变量。在类模块中建立过程就能实现方法。

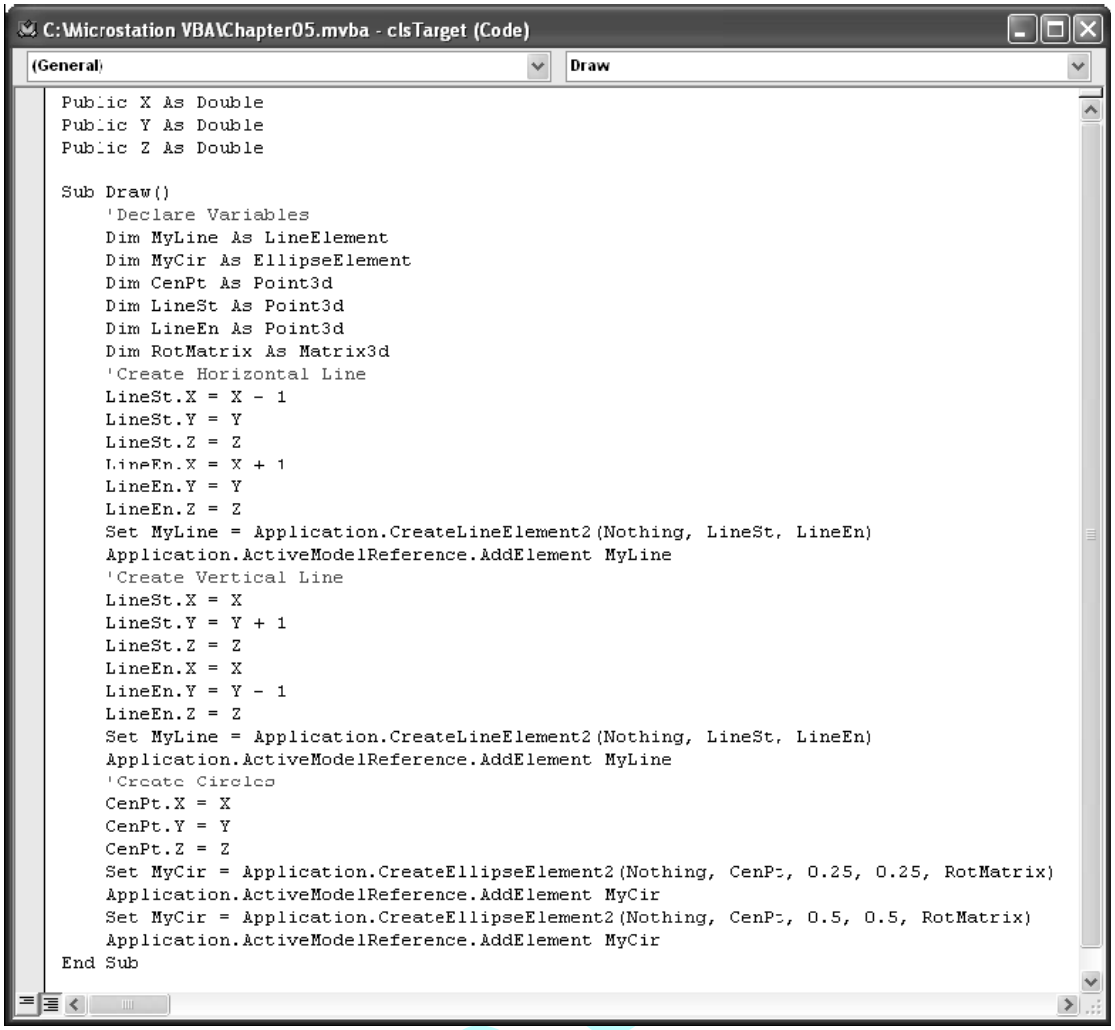
先从定义属性开始。

```
Public X As Double
Public Y As Double
Public Z As Double
```

下一步来实现 Draw 方法。不要忘了可以从随本书赠送的 CD 上得到最终的工程文件，从 CD 上打开就不需要键入整个代码了。Draw 方法是从 DrawTarget 复制并粘贴然后改 CenX 为 X，改 CenY 为 Y，改 CenZ 为 Z 而得来的，当然，Draw 方法要定义在类模块中。

确保类的所有代码都在同一页中，看一看完成后的类的屏幕快照。

China BDN

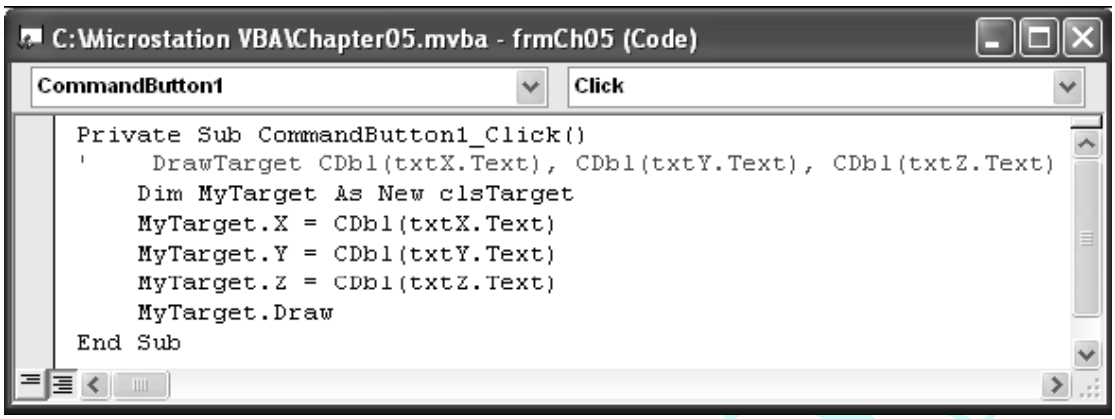


下面就是应当键入的代码：

```
Public X As Double
Public Y As Double
Public Z As Double
Sub Draw(X As Double, Y As Double, Z As Double)
    '声明变量
    Dim MyLine As LineElement
    Dim MyCir As EllipseElement
    Dim CenPt As Point3d
    Dim LineSt As Point3d
    Dim LineEn As Point3d
    Dim RotMatrix As Matrix3d
    '生成水平线
```

```
    LineSt.X = X - 1
    LineSt.Y = Y
    LineSt.Z = Z
    LineEn.X = X + 1
    LineEn.Y = Y
    LineEn.Z = Z
    Set MyLine = Application.CreateLineElement2(Nothing, LineSt, LineEn)
    Application.ActiveModelReference.AddElement MyLine
    '生成垂直线
    LineSt.X = X
    LineSt.Y = Y + 1
    LineSt.Z = Z
    LineEn.X = X
    LineEn.Y = Y - 1
    LineEn.Z = Z
    Set MyLine = Application.CreateLineElement2(Nothing, LineSt, LineEn)
    Application.ActiveModelReference.AddElement MyLine
    '生成圆
    CenPt.X = X
    CenPt.Y = Y
    CenPt.Z = Z
    Set MyCir = Application.CreateEllipseElement2(Nothing, CenPt, _
    0.25, 0.25, RotMatrix)
    Application.ActiveModelReference.AddElement MyCir
    Set MyCir = Application.CreateEllipseElement2(Nothing, CenPt, _
    0.5, 0.5, RotMatrix)
    Application.ActiveModelReference.AddElement MyCir
End Sub
```

一切就绪后就能像如下所示使用它了：



现在用 **clsTarget** 类来取代以前的 DrawTarget 过程。注释掉 DrawTarget 代码行，把一个变量声明为 New clsTarget，然后设置该对象的 X、Y 和 Z 属性并调用 Draw 方法。

人们总认为类模块是留给高级程序员用的，其实并非如此。正像我们这里看到，类实现起来非常容易、简单。在后面的一章中将更详细地讨论类。

过程和函数

模块、窗体和类都会用到过程和函数。用 Sub 关键字声明过程，用 Function 关键字声明函数。

让我们看几个过程和函数。

```
Sub DrawCircle()
    '声明变量
    Dim MyCir As EllipseElement
    Dim CenPt As Point3d
    Dim RotMatrix As Matrix3d
    '生成圆
    CenPt.X = 0
    CenPt.Y = 0
    CenPt.Z = 0
    Set MyCir = Application.CreateEllipseElement2(Nothing, CenPt, _
    0.25, 0.25, RotMatrix)
    Application.ActiveModelReference.AddElement MyCir
End Sub
```

DrawCircle 在(0,0,0)点处绘制了一个半径为 0.25 的圆。它不依赖任何其它的过程或函数就能独立运行。

```
Sub DrawCircle2(Radius As Double)
    '声明变量
    Dim MyCir As EllipseElement
    Dim CenPt As Point3d
    Dim RotMatrix As Matrix3d
    '生成圆
    CenPt.X = 0
    CenPt.Y = 0
    CenPt.Z = 0
    Set MyCir = Application.CreateEllipseElement2(Nothing, CenPt, _
    Radius, Radius, RotMatrix)
    Application.ActiveModelReference.AddElement MyCir
End Sub
```

DrawCircle2 是一个带有单个参数的过程。它不能独立运行——需要另一个过程或函数来运行它。当它被其它函数或过程调用时，所要求的“Radius(半径)”参数必须像下面这样提供：

```
    DrawCircle2    1.5


Sub DrawCircle3(X As Double, Y As Double, Z As Double, _
                    Optional Radius As Double = 1.25)
    '声明变量
    Dim MyCir As EllipseElement
    Dim CenPt As Point3d
    Dim RotMatrix As Matrix3d
    '生成圆
    CenPt.X = X
    CenPt.Y = Y
    CenPt.Z = Z
```

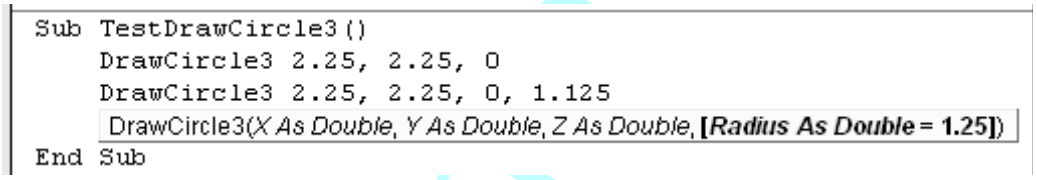
```
    Set MyCir = Application.CreateEllipseElement2(Nothing, CenPt, _
    Radius, Radius, RotMatrix)
    Application.ActiveModelReference.AddElement MyCir
End Sub
```

DrawCircle3 要求提供 X、Y 和 Z 值和可选的半径值。如果给了这个半径就使用这个值；如果未给这个半径，就用 1.25 这个值。

下面是测试 DrawCircle3 过程的一个方法。

```
Sub TestDrawCircle3()
    DrawCircle3 2.25, 2.25, 0
    DrawCircle3 2.25, 2.25, 0, 1.125
End Sub
```

第一次调用 DrawCircle3 时未提供可选参数，第二次调用时给了一个 1.125 的半径值。这样，第一个圆将以半径 1.25(缺省值)绘出，第二个圆将以半径 1.125 绘出。



一旦在过程中声明了一个可选参数，该参数之后的任何参数也都必须是可选的。

当调用函数和过程时，VBA 能显示正调用的函数或过程的参数提示。注意以上半径参数是如何被括在方括号中的。方括号告诉我们“半径”是一个可选参数，同时也示出了可选参数的缺省值。

在过程或函数中还能声明另一种叫做参数数组(ParamArray)的参数类型。参数数组必须是声明中的最后一个参数，因为在向其提供参数值时，参数值的个数是可变的。下面是一个声明和使用参数数组的例子：

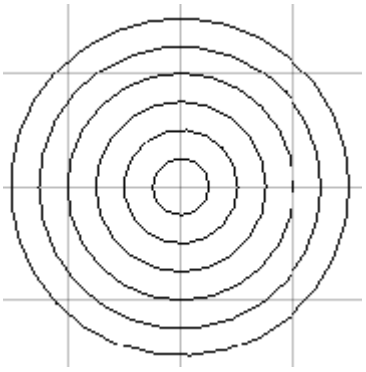
```
Sub DrawCircle4(X As Double, Y As Double, Z As Double, _
                ParamArray Radii() As Variant)
    '声明变量
    Dim MyCir As EllipseElement
```

```
    Dim CenPt As Point3d
    Dim RotMatrix As Matrix3d
    Dim I As Long
    '生成圆
    CenPt.X = X
    CenPt.Y = Y
    CenPt.Z = Z
    For I = LBound(Radii) To UBound(Radii)
        Set MyCir = Application.CreateEllipseElement2(Nothing, CenPt, _
                    Radii(I), Radii(I), RotMatrix)
        Application.ActiveModelReference.AddElement MyCir
    Next I
End Sub
```

我们事先并不知道 Radii 参数数组中有多少个半径值，所以在程序中使用了一个 For…Next 循环来查看每一个值并分别以之为半径来生成新的圆。下面是一个如何在代码中调用带有参数数组的过程的例子：

```
    Sub TestDrawCircle4()
        DrawCircle4 1, 1, 0, 0.25, 0.5, 0.75, 1, 1.25, 1.5
    End Sub
```

我们给 X 赋 1，Y 赋 1，Z 赋 0，然后开始提供半径值。代码运行后在 MicroStation 设计文件中生成六个新的圆。



右图是由 TestDrawCircle4 生成的六个圆。

本章迄今为止已经建立了超过 100 行的代码。当前模块中有九个不同的过程，其中五个能直接运行，其它几个必须由别的过程或函数调用。

说到函数，让我们仔细地分析一下它们。

```
    Function Pi() As Double
        Pi = Atn(1) * 4
    End Function
```

这里就是一个叫做 Pi 的函数。它不接受任何参数，返回值的类型是双精度。通过把返回值赋给函数名从而指定要返回什么值。

现在这个 Pi 函数可以在任何需要 Pi 值的地方使用。DrawCircle3 过程让我们提供圆的半径来绘制圆，但当我们只知道圆的面积时如何绘制这个圆呢？可以通过面积计算出半径，但需要这个 Pi 值。我们不是把 Pi 硬编码为“3.14159”而是使用刚刚建立的 Pi 函数。

```
Sub TestPi()
    Dim CircleArea As Double
    Dim CircleRadius As Double
    CircleArea = 3.5
    CircleRadius = Sqr(CircleArea / Pi)
    DrawCircle3 2.5, 2.5, 0, CircleRadius
End Sub
```

我们基于所给的面积计算出了圆的半径，然后把这个半径用到了过程 DrawCircle3 的半径参数中。

刚建立的函数 Pi 没有任何参数，因为计算总是相同的而不需要参数了。让我们看一些时不时会用到的其它函数，它们被命名为 RTD(弧度到角度)和 DTR(角度到弧度)。

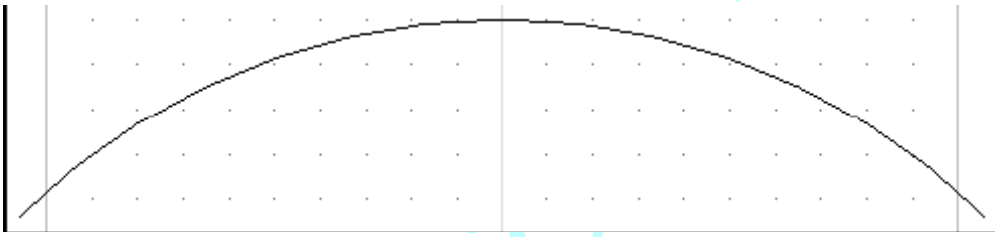
```
Function RTD(AngleInRadians As Double) As Double
    RTD = AngleInRadians * 180 / Pi
End Function
Function DTR(AngleInDegrees As Double) As Double
    DTR = AngleInDegrees * Pi / 180
End Function
```

这两个函数执行弧度角度互换计算，这种计算对于使用 MicroStation 的人来说是非常普遍的。在这里演示它们仅为教学之目的，因为 MicroStation VBA 中有一个 Degrees 函数转换弧度到角度，还有一个 Radians 函数转换角度到弧度。

DTR 和 RTD 函数(也包括 Degrees 和 Radians)只要求一个参数。下面是如何使用它们的例子：

```
Sub DrawArc1()
    Dim MyArc As ArcElement
    Dim CenPt As Point3d
    Dim RotMatrix As Matrix3d
    Set MyArc = Application.CreateArcElement2(Nothing, CenPt, 1.5, _
        1.5, RotMatrix, DTR(45), DTR(90))
    Application.ActiveModelReference.AddElement MyArc
End Sub
```

CreateArcElement2 要求几个参数，其中一个是起始角，另一个是扫描角。这两个参数都要求以弧度形式给值。人们通常大多都以角度来思考问题而不是以弧度，所以如上所示用 DTR 函数将角度(我们思考问题时所用)转换为弧度(函数所期望的)。



这就是以上代码生成的弧。它开始于 45 度，扫描过去的角度为 90 度。

返回一个数组

函数返回一个值，对吗？对。但通过采用数组函数实际上能返回更多的值。

下一章将对数组进行更多的讨论，但现在我们知道一个数组就是含有多于一个值的变量，能在函数中返回数组。下面代码就是这样：

```
Function PolarPoint(X As Double, Y As Double, Z As Double, _
                    Angle As Double, Distance As Double) As Variant
    Dim XChange As Double
    Dim YChange As Double
    XChange = Cos(Angle) * Distance
    YChange = Sin(Angle) * Distance
    Dim PPoint(0 To 2) As Double
    PPoint(0) = X + XChange
```

```
    PPoint(1) = Y + YChange
    PPoint(2) = Z
    PolarPoint = PPoint
End Function
```

续行符(一个空格跟上一个下划线字符)能使代码跨越多行。

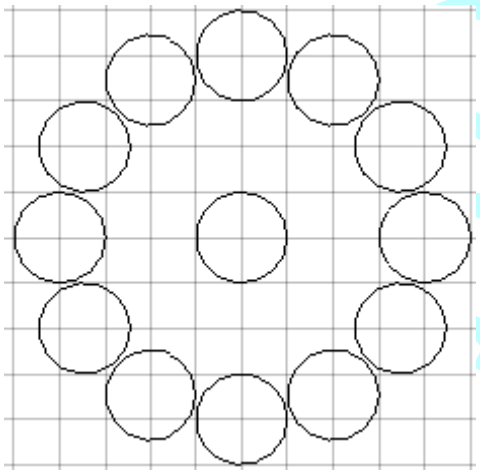
PolarPoint 函数定义一个起点(X、Y 和 Z)、一个角度和一个距离。它以数组形式返回坐标 X、Y 和 Z。

通过声明函数的返回类型为 variant 来返回一个数组。正如我们将要在变量一章中讨论的那样，一个 variant 变量能保存任何类型的值、对象、值数组或对象数组。我们在函数中声明了一个双精度的数组，然后将该数组赋给了函数名。下面是对 PolarPoint 函数的一个测试。

```
Sub TestPolarPoint()
    Dim StartCen As Point3d
    Dim CenPt As Point3d
    Dim RotMatrix As Matrix3d
    Dim X As Variant
    StartCen.X = 2
    StartCen.Y = 2
    StartCen.Z = 0
    Set MyCir = Application.CreateEllipseElement2(Nothing, _
                StartCen, 1, 1, RotMatrix)
    Application.ActiveModelReference.AddElement MyCir
    Dim RotAngle As Double
    For RotAngle = 0 To 360 Step 30
        X = PolarPoint(StartCen.X, StartCen.Y, StartCen.Z, _
            DTR(RotAngle), 4)
        CenPt.X = X(0)
        CenPt.Y = X(1)
        CenPt.Z = X(2)
```

```
        Set MyCir = Application.CreateEllipseElement2(Nothing, _
            CenPt, 1, 1, RotMatrix)
        Application.ActiveModelReference.AddElement MyCir
    Next RotAngle
End Sub
```

运行 TestPolarPoint 后能得到什么呢？



返回“类型”

迄今为止，我们编写的函数不是返回一个值就是返回一个数组，其实函数还能返回类型。MicroStation VBA 有一个 Point3d 类型，它有三个属性：X、Y 和 Z。让我们复制并粘贴 PolarPoint 函数来使用这个类型。

```
Function PolarPoint2(X As Double, Y As Double, Z As Double, _
                     Angle As Double, Distance As Double) As Point3d
    Dim XChange As Double
    Dim YChange As Double
    XChange = Cos(Angle) * Distance
    YChange = Sin(Angle) * Distance
    Dim PPoint(0 To 2) As Double
    PolarPoint2.X = X + XChange
    PolarPoint2.Y = Y + YChange
    PolarPoint2.Z = Z
End Function
```

我们用返回一个 Point3d 类型代替了前例中返回的数组。下面是使用 PolarPoint2 函数的一个例子：

```
Sub TestPolarPoint2()
    Dim StartCen As Point3d
    Dim CenPt As Point3d
    Dim RotMatrix As Matrix3d
    Dim X As Variant
    StartCen.X = 2
    StartCen.Y = 2
    StartCen.Z = 0
    Set MyCir = Application.CreateEllipseElement2(Nothing, _
                StartCen, 1, 1, RotMatrix)
    Application.ActiveModelReference.AddElement MyCir
    Dim RotAngle As Double
    For RotAngle = 0 To 360 Step 30
        CenPt = PolarPoint2(StartCen.X, StartCen.Y, StartCen.Z, _
                DTR(RotAngle), 4)
        Set MyCir = Application.CreateEllipseElement2(Nothing, _
                  CenPt, 1, 1, RotMatrix)
        Application.ActiveModelReference.AddElement MyCir
    Next RotAngle
End Sub
```

返回对象

另一种返回类型值得关注。函数除了返回值和类型外，还能返回对象。看一个例子。

```
Function GetExcelWS() As Object
    Dim ExcelApp As Object
    Set ExcelApp = GetObject(, "Excel.Application")
    Set GetExcelWS = ExcelApp.activesheet
```

China BDN

```
End Function
```

这个函数取得 Microsoft Excel 中的当前工作表。**该函数要能正常工作必须要 Excel 处于运行状态。**怎样使用这个函数？看下面的代码。

```
Sub TestGetExcelWS()
    Dim MyWS As Object
    Dim Cell1 As Double
    Dim Cell2 As Double
    Dim Cell3 As Double
    Set MyWS = GetExcelWS
    Cell1 = MyWS.Range("B2")
    Cell2 = MyWS.Range("C2")
    Cell3 = MyWS.Range("D2")
End Sub
```

这个过程取得 Excel 中三个单元的值。从 Excel 中得到数据或向 Excel 中写入数据真的非常容易。本书后面将会更详细地讨论与 Microsoft Excel 协同工作的情况。

ByVal 和 ByRef(按值和按引用传递)

我们已经看过了在调用过程和函数时是如何提供参数的。缺省情况下，参数值是按引用被传递的。传递参数的另一种方式是按值传递。这些都是什么意思？

```
Sub GetThreeVals(X As Double, Y As Double, Z As Double)
    X = 1
    Y = 2
    Z = 3
End Sub
```

该过程接受三个参数。我们在代码中给参数名赋了值。明白这一点很重要，因为直接以这种方式使用变量将改变调用该过程的函数或过程中变量的值。

```
Sub TestGetThreeVals()
    Dim A As Double
    Dim B As Double
```

China BDN

```
    Dim C As Double
    A = 100
    B = 200
    C = 300
    GetThreeVals A, B, C
End Sub
```

这里有变量 A、B 和 C，分别给它们赋值为 100、200 和 300。然后在调用 GetThreeVals 时使用了这些变量(A、B 和 C)。由于 GetThreeVals 过程在声明参数时没有用关键字 ByVal，所以参数值以 ByRef 方式传递进来。ByRef 的意思是：参数的值可以在过程内被修改。如果参数的值在过程中被修改了，这些变量在过程外也将保持它们的值。所以，在“GetThreeVals A, B, C”代码行被执行前，A、B 和 C 的值是 100、200 和 300。GetThreeVals 被执行后，A、B 和 C 的值就是 1、2 和 3 了。

GetThreeVals 改变了传递给它的参数的值。如果能正确地使用这一点，它就是一个强大的特性。如果没有搞明白这一点，还会引起极大的混乱。这个混乱就是：一个变量的值怎么会突然发生了改变。

如果不想让函数或过程改变参数变量的值，有好几种做法都能实现这一点。第一种做法要求有一定的修炼才行，第二种做法更明确一些。

```
Sub GetThreeVals2(X As Double, Y As Double, Z As Double)
    Dim dblX As Double
    Dim dblY As Double
    Dim dblZ As Double
    dblX = X
    dblY = Y
    dblZ = Z
    dblX = 1
    dblY = 2
    dblZ = 3
End Sub
```

我们在过程内声明变量，把参数的值赋给这些变量而不直接操作参数，然后我们操作这些局部变量。这就保证了不会改变参加的值。

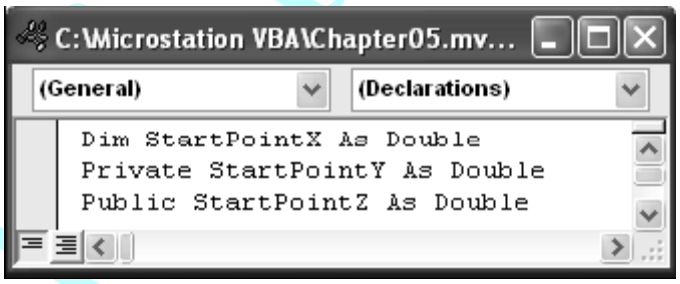
另一种保证传递给函数和过程的参数不变的做法是明确地声明这些参数为ByVal。

```
Sub GetThreeVals3(ByVal X As Double, ByVal Y As Double, _
                  ByVal Z As Double)
    X = 1
    Y = 2
    Z = 3
End Sub
```

把参数声明为 ByVal，多做这一步就能保证参数的完整性。

声明变量

代码中无处没有变量。变量以一个名字和一种类型来声明。下一章将学习更多关于声明变量的知识。现在需要明白的是变量具有范围。能预先确定变量在什么时间可被使用。变量的范围依赖于声明它的位置以及声明它时用了什么关键字(如果有的话)。有两个地方可以声明变量：一个地方是位于使用这些变量的过程和函数内部，至今已经看过许多这样的例子了；另一个地方是在代码模块、窗体和类模块的通用声明区。



在该代码模块的通用声明区，我声明了三个相同类型(双精度)的变量但使用了不同的关键字：Dim、Private 和 Public。

```
Dim StartPointX as Double
```

在代码模块或窗体代码的通用声明区用“Dim”声明的变量意味着该变量只能在该模块、窗体或类模块中使用。再看一下，StartPointX 是用 Dim 关键字声明的，所以它只能在声明它的模块或窗体中使用。

“Private(私有)”和“Dim”的效果一样。可以在声明它的代码区内的任意函数或过程中使用这个变量。一个函数设置变量的值，同一模块、窗体或类模块中的另一个函数能读取这个值。

“Public(公共)”变量声明依赖于声明的位置而有不同的表现。

窗体——对于在窗体中声明的公共变量，当窗体在作用范围内(通常仅当窗体显示时)时它才在作用范围内。工程中其它地方可通过窗体名来访问该变量。例如，

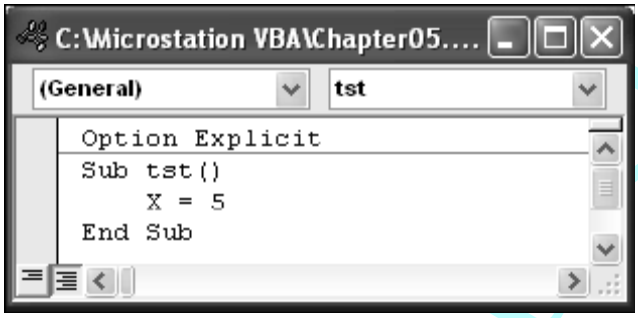
```
UserForm1.TestVariable = 4.5
```

我们只能在整个窗体中使用变量 TestVariable 并且此时窗体必须“在作用范围内”。

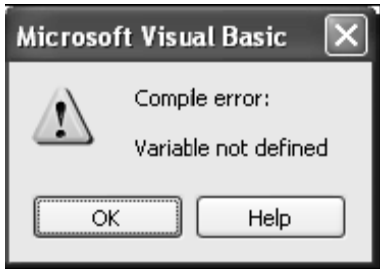
模块——模块中声明的公共变量在整个工程中都有效。

类——类中声明的公共变量被看作可读/写的属性。

Optional Explicit



缺省情况下，如果使用一个没有声明的变量，它将继承“Variant”类型。在模块、窗体和类的通用声明区使用“Optional Explicit”能强制我们必须声明变量后才能使用变量。



本例中，我们在通用声明区声明了“Optional Explicit”。当试着去运行上面的 test 宏时将得到一个错误。

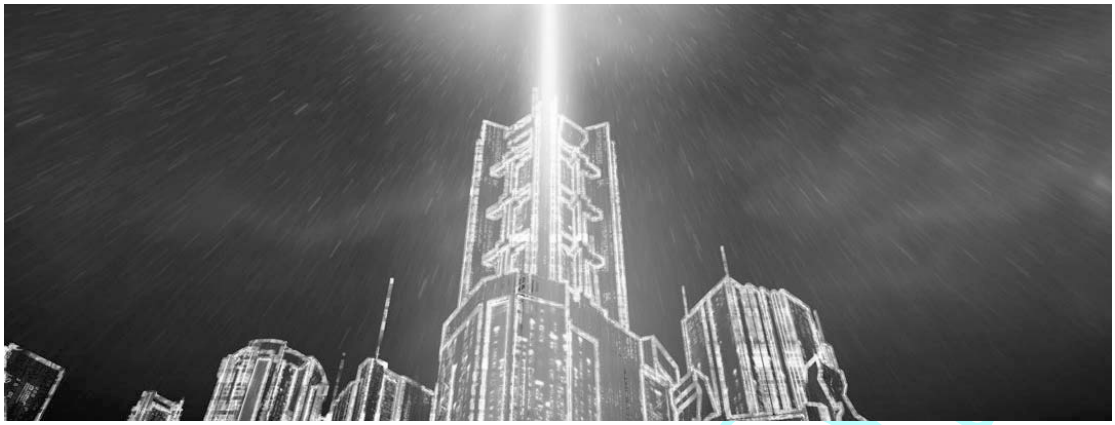
要避免该错误的出现，需要将 X 声明为双精度、整型或长整型。更多关于变量类型的内容在下一章介绍。

本章回顾

- 在过程、函数或用户窗体事件中写代码；
- 在过程和函数中利用必需的和可选的参数；
- 函数可返回值、数组、类型和对象；

- 在过程和函数中，如果参数声明为“ByRef”，则可以改变作为参数传递进来的变量。把参数声明为“ByVal”将保持变量不变；
- 能在过程、函数和事件中或者在通用声明区内声明变量。这些变量的范围依赖于它们在何处被声明以及声明中所用的关键字。

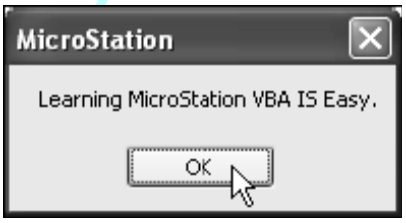
China BDN



6. 变 量

1 + N + 3 = 7

N 是什么？N 是一个变量。在上面的等式中，N 代表一个数字。如果要求解 N 的话将得到值 3。



“Learning MicroStation VBA” & N & “Easy.”

N 是什么？N 是一个变量。在上面的等式中，N 代表一个字符串。它代表什么字符串呢？”IS”。

本章介绍：

- 标准 VBA 变量类型
- MicroStation 特有的变量类型
- 赋值和设置对象
- 数组
- 常数
- 变量名
- Optional Explicit
- 使用变量

标准 VBA 变量类型

一个变量就是代表一个值或一个对象的名字。上面所示的例子中变量的名字是 N。在第一个例子中，变量保存一个数字值；在另一个例子中变量保存一个字符串。我们事先就知道一个变量代表的是什么值类型或对象类型，所以我们通过

声明变量来指定变量所用的类型。

```
Dim N as Integer
N = 7 - 3 – 1
```

这里，我们声明变量 N 为整型(integer)，意思是该变量为一个在-32,768 到 32,767 之间的整数。

```
Dim N As String
N = "IS"
MsgBox "Learning MicroStation VBA " & N & " Easy."
```

这里，我们声明变量 N 为字符串(string)。字符串是一组字符。变量被赋值后就能用在数字、文本或一些其它值类型和对象类型的地方。

本书从始至终都在用变量。下面我们研究一些最常用的变量类型。

Integer(整型)

```
Dim PageNumber as Integer
PageNumber = 2
```

我们说过，整型就是一个在-32,768 到 32,767 之间的整数。如果变量保存的值大于或小于整型范围，那就必须把它声明为不同的类型。

Long(长整型)

```
Dim MySalary as Long
MySalaray = 123456
```

长整型就是一个在-2,147,483,648 和 2,147,483,647.之间的整数。这些数比整型数的范围大得多。长整型需要更多的内存来保存这个较大范围的数字，所以仅在需要时才用它。

Double(双精度型)

```
Dim HoursToLearnVBA as Double
HoursToLearnVBA = 36.25
```

double 也叫作双精度浮点数字。这是什么意思？意思是 double 的精度是 single(也是一种变量类型但用的不多)精度的两倍且小数点位置可以浮动，小的数就给以大的精度，大的数就减小精度。理解描述中“浮点”部分的含义很重要。

如果希望有一个极大的数字精度极高，那 double 类型不仅会让您失望而且还比您所期望的精度低。

考虑如下的宏 VariableTestC。它有一个变量 N 声明为 double，其中保存数字“1234567890123456789”，每行数字的小数点位置都不同，最后两个数字是“46”。VBA 圆整“456”到“46”，因为双精度变量用指定的内存量来保存其值，当试图向其中放入多于其能保存的量时，将被圆整到所能保存的数值。

```
Sub VariableTestC()
    Dim N As Double
    N = 1.23456789012346
    N = 12.3456789012346
    N = 123.456789012346
    N = 1234.56789012346
    N = 12345.6789012346
    N = 123456.789012346
    N = 1234567.89012346
    N = 12345678.9012346
    N = 123456789.012346
    N = 1234567890.12346
    N = 12345678901.2346
    N = 123456789012.346
    N = 1234567890123.46
    N = 12345678901234.6
End Sub
```

双精度能保存非常精确的数字，但随着数值的增加其精度在降低。在开发应用时对这一点需要有印象。

Boolean(布尔型)

```
Dim ICanLearnThis as Boolean
ICanLearnThis = True
```

布尔型数据类型能保存两个值中的一个：True(真)或 False(假)。

China BDN

Date(日期型)

```
Dim XMReleaseDate as Date
XMReleaseDate = “5/19/2006 8:00:00 AM”
```

日期数据类型保存日期/时间值。

String(字符串型)

```
Dim MyLevelName as String
MyLevelName = “utilElectricity”
```

字符串数据类型包含文本。字母、数字以及在计算机键盘上的其它字符都能保存在这种变量中。数字变量类型都有值的范围，这是因为那些数据类型都有一个预定量的内存来保存。字符串类型也不例外，那么一个字符串变量能保存多少字符呢？将近 20 亿(2,000,000,000)，这可是许多字符呀。

Object(对象型)

```
Dim MyExcelApp as Object
Set MyExcelApp = GetObject(, "Excel.Application")
```

对象类型变量指向对象。例如，变量 MyExcelApp 指向 Microsoft Excel 的一个实例，也就是一个应用。Microsoft Excel 是一个具有特定属性、方法和事件的对象，其它对象有其独自的属性、方法和事件。把变量声明为一个对象时，该变量是一个通用的对象，没有任何属性、方法和事件。仅当把该变量设置到一个具体对象后才知道它是哪种对象以及其属性、方法和事件。

Variant

```
Dim PointArray as Variant
```

被声明为 Variant 的变量能保存任何类型的值，能指向任何类型的对象，甚至能包含一个数组。

MicroStation 特有的变量类型

我们已经讨论的变量类型都是标准的 VBA 变量类型。它们能用于 MicroStation VBA、Excel VBA、Word VBA 或 Access VBA。接下来让我们考察 MicroStation 特有的变量类型，这些类型是我们今后经常要用到的。

Application(应用)

Dim MSApp As Application

Set MSApp = Application

Application 变量类型指向 MicroStation 应用。它是操纵 MicroStation 时的顶级对象。能从 Application 做如下事情：

- 取得 ActiveDesignFile(当前设计文件)属性
- 取得 ActiveModelReference(当前模型参考)属性
- 取得 ActiveSettings(当前设置)对象及其属性
- 取得正在执行的 VBProject 对象及其属性
- 取得 UserName(用户名)属性
- 取得左、顶、宽和高属性

DesignFile(设计文件)

Dim MyDGN As DesignFile

Set MyDGN = Application.ActiveDesignFile

DesignFile 对象指向一个 MicroStation DGN 文件。通过 DesignFile 对象能获得最顶层 DGN 的属性和集合。

- 取得和设置 Author(作者)、Client(客户)、Comments(注释)、Company(公司)、Keywords(关键字)、Manager(管理者)、Subject(主题)和 Title(标题)属性
- 取得 FormatMajorVersion(格式主版本)和 FormatMinorVersion(格式次版本)属性
- 取得 Levels(层)集合
- 取得 Models(模型)集合
- 取得 Name(名称)和 Path(路径)属性

ModelReference(模型参考)

Dim MyModel As ModelReference

Set MyModel = Application.ActiveModelReference

ModelReference 对象是用的最多的对象。当在文件中绘图时要用

ModelReference 对象，当在文件中查找时要用 ModelReference 对象。整本书中都将会使用它。

Level(层)

```
Dim MyLevel As Level
Set MyLevel = Application.ActiveDesignFile.Levels(1)
```

层能把设计细分成多个对象组。我们通常按图形对象的几何或标注类型来分组。道路中心线可能在一个层上，地块区号在另一个层上，图置标题框线也在其独立的一个层上等等。下面是几个可取得和设置的层对象属性：

- Description(描述)
- ElementColor(元素颜色)
- ElementLineStyle(元素线型)
- ElementLineWeight(元素线宽)
- IsActive(是否激活)
- IsDisplayed(是否显示)
- IsFrozen(是否冻结)
- IsLocked(是否锁定)
- Name(名字)
- Number(层号)
- Plot(光栅绘图)

LineElement(线元素)

```
Dim MyLine As LineElement
Set MyLine = Application.CreateLineElement2(Nothing, _
Point3dFromXYZ(0, 0, 0), Point3dFromXYZ(4, 5, 6))
Application.ActiveModelReference.AddElement MyLine
```

用一个起点和一个终点来创建线元素。线元素建立后就可以加入到模型中。

EllipseElement(椭圆元素)

```
Dim MyCircle As EllipseElement
Dim RotMatrix As Matrix3d
```


China BDN

```
Set MyCircle = CreateEllipseElement2(Nothing, _
Point3dFromXYZ(0, 0, 0), 1.5, 1.5, RotMatrix)
Application.ActiveModelReference.AddElement MyCircle
```

线、圆和弧构成了 MicroStation 文件中大多数的几何形状。从 MicroStation 的角度来看，圆实际上就是长轴半径和短轴半径相等的椭圆。上面所示的代码绘制了一个中心点为(0,0,0)半径为 1.5 的圆。

ArcElement(弧元素)

```
Dim MyArc As ArcElement
Dim RotMatrix As Matrix3d
Set MyArc = CreateArcElement2(Nothing, _
                    Point3dFromXYZ(0, 0, 0), 1.75, 1.75, _
                    RotMatrix, Radians(45), Radians(90))
Application.ActiveModelReference.AddElement MyArc
```

MicroStation 给出了多种建立新的线元素、椭圆元素和弧元素的方法。在本例中，通过提供一个中心点、长轴和短轴半径、一个起始角和一个扫描角来建立弧元素的。

TextElement(文本元素)

```
Dim MyText As TextElement
Dim RotMatrix As Matrix3d
Set MyText = CreateTextElement1(Nothing, "MicroStation VBA", _
Point3dFromXYZ(0, 0, 0), RotMatrix)
Application.ActiveModelReference.AddElement MyText
```

建立文本元素对象需要所显示的文本和一个起点。文本元素建立后还能设置它的其它属性，如颜色、层和字型(字型包括字体、大小等)。

在进行 MicroStation VBA 编程的工作中，我们会使用到非常多的对象类型，对于上面刚刚介绍过的几个对象也有更多的内容要学习。所有这些都会随着继续深入学习 MicroStation VBA 而得到更加详细的说明。

赋值和设置对象

值类型变量和对象类型变量的声明是相似的，但给变量赋值或把值设置给对象却有很大的不同。

```
Dim LevelName As String
LevelName = "Easement"
Dim EasementLevel As Level
Set EasementLevel = _
        ActiveDesignFile.AddNewLevel(LevelName)
```

这里有两个变量，一个被声明为字符串，另一个被声明为层。

给变量 **LevelName** 赋值的步骤是：变量名，等号，所需的值。而在使用层对象时是通过关键字“Set”把变量设置到一个对象。仅在把变量设置到一个对象时才用“Set”。在把变量设置到一个对象后就能用变量名来访问对象了。

数组(Array)

在 MicroStation 中提到 Array 这个词，我们会想到把一个元素复制多次(译者注：Array 是 MicroStation 中的元素阵列功能)。VBA 中的 Array 是一个存有多个元素的变量名，即数组。

```
Dim StartPoint(0 to 2) as Double
StartPoint(0) = 4.5
StartPoint(1) = 5.6
StartPoint(2) = 6.7
```

我们在本例中声明了一个数组变量，它有三个编号为 0、1 和 2 的元素。通过指定数组的索引能单独定位到某个元素。

在下一个例子中，我建立了一个 Point3d 类型的数组，Point3d 具有 X 属性、Y 属性和 Z 属性。请注意我是如何靠数组索引(0 和 1)来定位数组中的每个元素，尔后又是如何定位 X、Y 和 Z 属性的。

```
Sub ArrayTestA()
    Dim MyVerticies(0 To 1) As Point3d
    Dim MyLine As LineElement
```

```
    MyVerticies(0).X = 1
    MyVerticies(0).Y = 2
    MyVerticies(0).Z = 3
    MyVerticies(1).X = 4
    MyVerticies(1).Y = 5
    MyVerticies(1).Z = 6
    Set MyLine = CreateLineElement1(Nothing, MyVerticies)
    ActiveModelReference.AddElement MyLine
End Sub
```

常数

常数与变量类似但又有一个显著的不同点：常数值不能改变！

```
Const PI As Double = 3.14159
```

可以如上所示在代码模块的通用声明区声明公共常数。当你需要一个不变的值时定义一个常数会很有用。例如，假设您在编写一个对设计文件中的线标注其长度的程序，这种情况下就可以把标注文字与线之间的距离定义为一个常数。常数也能用作单位转换比例，如从英寸到库比特长度单位的转换。

使用常数的另一个原因是，在计算中要多次引用一个值，如一个 ShimWidth 值。这种情况下，我们声明一个常数叫做 **ShimWidth** 并赋以 0.6 的常数值，然后把程序中出现 0.6 的地方都替换为该常数。这使得代码更容易读懂且当想要改变 **ShimWidth** 这个值时只需要改变一个地方(就是声明常数的地方)就行了。

变量名

有数千页的文字致力于探讨如何命名变量。我们应该先搞清 VBA 强加给我们的一些规则后才能开始这个讨论。

- 变量必须以字母字符(A 到 Z)开始；
- 变量名不能含有空格；
- 变量名字符是 A～Z、0～9 和_(下划线)；
- 变量名长度不能大于 255 个字符；
- 变量名不能是关键字(如：Dim、New、Left、Right、Form、Public)；

➢ 变量名所用的字母可以是大写的也可以是小写的。

基于已经说明的规则，下面是一些能正常工作的变量声明：

```
Dim myLine As LineElement
Dim txteMyText As TextElement
Dim strName As String
Dim dblStartX As Double
Dim intLevelNumber As Integer
Dim pt3dStartPoint As Point3d
```

如上所示的每个声明都是合法的变量声明，它们符合规则。第一个变量 myLine 和其它几个稍微有些不同。其它的变量声明都以指定其变量类型的字符开头。strName 说明变量类型是字符串，dblStartX 说明变量类型是双精度型。

了解一个工程中是否要求采用变量命名约定很重要。命名约定是一个附加的关于如何命名变量的一套规则。例如，某一个约定可能要求每个变量名要以三个字符后跟一下划线(_)开头，其后是不多于七个字符的名字；另一个约定可能不使用下划线；还有一个约定可能要求在变量名中指定变量的作用范围。

正如前面提及的那样，有许多文章致力于变量命名约定这个主题，所以我们不再花更多的时间在这个专题上。您应当明白把一个变量命名为 myLine 或 lineMyLine 或 line123 与命名为 elemline_LineA 对程序的工作没有任何影响。

命名约定可扩展到变量命名范围以外。过程名、函数名和控件名也都属于命名约定的范畴。

下面是一个讨论变量命名约定的网页：

http://msdn.microsoft.com/library/en-us/modcore/html/deconVariableNames.asp

想熟悉命名约定的另一途径是在线搜索“variable naming convention(变量命名约定)”或“Hungarian Notation(匈牙利符号)”。

区分大小写

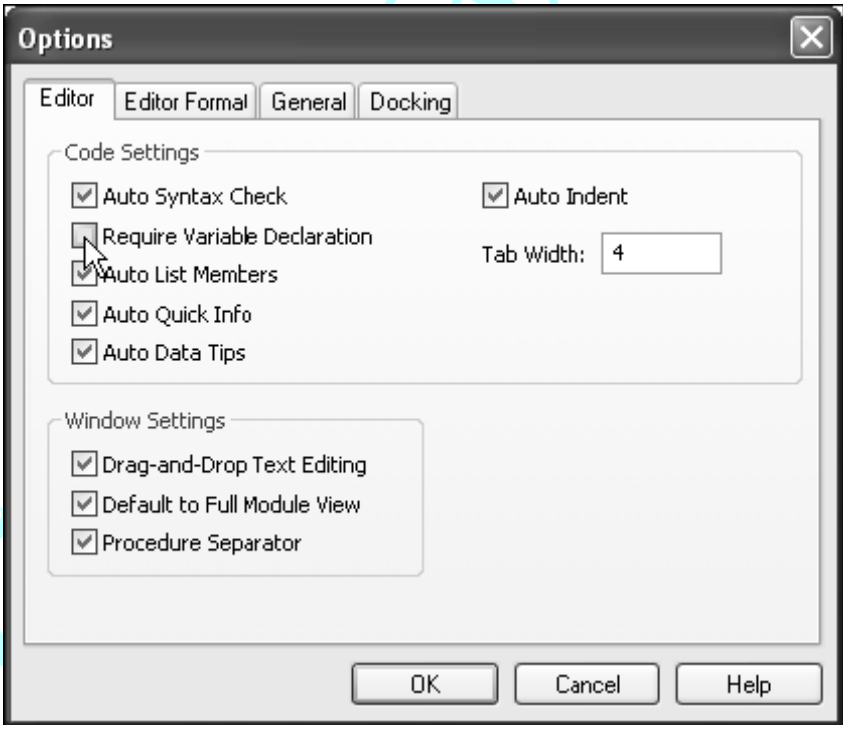
请看如下变量声明：

```
Dim myLine As LineElement
Dim MYline As LineElement
```

由于 VBA 不关心变量名的大小写，因而以上声明的两个变量是相同的。在 VBA 中使用变量时，VBA 会自动将变量名的大小写改成声明时的样子。有些程序员正是利用了这一特性来保证其代码键入的正确性。他们在声明变量时总是按一定规则在变量名的某些地方用大写字母，在使用变量时全部小写键入。当输入完一行代码并把光标移到下一行时，VBA 会自动更新那些以小写键入的变量的大小写。

Optional Explicit

我们已经谈论变量类型和声明变量很长一段时间了。有许多人抱怨：为什么非要声明变量。其实 VBA 并没有强制我们这样做。在一个变量没有被正式声明前就能使用它。这样做时，变量会具有首次使用它时的一种特定变量类型特性。例如，如果给变量的值是 1.23456 则变量将是一个“Variant Double”类型；或者如果给变量的值是“owhatafooliam”则变量将变成一个“Variant String”类型。确保变量需要声明后才能使用的一种方法是，在模块的通用声明区使用“Optional Explicit”。另一种方法是操作菜单 **Tools > Options (工具 > 选项)**。



在 “Require Variable Declaration(要求变量声明)”按钮上打勾能使 VBA 为我们输入“Optional Explicit”。

使用变量

声明了一个变量并且对其赋值或把它设置到一个对象后，该变量可在任何需要其值的时候来使用。

```
Sub VariableTestD()
    Dim MySalary As Double
    Dim MyHourly As Double
    MySalary = 1234567
    MyHourly = MySalary / 52 / 40
    MsgBox "My Hourly Rate is " & FormatCurrency(MyHourly, _
            2, vbFalse, vbFalse, vbTrue)
End Sub
```



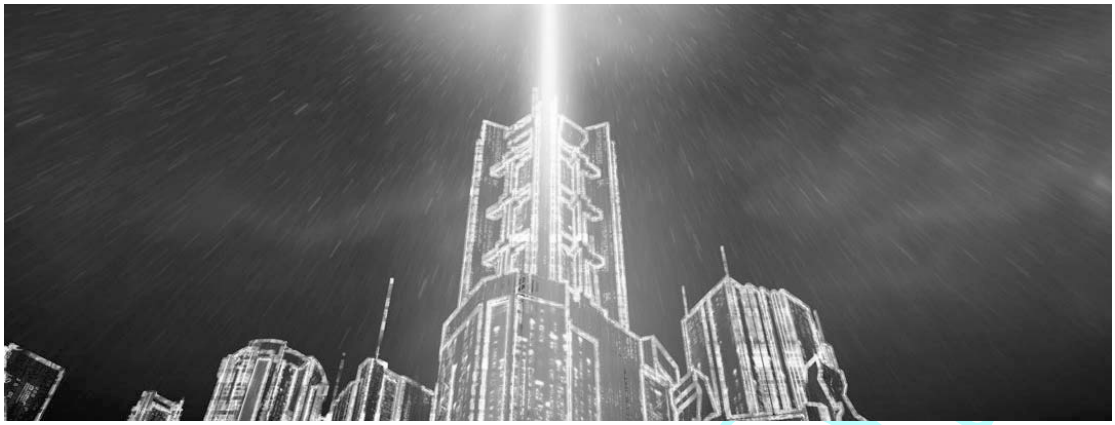
此处有两个变量，其中之一(**MySalary**)被赋以值1234567。用 **MySalary**(年薪)计算 **MyHourly**(时薪)。然后用标准 VBA 函数 FormatCurrency 将变量 **MyHourly** 转换为保留两位小数点的货币值，最后再把 FormatCurrency 函数的结果与“My Hourly Rate is(我的时薪是)”相连接。

本章回顾

变量是保存值或指向对象的名字。在函数、过程或事件内声明的变量是局部的，仅能在声明它的函数中使用，不能在函数外使用。在窗体或代码模块通用声明区中声明的变量能在声明它们的窗体或代码模块内任何地方使用。在代码模块中声明为“Public”的变量能在 VBA 工程的任何位置使用。在类模块中声明为“Public”的变量成为类模块的读/写属性。

本书从始至终都在使用变量。如果没有变量，一切都将是静态的，什么也不能改变——线总是从固定的一点画到固定的另一点，文本总是插入在同一点且内容总是相同的。

China BDN

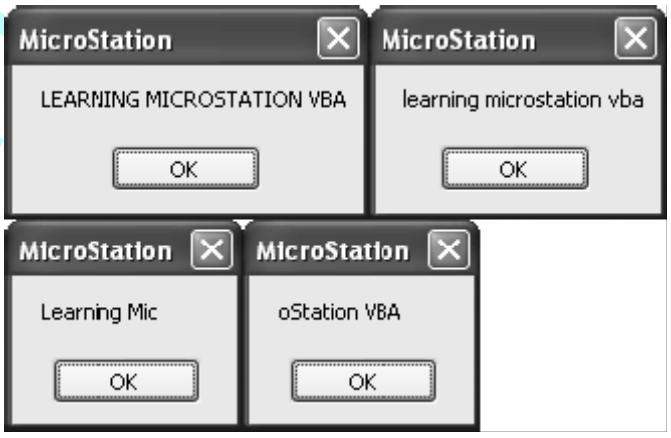


7.操作文本

我们每天都在与文本打交道。这本书就是由文本组成的：词、短语、句子、段落。操作文本的能力在编程中价值无限。

回忆一下处理文本的变量类型是一个 String(字符串)。

```
Sub TextWork01()
    Dim BookTitle As String
    BookTitle = "Learning MicroStation VBA"
    MsgBox UCase(BookTitle)
    MsgBox LCase(BookTitle)
    MsgBox Left(BookTitle, 12)
    MsgBox Right(BookTitle, 12)
End Sub
```



本例中，变量 **BookTitle** 被声明为一个字符串并被赋值为”Learning MicroStation VBA”。然后使用了四个不同的函数显示四个消息框，这四个函数都以 **BookTitle** 为参数。

VBA 字符串函数

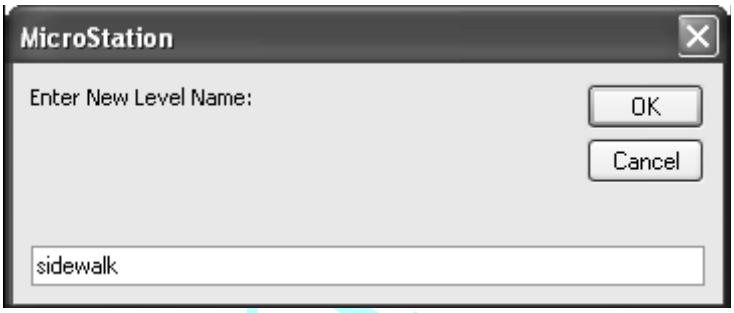
让我们逐个考察处理文本(字符串)的每一个 VBA 函数。

UCase(大写转换)

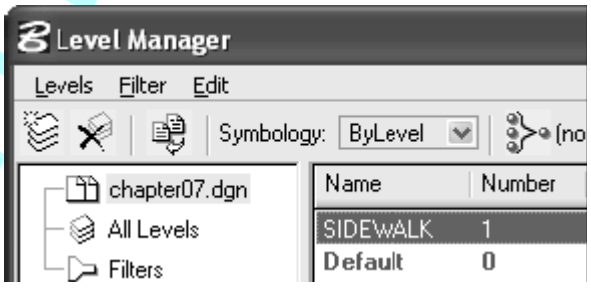
Function UCase(String)

UCase 函数把字符串转换为大写形式。

```
Sub TextWork02()
    Dim strNewLevel As String
    strNewLevel = InputBox("Enter New Level Name:")
    strNewLevel = UCase(strNewLevel)
    Application.ActiveDesignFile.AddNewLevel strNewLevel
End Sub
```



本例中用 InputBox 让用户输入一个新层名，然后将输入值转换为大写，再用大写名在当前设计文件中建立一个新层(使用了 AddNewLevel 方法)。

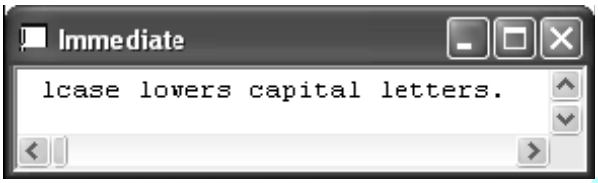


LCase(小写转换)

Function LCase(String)

L Case 函数把字符串转换为小写形式。

```
Sub TextWork03()
    Debug.Print LCase("LCase Lowers Capital Letters.")
End Sub
```



Debug.Print 用于在立即窗口中放置文本。经常用它显示一些有助于调试程序的文本。选 VBA 菜单 **View > Immediate Window(视图 > 立即窗口)**能打开立即窗口。

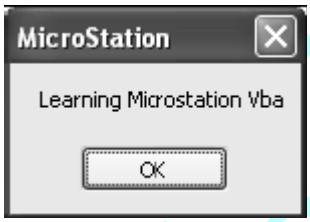
本例中，直接在函数里使用了文本，而不是把文本赋给一个变量。

StrConv(字符串转换)

```
Function StrConv(String, Conversion As VbStrConv, _
    [LocaleID As Long])
```

StrConv 用于将字符串转换成其参数所指定的格式。StrConv 最常用的常数是“vbProperCase”。

```
Sub TextWork04()
    Dim BookTitle As String
    BookTitle = "learning microstation vba"
    MsgBox StrConv(BookTitle, vbProperCase)
End Sub
```



本例使用 VbProperCase 常数把每个词的首字母转换成大写。

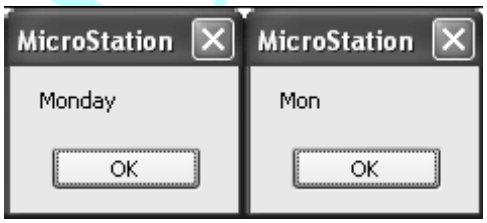
WeekDayName(星期名)、WeekDay(日期转星期)

```
Function WeekDayName(Weekday As Long, _
[Abbreviate As Boolean = False], _
[FirstDayOfWeek As VbDayOfWeek = vbUseSystemDayOfWeek]) _
As String
Function Weekday(date As Date, _
[FirstDayOfWeek As VbDayOfWeek = vbUseSystemDayOfWeek]) _
```

As Integer

一周中的每一天(周日至周六)都被赋予了一个数字，WeekDayName 函数取得这个数字后将其转换成星期名。

```
Sub TextWork05()
    Dim TodaysDate As Date
    Dim WeekDayNumber As Long
    TodaysDate = Now
    WeekDayNumber = Weekday(TodaysDate)
    MsgBox WeekdayName(WeekDayNumber)
    MsgBox WeekdayName(WeekDayNumber, True)
End Sub
```

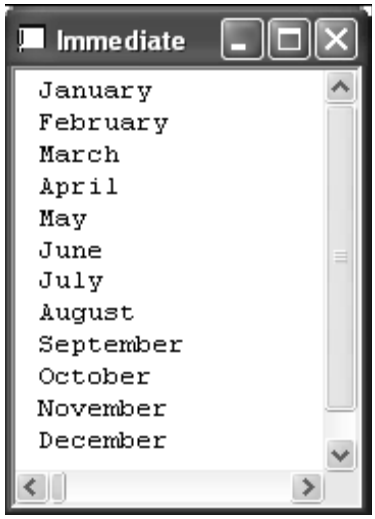


WeekDayNumber 的第一个参数是表示星期几的数字；第二个参数“Abbreviate(缩写)”是可选项，缺省值为 False。当把该参数赋值为 True 时，返回的星期名为缩写形式。

MonthName(月名)

```
Function MonthName(Month As Long, _
[Abbreviate As Boolean = False]) As String
```

MonthName 函数与 WeekdayName 函数相似，但正如其名字所隐含的那样，该函数返回的是月的名字而不是星期的名字。




```
Sub TextWork05B()
    Dim MonthNum As Long
    For MonthNum = 1 To 12
        Debug.Print MonthName(MonthNum)
    Next MonthNum
End Sub
```

China-PDN

LTrim(左截取)、RTrim(右截取)、Trim(两端截取)

Function LTrim(String)

Function RTrim(String)

Function Trim(String)

这三个 Trim 函数分别从字符串的开始、尾部和两端移去空格。

```
Sub TextWork06()
    Dim StringToTrim As String
    StringToTrim = " Trim Functions Trim Space Characters. "
    Debug.Print LTrim(StringToTrim)
    Debug.Print RTrim(StringToTrim)
    Debug.Print Trim(StringToTrim)
End Sub
```



StrComp(字符串比较)

Function StrComp(String1, String2, _

[Compare As VbCompareMethod = vbBinaryCompare])

比较两个文本片段的需求是普遍的。“Sidewalk”与“SIDEWALK”一样吗？并不总是一样。

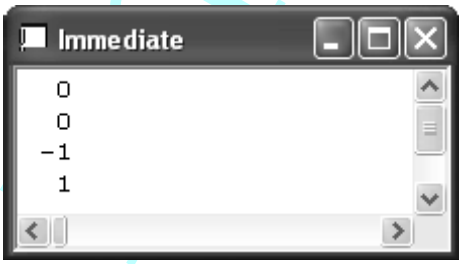
```
Sub TextWork07()
    Dim strNewLevel As String
    Dim lvlExistLevel As Level
    strNewLevel = InputBox("Enter New Level Name:")
    For Each lvlExistLevel In Application.ActiveDesignFile.Levels
        If StrComp(strNewLevel, lvlExistLevel.Name, _
                vbTextCompare) = 0 Then
```

```
                MsgBox "The level " & strNewLevel & " already exists."
                Exit Sub
            End If
        Next
        Application.ActiveDesignFile.AddNewLevel strNewLevel
End Sub
```

该过程向用户提问一个新的层名，把新输入的名字与现有的每一个层名进行比较，如果找到一个匹配就会显示一个消息框并退出本过程。

StrComp 能让我们指定如何比较文本。在上述例子中，当被比较的两个字符串相同(不分大小写)时，常数“vbTextCompare”返回一个零值(0)。使用比较选项“vbTextCompare”时，“Swalk” 和 “swalk”是一样的。

```
Sub TextWork08()
    Debug.Print StrComp("SWalk", "swalk", vbTextCompare)
    Debug.Print StrComp("swalk", "SWalk", vbTextCompare)
    Debug.Print StrComp("SWalk", "swalk", vbBinaryCompare)
    Debug.Print StrComp("swalk", "SWalk", vbBinaryCompare)
End Sub
```



StrComp 能让我们知道两个字符串是否相同，还能告诉我们字符串的前后顺序，这一点常用于对字符串按字母排序。

下面是 StrComp 的另一个例子，叫作冒泡排序。它对一个字符串数组按字母进行排序。这项技术有些高深，所以最好在学习了更多的 VBA 编程知识以后，再回过头来看这个例子。

```
Sub BubbleSort()
    Dim strNms(0 To 7) As String
    strNms(0) = "Jerry"
```

```
    strNms(1) = "Candice"
    strNms(2) = "Brandon"
    strNms(3) = "Kyle"
    strNms(4) = "Benjamin"
    strNms(5) = "Jacob"
    strNms(6) = "Nathan"
    strNms(7) = "Olivia"
    Dim MadeChange As Boolean
    Dim tmpName As String
    Dim I As Long
    MadeChange = True
    While MadeChange = True
        MadeChange = False
        For I = LBound(strNms) To UBound(strNms) - 1
            If StrComp(strNms(I), strNms(I + 1), _
                vbBinaryCompare) = 1 Then
                tmpName = strNms(I)
                strNms(I) = strNms(I + 1)
                strNms(I + 1) = tmpName
                MadeChange = True
            End If
        Next I
    Wend
    For I = LBound(strNms) To UBound(strNms)
        Debug.Print I & " " & strNms(I)
    Next I
End Sub
```

让我们把这个过程分成几段来讨论吧。

```
    Dim strNms(0 To 7) As String
    strNms(0) = "Jerry"
```


China BDN

```
strNms(1) = "Candice"
strNms(2) = "Brandon"
strNms(3) = "Kyle"
strNms(4) = "Benjamin"
strNms(5) = "Jacob"
strNms(6) = "Nathan"
strNms(7) = "Olivia"
```

我们做的第一件事是声明一个字符串数组并给数组中的每一个元素赋值。

```
Dim MadeChange As Boolean
Dim tmpName As String
Dim I As Long
MadeChange = True
```

现在，我们要设置程序中的排序部分。排序部分至少要被运行一次，所以我们把变量 MadeChange 设为 True，然后立即开始一个 While … Wend 循环。

```
While MadeChange = True
    MadeChange = False
    For I = LBound(strNms) To UBound(strNms) - 1
        If StrComp(strNms(I), strNms(I + 1), _
            vbBinaryCompare) = 1 Then
            tmpName = strNms(I)
            strNms(I) = strNms(I + 1)
            strNms(I + 1) = tmpName
            MadeChange = True
        End If
    Next I
Wend
```

上面这一段是程序的核心。我们一直检查 strNms 数组中的每一个值，比较当前值与其后的一个值。如果当前值按字母顺序比后面的大，就交换数组中的这两个元素并设置 MadeChange 变量为 True。仅当每个值都被检查过且不再发生

交换才继续下一段代码。

```
For I = LBound(strNms) To UBound(strNms)
    Debug.Print I & " " & strNms(I)
Next I
```

最后这一小段代码按其排序条件打印输出 strNms 变量数组中的元素。

学习 VBA 一段时间后再回到这个过程来看会感觉容易一些。采用 StrComp 函数的冒泡排序算法完成文本排序既容易又快速。

Len(取字串长度)

```
Function Len(Expression)
```

Len 函数告诉我们字符串中有多少个字符。

```
Sub TextWork09()
    Dim LevelName As String
    LevelName = InputBox("Enter new level name: “ & _
            “(Must be 8 characters)")
    If Len(LevelName) <> 8 Then
        MsgBox "The level name must be 8 characters. Try again."
    End If
End Sub
```

本例中，要求用户输入一个新的层名，还要求层名为八个字符长度。输入值后使用 Len 函数检查其长度，如果长度不是八个字符(<>意思是不等于)，提示用户重新输入。

Left(左取子串)

```
Function Left(String, Length As Long)
```

Left 函数返回从第一个字符(左侧)算起的指定长度的子字符串。

```
Sub TextWork10()
    Dim FilePath As String
    Dim FileDrive As String
    FilePath = Application.ActiveDesignFile.Path
```

Chinaa BDN

```
    FileDrive = Left(FilePath, 1)
    MsgBox "The current file is on the " & FileDrive & " drive."
End Sub
```

以上代码中先取得当前设计文件的路径，然后查找 FilePath 变量的第一个字符并把它放入 FileDrive 变量中。最后用一个消息框显示 FileDrive 变量以及一些其它的文本。

Right(右取子串)

```
Function Right(String, Length As Long)
```

您已经猜到它的功能了吧，Right 函数是 Left 函数的同类。它从字符串最后(右侧)取指定的字符数。

```
Sub TextWork11()
    Dim FileName As String
    FileName = Dir("C:\Program Files\Bentley\MicroStation\*.*", vbArchive)
    While FileName <> ""
        MsgBox FileName & " is a " & Right(FileName, 3) & " file."
        FileName = Dir
    Wend
End Sub
```

该过程在消息框中显示 C:\Program Files\Bentley\MicroStation 文件夹下的所有文件及其文件扩展名。由于文件较多，点击每个消息框中的 OK 按钮也需要花一段时间。

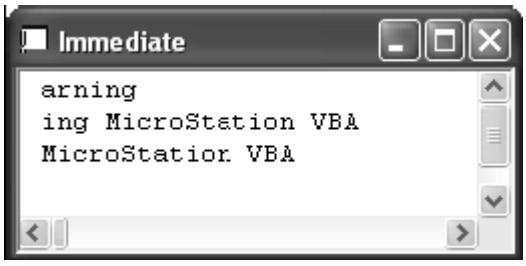
> **提示：**当一个程序正在执行的过程中，可以按住控制键(<Ctrl>)的同时按<Break>键来打断代码的执行。<Break>键在键盘的右上角。

Mid(取中串)

```
Function Mid(String, Start As Long, [Length])
```

Mid 函数要求指定一个字符串和要返回的字串的起始索引。还有一个指定要返回多少个字符的选项，如果不指定 Length 参数，Len 将返回 Start 指定的字符索引后的所有字符。

```
Sub TextWork12()
    Dim BookTitle As String
    BookTitle = "Learning MicroStation VBA"
    Debug.Print Mid(BookTitle, 3, 6)
    Debug.Print Mid(BookTitle, 6)
    Debug.Print Mid(BookTitle, InStr(1, BookTitle, " ") + 1)
End Sub
```



上例中使用了三次 Mid 函数，每次都有一点不同：第一次要求 Mid 从第三个字符开始并返回六个字符；第二次要求从第六个字符开始并返回其后的每个字符。注意这次根本没用 Length 参数；第三次没有把起始字符索引写死，而是用 InStr 函数在变量 BookTitle 中查找第一个空格的位置并加上一(1)作为起始，这样，实际上是从第一个空格后开始的。没有指定长度，将取得空格后的所有内容。

Replace(替换字串)

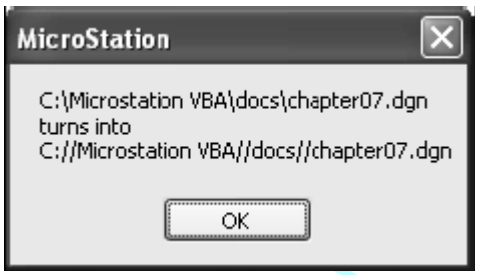
```
Function Replace(Expression As String, Find As String, _
Replace As String, _
[Start As Long = 1], [Count As Long = -1], _
[Compare As VbCompareMethod = vbBinaryCompare]) As String
```

Replace 函数要求提供一个字串、一个查找字符或字串、一个替换的字符或字串。

```
Sub TextWork13()
    Dim FilePath As String
    Dim FilePath2 As String
    FilePath = Application.ActiveDesignFile.FullName
    FilePath2 = Replace(FilePath, "\", "//")
    MsgBox FilePath & vbCr & "turns into" & vbCr & FilePath2
End Sub
```

China BDN

本例查找每个反斜杠然后用两个正斜杠代替之。



InStr(子串位置)

Function InStr([Start], _

[String1], [String2], _

[Compare As VbCompareMethod = vbBinaryCompare])

InStr 能指出一个字符或一组字符在一个字符串中出现的位置。例如，在字符串“ABCD”中找串“C”，InStr 将返回数字 3，因为“C”是“ABCD”中的第三个字符。如果所找的字符或字串不存在，InStr 将返回零值(0)。

```
Sub TextWork14()
    Dim FullName As String
    Dim FirstSpace As Long
    Dim FirstName As String
    FullName = InputBox("Enter your full name.")
    FirstSpace = InStr(1, FullName, " ")
    FirstName = Left(FullName, FirstSpace - 1)
    MsgBox "Your first name is " & FirstName & "."
End Sub
```

这里是另一个使用 InStr 函数的简单例子。该例子要求输入用户全名，在输入的名字中查找第一个空格，然后取得 **FullName** 中从开始到 **FisrtSpace(第一个空格)**之前的所有字符。

```
Sub TextWork15()
    Dim FilePath As String
    Dim FirstFolderPath As String
    Dim SecondBackSlash As Long
    FilePath = ActiveDesignFile.FullName
```

```
        SecondBackSlash = InStr(4, FilePath, "\")
        FirstFolderPath = Left(FilePath, SecondBackSlash)
        MsgBox FilePath & vbCr & FirstFolderPath
    End Sub
```

使用 InStr 时经常从第一个字符开始查起，这就是为什么第一个参数是数字 1 的原因。然而，本例中从第四个字符开始查找，因为我们知道文件路径中第三个字符很可能就是一个反斜杠。TextWork15 的目的是得到当前 DGN 文件的根文件夹路径。

再给一个例子：

```
Sub TextWork16()
    Dim TextElem As TextElement
    Dim MyMod As ModelReference
    Set MyMod = Application.ActiveModelReference
    Dim MyElems As ElementEnumerator
    Set MyElems = MyMod.GetSelectedElements
    Dim MyElem As Element
    While MyElems.MoveNext
        Set MyElem = MyElems.Current
        Select Case MyElem.Type
            Case msdElementTypeText
            Set TextElem = MyElem
            If InStr(1, TextElem.Text, "[BY]") > 0 Then
                TextElem.Text = Replace(TextElem.Text, "[BY]", "JKW")
                TextElem.Rewrite
            End If
        End Select
    Wend
End Sub
```

本例中在当前文件查找每一个选中的元素。如果找到一个选择的文本元素，

China BDN

就用 InStr 来看这个文本元素中是否含有一个特殊的字符串，如果确实含有，InStr 返回一个大于 0 的数字。当知道文本元素中含有搜索的字符串时，用 Replace 函数将“[BY]”替换为“JKW”。

这是在文件中执行“搜索并替换”操作的一种方法。该功能在这样的情况下会用到：您需要在文件中放入承包商的名字，但此时承包商还没有最终确定。先用一个标记，如[CONTRACTOR]，放入文件中，待承包商名称确定后用真正的名称替换这个标记。

InStrRev(反向子串位置)

```
Function InStrRev(StringCheck As String, _
StringMatch As String, _
[Start As Long = -1], _
[Compare As VbCompareMethod = vbBinaryCompare]) As Long
```

正如函数名表达的那样，InstrRev 先从字符串尾部而不是从开头查找。它是 Instr 的反函数。下面是它的一种用法：

```
Sub TextWork17()
    Dim FilePath As String
    Dim FolderName As String
    FilePath = ActiveDesignFile.Path
    FolderName = Mid(FilePath, InStrRev(FilePath, "\") + 1)
    MsgBox "The current file is in the " & FolderName & "folder."
End Sub
```

上面的例子取得当前文件的路径，然后用 Mid 函数中嵌套的 InStrRev 函数从反方向取得第一个反斜杠的位置。因为不想显示文件夹名字前的反斜杠，所以在 Mid 函数中第二个参数加 1 取得反斜杠后的字符。

Split(分割)和 Join(联接)

```
Function Split(Expression As String, [Delimiter], _
[Limit As Long = -1], _
[Compare As VbCompareMethod = vbBinaryCompare])
```

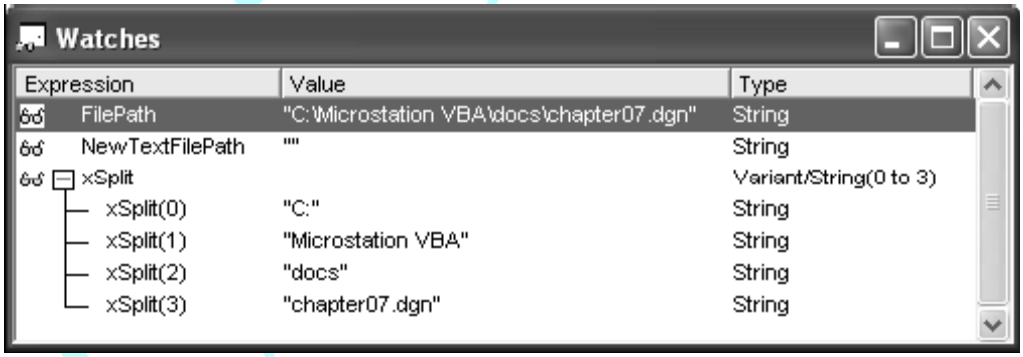
Function Join(SourceArray, [Delimiter]) As String

Split 把一个字符串分割成一个文本数组。

Join 把一个文本数组联接成一个字符串。这两个函数都需要指定一个分隔符才能正常工作。让我们看一个使用 Split 和 Join 的例子。

```
Sub TextWork18()
    Dim FilePath As String
    Dim NewTextFilePath As String
    Dim xSplit As Variant
    FilePath = ActiveDesignFile.FullName
    xSplit = Split(FilePath, "\")
    xSplit(UBound(xSplit)) = xSplit(UBound(xSplit)) & ".extract"
    NewTextFilePath = Join(xSplit, "\")
    Open NewTextFilePath For Output As #1
    Print #1, FilePath
    Close #1
End Sub
```

本例要建立一个与当前设计文件在同一文件夹下的扩展名为“.extract”的 ASCII 文本文件。用 Split 函数把当前设计文件的路径分割成一个数组，分割时以反斜杠为分隔符。当单步进入代码中时，分割后的情况如下所示：

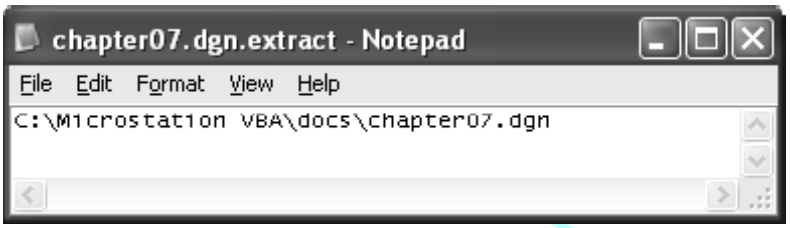


变量 FilePath 包含当前设计文件的路径。变量 xSplit 是以反斜杠(\)为分隔符从 FilePath 中分割出的一个数组。取出数组中的最后一个元素(用 UBound 函数)并加上新的文件扩展名“.extract”。

下一步，以反斜杠(\)为分隔符把数组联接成变量 NewTextFilePath。

最后，以 NewTextFilePath 变量为文件名建立一个 ASCII 文本文件。在这个新的文件中打印输出 FilePath 变量的内容。

右图为该文件在记事本中打开后的样子。



Asc(字符到 ASCII 码)和 Chr(ASCII 码到字符)

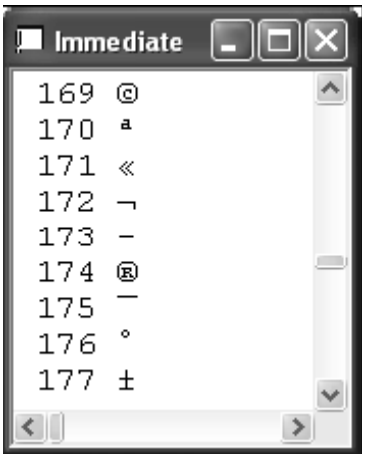
Function Asc(String As String) As Integer

Function Chr(CharCode As Long)

计算机键盘上的每个字符都有一个 ASCII 码与之关联。大写的“A”有其唯一的数码，与小写“a”是不同的。有些不在键盘上的字符也有其 ASCII 码。用 Asc 函数取得一个字符的 ASCII 码。如果知道字符的 ASCII 码，就能用 Chr 函数得到对应的字符。

```
Sub TextWork19()
    Dim strCharacter As String
    For I = 0 To 255
    Debug.Print I & vbTab & Chr(I)
    Next I
End Sub
```

这段代码在调试(立即)窗口中打印从 0 到 255 的每个 ASCII 字符。我们为什么想要这样做呢？因为有些不在键盘上的 ASCII 字符早晚也有用。例如，字符码 169 是版权符号，字符码 174 是注册符号，字符 176 是度符号，字符 177 是用于几何公差的加/减符号。



既然知道了字符 169 是版权符号，那就可以在消息框中使用它：

```
Sub TextWork20()
    Dim strCopyrightNotice As String
```

```
    strCopyrightNotice = "Learning MicroStation VBA " & _
            Chr(169) & "2005"
    MsgBox strCopyrightNotice
End Sub
```



Asc 函数做与 Chr 函数相反的动作，从一个字符得到其对应的 ASCII 码。在建立图形用户界面(GUI)时可能会用到这个功能。

```
Private Sub TextBox1_KeyPress(ByVal KeyAscii As _
                              MSForms.ReturnInteger)
    Select Case KeyAscii
        Case Asc("0") To Asc("9")
        Case Else
            KeyAscii = 0
    End Select
End Sub
```

上面的代码是一个文本框的按键(KeyPress)事件。按键事件给出用户在文本框中键入的字符的 ASCII 码。在这个例子中，如果 **KeyAscii** 属性含有数字 0 至 9，则程序什么事也不做；否则给 KeyAscii 属性赋值 0。最终的结果是：仅有数字 0 至 9 才能被输入到这个文本框中。

FormatCurrency(格式化货币)

```
Function FormatCurrency(Expression, _
[NumDigitsAfterDecimal As Long = -1], _
[IncludeLeadingDigit As VbTriState = vbUseDefault], _
[UseParensForNegativeNumbers As VbTriState = vbUseDefault], _
[GroupDigits As VbTriState = vbUseDefault]) As String
```

FormatCurrency 函数要求一个数字或字串，尔后以货币形式显示。在某些国家，如美国，该函数会在货币数前加上一个美元符号。该函数的其它参数有使数字以逗号分组等功能。

China BDN

```
Sub TextWork21()
    Dim MySalary As Double
    Dim MySalary2 As Double
    MySalary = 123456.78
    MySalary2 = 0.1234
    MsgBox FormatCurrency(MySalary, 2, vbFalse, vbFalse, vbTrue)
    MsgBox FormatCurrency(MySalary, 0, vbFalse, vbFalse, vbTrue)
    MsgBox FormatCurrency(MySalary2, 2, vbFalse, vbFalse, vbTrue)
    MsgBox FormatCurrency(MySalary2, 2, vbTrue, vbFalse, vbTrue)
End Sub
```



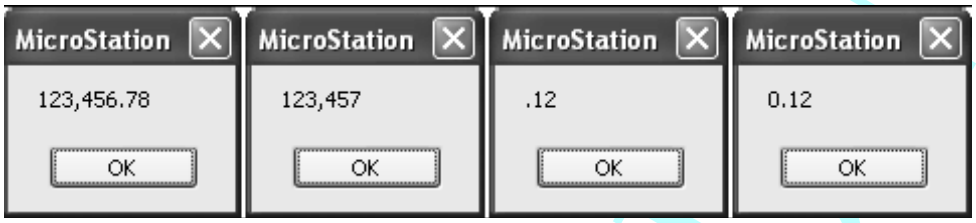
FormatNumber(格式化数字)

```
Function FormatNumber(Expression, _
[NumDigitsAfterDecimal As Long = -1], _
[IncludeLeadingDigit As VbTriState = vbUseDefault], _
[UseParensForNegativeNumbers As VbTriState = vbUseDefault], _
[GroupDigits As VbTriState = vbUseDefault]) As String
```

FormatNumber 函数看起来与 FormatCurrency 一样。它们的主要不同在于：FormatCurrency 在数字前放置一个货币字符，而 FormatNumber 仅返回格式化后的数字。

```
Sub TextWork22()
    Dim MySalary As Double
    Dim MySalary2 As Double
    MySalary = 123456.78
    MySalary2 = 0.1234
    MsgBox FormatNumber(MySalary, 2, vbFalse, vbFalse, vbTrue)
```

```
    MsgBox FormatNumber(MySalary, 0, vbFalse, vbFalse, vbTrue)
    MsgBox FormatNumber(MySalary2, 2, vbFalse, vbFalse, vbTrue)
    MsgBox FormatNumber(MySalary2, 2, vbTrue, vbFalse, vbTrue)
End Sub
```

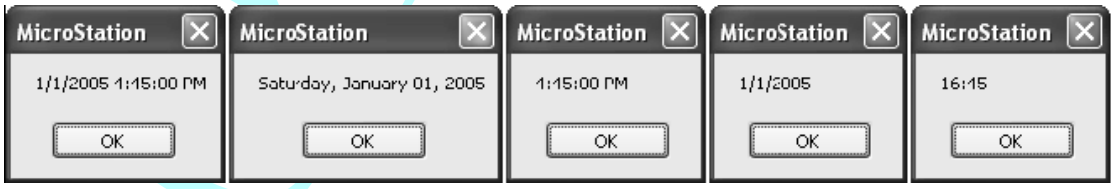


FormatDateTime(格式化日期)

```
Function FormatDateTime(Expression, _
[NamedFormat As VbDateTimeFormat = vbGeneralDate]) _
As String
```

FormatDateTime 格式化一个日期/时间。下面的例子示出了选项和结果：

```
Sub TextWork23()
    Dim DateToFormat As Date
    DateToFormat = "1/1/2005 4:45 PM"
    MsgBox FormatDateTime(DateToFormat, vbGeneralDate)
    MsgBox FormatDateTime(DateToFormat, vbLongDate)
    MsgBox FormatDateTime(DateToFormat, vbLongTime)
    MsgBox FormatDateTime(DateToFormat, vbShortDate)
    MsgBox FormatDateTime(DateToFormat, vbShortTime)
End Sub
```



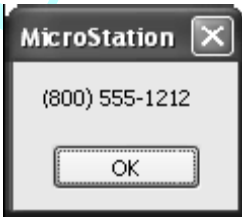
Format(格式化)

```
Function Format(Expression, [Format], _
[FirstDayOfWeek As VbDayOfWeek = vbSunday], _
```

[FirstWeekOfYear As VbFirstWeekOfYear = vbFirstJan1])

我们已经有了对特定类型数据进行格式化的例子：FormatCurrency、FormatNumber、FormatDateTime，这些函数在标准格式化条件下工作得很好。然而，VBA 并没有提供一个 FormatPhoneNumber(格式化电话号码)的函数。那如何将十位数字格式化成一个电话号码呢？

```
Sub TextWork24()
    Dim MyPhone As String
    MyPhone = "8005551212"
    MsgBox Format(MyPhone, "(###) ###-####")
End Sub
```



Format 函数中的格式化参数具有非常大的灵活性，参见 VBA 帮助文件取得更多的格式化选项。

&(连接符)

用和符号(&)连接字符串。本书中大量使用了这个连字符把多个字符串组合成一个。

VbCr(回车符)

在字符串中能用到几个常数，如 vbCr 就是回车符。它类似于在键盘上按**<Enter>**键。观察以前用到 vbCr 常数的例子及其生成的结果。

VbTab(制表符)

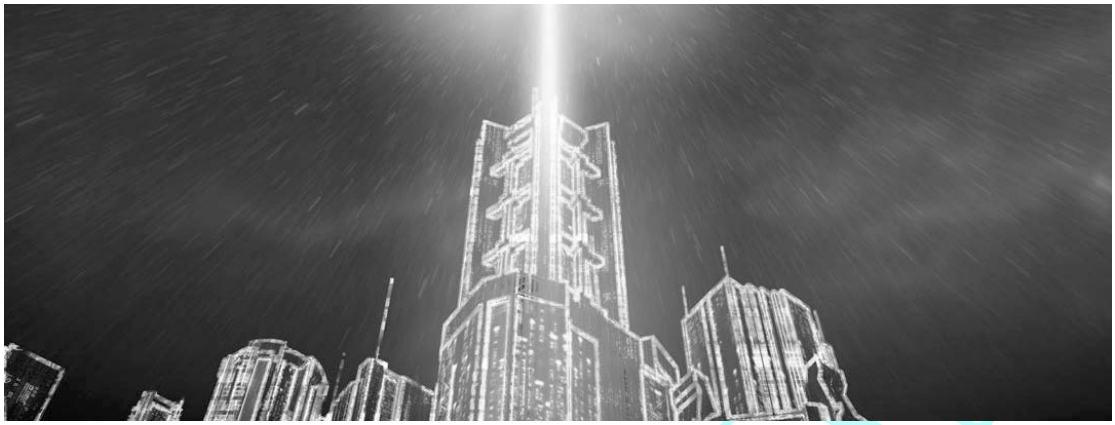
使用 vbTab 常数类似于用户按键盘上的**<Tab>**键。

本章回顾

字符串是指文本、字母、数字和其它字符组合成单个文本块。本章集中于讲述如何操作这些字符串。您学了大写转换、小写转换、从左边取或从右边取字符、分割字符串、连接字符串、格式化字符串以及其它一些操作。

花些时间分析每个函数所带的例子。记住，按**<F8>**键能随时单步运行代码。

下一章讲述如何处理数字。



8.操作数字

数百甚至数千年来，世界上最伟大的数学家们都在尝试着计算 Pi 的精确值。当今天再问到这个值时，我们会轻易地说出 3.14159 或接近这个值的一个数。$A^2 + B^2 = C^2$。这个公式对于今天在我们学校操场上玩耍的孩子们来说再熟悉不过了，然后，在并不很久以前，直角三角形两直边的平方和等于其斜边的平方这一公式还并不为人所共知。

通常，进行软件开发时需要大量的逻辑处理，再加上对 MicroStation 的处理后使得对数字操作的需求呈指数级增长。一条线究竟是什么？两点之间的最短距离。每个点都由三个数值组成：X、Y 和 Z。要用 VBA 在这条线的中点画一个圆，就需要计算线的中点。怎样计算呢？需要找一个数学老师的电话吗？不需要这样，我的朋友。

数字函数

VBA 给操作数字带来了方便性。虽然它没有为我们做所有的工作，但我们用它能很舒服地完成工作。

加

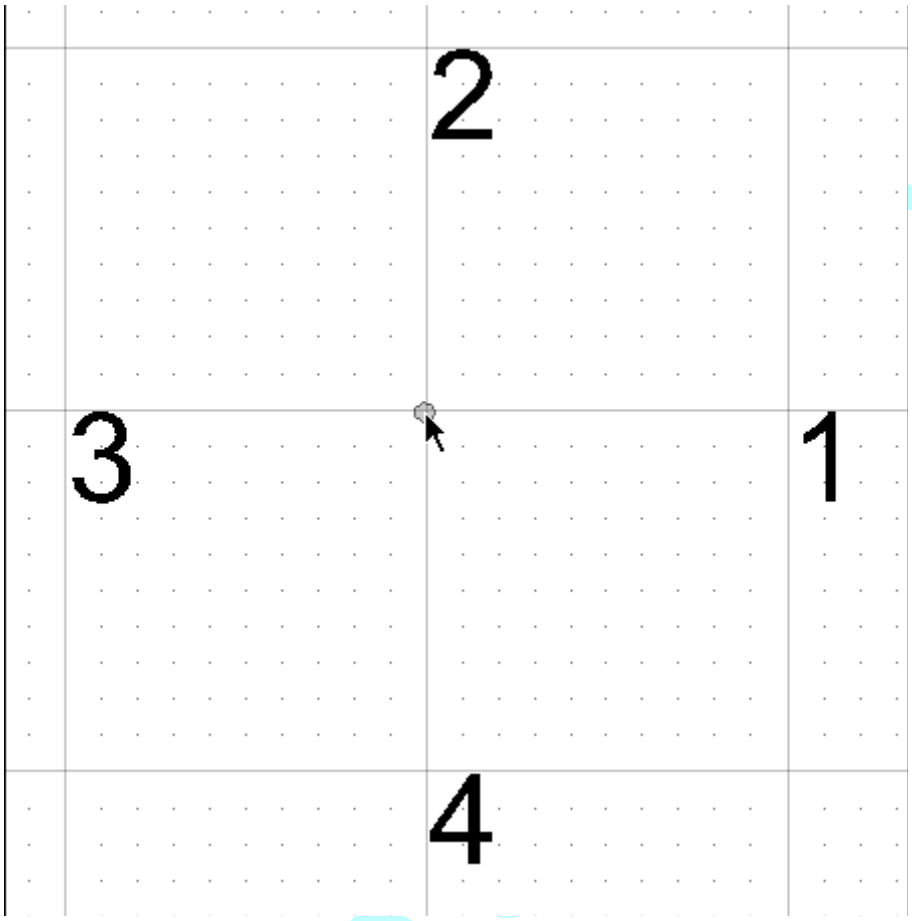
1 + 1 = 2。许多、许多年以前我们就学了。VBA 中用正号(＋)来加数字。请看：

```
Sub TestAdditionSubtraction()
    Dim SelPt As Point3d
    Dim CenPt As Point3d
```

```
Dim CadMsg As CadInputMessage
Dim TextElem As TextElement
Dim RotMatrix As Matrix3d
Set CadMsg = Application.CadInputQueue.GetInput
Do While True
    Select Case CadMsg.InputType
        Case msdCadInputTypeDataPoint
            SelPt = CadMsg.Point
            Exit Do
    End Select
Loop
CenPt = SelPt
CenPt.X = CenPt.X + 1
Set TextElem = Application.CreateTextElement1(Nothing, "1", _
                CenPt, RotMatrix)
ActiveModelReference.AddElement TextElem
CenPt = SelPt
CenPt.Y = CenPt.Y + 1
Set TextElem = Application.CreateTextElement1(Nothing, "2", _
                CenPt, RotMatrix)
ActiveModelReference.AddElement TextElem
CenPt = SelPt
CenPt.X = CenPt.X - 1
Set TextElem = Application.CreateTextElement1(Nothing, "3", _
                CenPt, RotMatrix)
ActiveModelReference.AddElement TextElem
CenPt = SelPt
CenPt.Y = CenPt.Y - 1
Set TextElem = Application.CreateTextElement1(Nothing, "4", _
                CenPt, RotMatrix)
```

China BDN

```
    ActiveModelReference.AddElement TextElem
End Sub
```



我们让用户在 MicroStation 中选一点，然后用选取的点作为加入模型的文本元素的插入基准点。在选取点的 X 分量上加 1 得到文本“1”的位置；在选取点的 Y 分量上加 1 得到文本“2”的位置；点 3 和点 4 要求从 X、Y 分量分别减 1 得到。

减

10 – 3 = 7。VBA 中用负号(–)来减数值，就像 TestAdditionSubtraction 过程中所举的例子那样。

乘

2 X 6 = 12。VBA 中用星号(*)表示乘。行首所写的形式在数学书中是正确的，但在 VBA 中要写成：2 * 6 = 12。

```
    Sub TestMultiplication()
        Dim DistanceInInches As Double
```

```
    Dim DistanceInMM As Double
    DistanceInInches = CDbl(InputBox("Enter distance in inches:"))
    DistanceInMM = DistanceInInches * 25.4
    MsgBox DistanceInInches & " is equal to " & _
            DistanceInMM & " Millimeters."
End Sub
```

把输入的值乘 25.4 就把输入值由英寸转换为了毫米。

除

在 VBA 中有两种除。不，不是长除和短除。第一种除返回一个非常精确的数。当需要精确值(通常都是这样)时，使用正斜杠(/)符号，如：5 / 2 = 2.5

```
Function ToMiles(DistanceInFeet as Double) As Double
    ToMiles = DistanceInFeet / 5280
End Function
```

函数 ToMiles 输入以英尺为单位的距离，返回以英里为单位的距离。实际上，返回的值是小数的英里。

另一种除使用反斜杠符号(\)。这种除返回整数而不是小数。5 \ 2 = 2。

平方与指数

还记得 $A^2 + B^2 = C^2$ 吗？在 A、B 和 C 右上角显示的小 2 数字表示 A、B 和 C 的平方。

在 VBA 中，这个表达式应写成：

A^2 + B^2 = C^2。脱字符号表示指数。如果要求一个数的立方(3 次方幂)，应该用 D^3。

```
Function GetCircleArea(CircleRadius As Double) As Double
    Dim Pi As Double
    Pi = Atn(1) * 4
    GetCircleArea = Pi * CircleRadius ^ 2
End Function
```

平方根

用 Sqr 函数求一个数的平方根。下面是其一种用法：

```
Sub GetLineLength()
    Dim SelElem As Element
    Dim LineElem As LineElement
    Dim SelElems As ElementEnumerator
    Set SelElems = ActiveModelReference.GetSelectedElements
    While SelElems.MoveNext
        Set SelElem = SelElems.Current
        Select Case SelElem.Type
            Case msdElementTypeLine
                Set LineElem = SelElem
                Dim StPt As Point3d
                Dim EnPt As Point3d
                StPt = LineElem.StartPoint
                EnPt = LineElem.EndPoint
                LineLength = Sqr((StPt.X - EnPt.X) ^ 2 + _
                        (StPt.Y - EnPt.Y) ^ 2)
                MsgBox "Line found with length of " & LineLength
        End Select
    Wend
End Sub
```

本例使用了勾股定理(也叫毕达哥拉斯定理)。先取得选中的直线在 X 和 Y 方向上的变化量，再把这些值平方后加起来，最后求得总和的平方根。

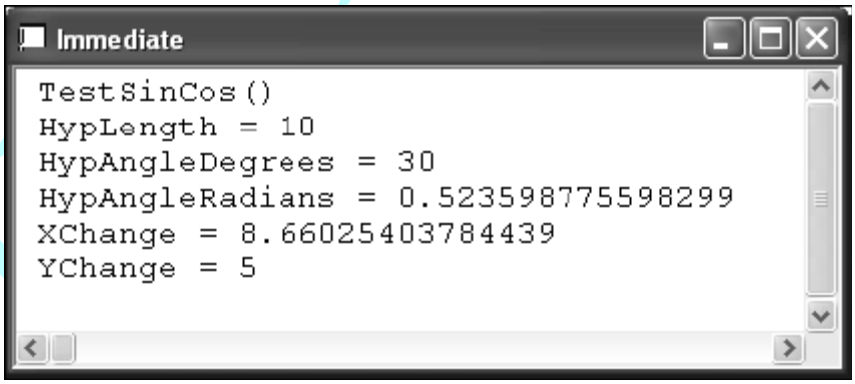
正弦、余弦、正切

Sin、Cos 和 Tan 函数要求一个弧度角。

```
Sub TestSinCos()
    Dim XChange As Double
```

```
    Dim YChange As Double
    Dim Pi As Double
    Dim HypLength As Double
    Dim HypAngleDegrees As Double
    Dim HypAngleRadians As Double
    Pi = Atn(1) * 4
    HypLength = CDbl(InputBox("Enter Hypotenuse Length:"))
    HypAngleDegrees = CDbl(InputBox("Enter Angle:"))
    HypAngleRadians = HypAngleDegrees * Pi / 180
    YChange = HypLength * Sin(HypAngleRadians)
    XChange = HypLength * Cos(HypAngleRadians)
    Debug.Print "TestSinCos()"
    Debug.Print "HypLength = " & HypLength
    Debug.Print "HypAngleDegrees = " & HypAngleDegrees
    Debug.Print "HypAngleRadians = " & HypAngleRadians
    Debug.Print "XChange = " & XChange
    Debug.Print "YChange = " & YChange
End Sub
```

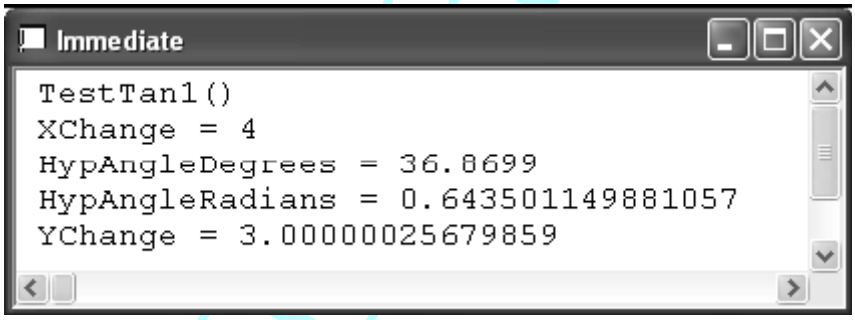
为了计算 X 和 Y 方向上的变化量，我们需要斜边长度和一个角度以及这个角度的正弦和余弦。在取得用户输入后，把输入的角从度转换为弧度，然后用弧度角的 Sin 值和 Cos 值求得 X 和 Y 方向上的变化量。



现在让我们使用 Tan 函数。第一个例子假定你知道沿 X 轴的三角形的边。

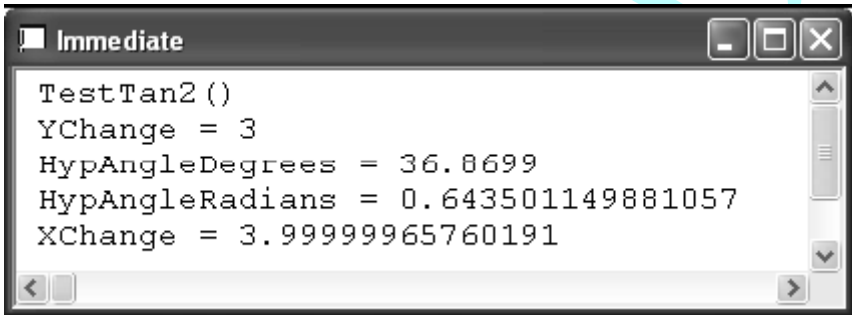
```
Sub TestTan1()
    Dim XChange As Double
```

```
    Dim YChange As Double
    Dim Pi As Double
    Dim HypAngleDegrees As Double
    Dim HypAngleRadians As Double
    Pi = Atn(1) * 4
    XChange = CDbl(InputBox("Enter X Side Length:"))
    HypAngleDegrees = CDbl(InputBox("Enter Angle:"))
    HypAngleRadians = HypAngleDegrees * Pi / 180
    YChange = Tan(HypAngleRadians) * XChange
    Debug.Print "TestTan1()"
    Debug.Print "XChange = " & XChange
    Debug.Print "HypAngleDegrees = " & HypAngleDegrees
    Debug.Print "HypAngleRadians = " & HypAngleRadians
    Debug.Print "YChange = " & YChange
End Sub
```

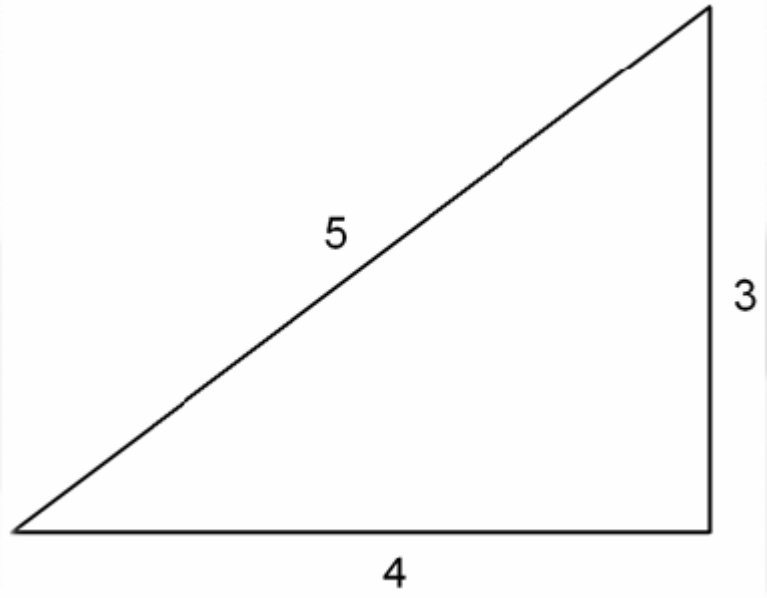



```
Sub TestTan2()
    Dim XChange As Double
    Dim YChange As Double
    Dim Pi As Double
    Dim HypAngleDegrees As Double
    Dim HypAngleRadians As Double
    Pi = Atn(1) * 4
    YChange = CDbl(InputBox("Enter Y Side Length:"))
    HypAngleDegrees = CDbl(InputBox("Enter Angle:"))
```

```
    HypAngleRadians = HypAngleDegrees * Pi / 180
    XChange = YChange / Tan(HypAngleRadians)
    Debug.Print "TestTan2()"
    Debug.Print "YChange = " & YChange
    Debug.Print "HypAngleDegrees = " & HypAngleDegrees
    Debug.Print "HypAngleRadians = " & HypAngleRadians
    Debug.Print "XChange = " & XChange
End Sub
```



以上例子中所用的值构成了直角三角形。

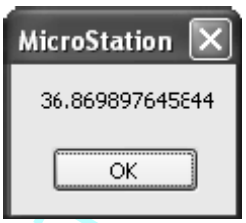


当我们编写代码时，犯点小错误再正常不过了。人们称之为“bug(臭虫)”但我们把它叫做“创造性编程”。最终的结果是一样的：代码不能工作。所以，用可预知结果的输入值来测试计算对消除程序中的错误是很有帮助的。

反正切

当知道有关的角度时，Sin、Cos 和 Tan 能求出有用的值来。如果不知道角度，可以用 Atn(反正切)求得角度。

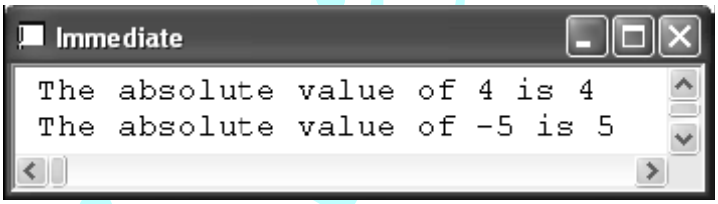
```
Sub TestATan()
    Dim Pi As Double
    Dim AngleDegrees As Double
    Dim AngleRadians As Double
    Pi = Atn(1) * 4
    AngleRadians = Atn(3 / 4)
    AngleDegrees = AngleRadians / Pi * 180
    MsgBox AngleDegrees
End Sub
```



绝对值

Abs 函数求得给定数的绝对值。

```
Sub TestAbs()
    Debug.Print "The absolute value of 4 is " & Abs(4)
    Debug.Print "The absolute value of -5 is " & Abs(-5)
End Sub
```



转换为整数、长整数、双精度数和值

我们讨论过声明变量的类型。VBA 能将值从一种类型转换到另一种。

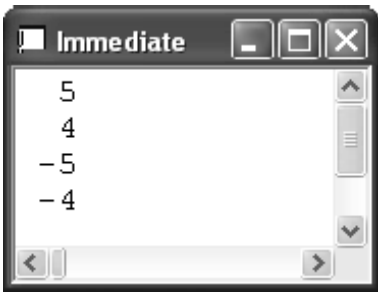
最常用的一个转换是从字符串转换到数字。

```
YChange = CDbl(InputBox("Enter Y Side Length:"))
```

以上代码中用 CDbl 函数将输入框的结果转换为一个双精度数。

```
Sub TestCInt()
    Debug.Print CInt(4.56)
    Debug.Print CInt(4.23)
    Debug.Print CInt(-4.56)
    Debug.Print CInt(-4.23)
```

```
End Sub
```

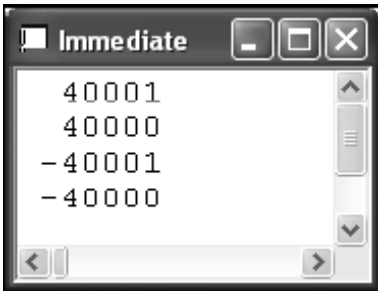


当从双精度数向整数转换时，对数的小数部分需要做一些工作，明白它是如何工作的很重要。CInt 通过圆整数字得到一个整数。仔细看一下“TestCInt”中的代码以及在立即窗口中所示的结果。

CLng

CLng 函数除了是把所提供的值转换为长整型外与 CInt 函数相似。您可能会问：“假如 CLng 和 CInt 完成同样的工作，那究竟应该用哪一个呢？”这是一个好问题。请记住，长整数比整数大得多！用 CInt 转换像 40,000.123 这样的数将会产生溢出错误。CInt 和 CLng 常用于对变量的赋值，所以如果对声明为整数的变量赋值就应该用 CInt，对声明为长整数的变量赋值就应该用 CLng。

```
Sub TestCLng()
    Debug.Print CLng(40000.56)
    Debug.Print CLng(40000.23)
    Debug.Print CLng(-40000.56)
    Debug.Print CLng(-40000.23)
End Sub
```

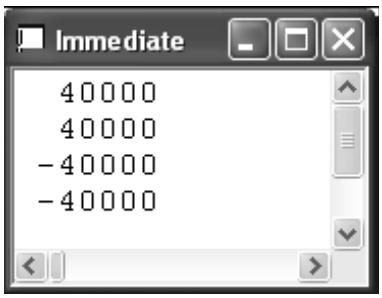


Fix

Fix 函数的作用看起来很像 CInt 或 CLng 函数，它返回一个数字的整数部分，然而，它又有些不同。让我们仔细观察如下代码的运行结果。

```
Sub TestFix()
    Debug.Print Fix(40000.56)
    Debug.Print Fix(40000.23)
    Debug.Print Fix(-40000.56)
    Debug.Print Fix(-40000.23)
End Sub
```

Fix 函数只是简单地甩掉数字的小数部分，它不做任何圆整。Fix 能操作整



数和长整数。

CDbl

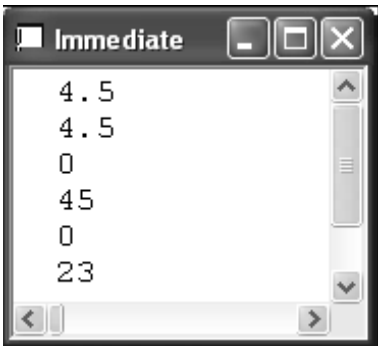
CDbl 把提供的参数转换为一个双精度数。

```
Sub TestDouble()
    Dim LineLength As Double
    LineLength = CDbl(InputBox("Enter the line length:"))
End Sub
```

Val

CInt、CLng 和 CDbl 在参数为数字时工作得很好，这其中的任何一个函数都能对 3.14159 进行操作。然而，如果给它们传递参数 2.5”将会弹出一个错误。Val 函数能给出参数中的数字值。弄明白该函数如何工作的最好方法是运行一些代码来观察其结果。

```
Sub TestVal()
    Debug.Print Val("4.5""")
    Debug.Print Val("4.5 inches")
    Debug.Print Val("$5,000")
    Debug.Print Val("45 degrees")
    Debug.Print Val("Approx. 5280 feet")
    Debug.Print Val("23 feet 12 inches")
End Sub
```



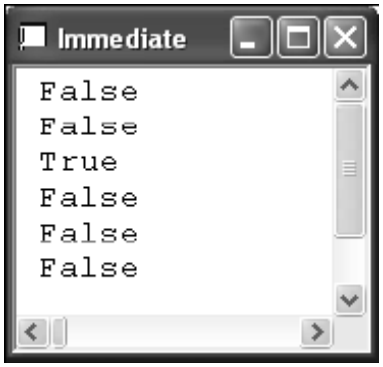
注意，当提供给 Val 函数的参数以数字值开头时，Val 返回所有的数字字符直到它遇到一个非数字字符为止。

IsNumeric

我们刚刚复习过的许多函数都是返回数字值，IsNumber 则返回一个布尔值

(True 或 False)。它考察其参数并确定参数是否是数字。

```
Sub TestIsNumeric()
    Debug.Print IsNumeric("4.5""")
    Debug.Print IsNumeric("4.5 inches")
    Debug.Print IsNumeric("$5,000")
    Debug.Print IsNumeric("45 degrees")
    Debug.Print IsNumeric("Approx. 5280 feet")
    Debug.Print IsNumeric("23 feet 12 inches")
End Sub
```

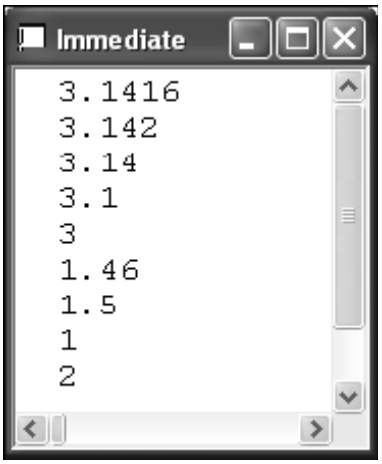


IsNumeric 考察整个参数并确定其是否为数字。如果参数的任何部分不是数字，怎返回一个假(False)。注意，美元符($)算作数字符号。

Round

CInt 和 CLng 把小数圆整为整数。Round 函数能让我们指定小数点后保留的位数。请看：

```
Sub TestRound()
    Debug.Print Round(3.14159, 4)
    Debug.Print Round(3.14159, 3)
    Debug.Print Round(3.14159, 2)
    Debug.Print Round(3.14159, 1)
    Debug.Print Round(3.14159, 0)
    Debug.Print Round(1.455, 2)
    Debug.Print Round(1.455, 1)
    Debug.Print Round(1.4, 0)
    Debug.Print Round(1.5, 0)
```



China RDN

```
End Sub
```

Mod—求余

Mod 函数求出两个数字的余数，但它的使用方法与其它大多数函数非常不同。大多数函数先调用函数，然后提供用逗号(,)分割的参数。而 Mod 函数则是先给出分子，再调用 Mod，最后给出分母。

```
Sub TestMod1()
    Debug.Print 5 Mod 2
    Debug.Print 7 Mod 3
    Debug.Print 23 Mod 7
    Debug.Print 280 Mod 2
End Sub
```



Sgn—求数的正负号

一个数字到底是正数还是负数？或两者都不是？Sgn 函数根据所给的值是负数、零或正数而返回-1、0 或 1。

```
Sub TestSgn()
    Debug.Print Sgn(-4.5)
    Debug.Print Sgn(0)
    Debug.Print Sgn(4.5)
End Sub
```



Rnd 与 Randomize

有时需要生成随机数。下面的例子演示了怎样在一个小的数与一个大的数之间生成随机数。程序运行的结果是在(25,25)到(50,50)之间，由 300 个随机点组成的云图。

```
Sub TestRnd()
    Dim I As Long
    Dim Lower As Long
    Dim Higher As Long
    Dim PointCen(0 To 1) As Point3d
```

```
    Dim PointElem As PointStringElement
    Lower = 25
    Higher = 50
    Randomize
    For I = 1 To 300
        PointCen(0).X = Round((Higher - Lower + 1) * Rnd(1), 2)
        PointCen(0).Y = Round((Higher - Lower + 1) * Rnd(1), 2)
        PointCen(1).X = PointCen(0).X
        PointCen(1).Y = PointCen(0).Y
        Set PointElem = _
            Application.CreatePointStringElement1(Nothing, _
            PointCen, True)
        ActiveModelReference.AddElement PointElem
    Next I
End Sub
```

操作符的优先级

2 + 5 * 8 / 12 + 13 = ?

(2 + 5) * 8 / (12 + 13) = ?

2 + (5 * 8 / (12 + 13)) = ?

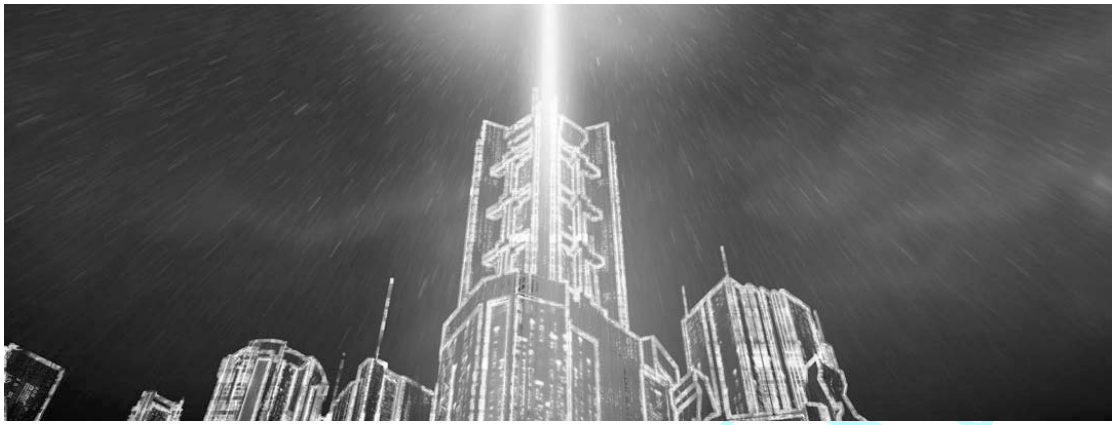
这每一个表达式都产生一个不同的结果。数字都是一样的，操作符也都是一样的，但结果却不同。

理解数字操作符的优先级非常重要。先乘除后加减。如果仅应用该规则还有问题的话，为了使 VBA 更清晰地计算表达式，在希望优先运算的部分两端添加括号。

本章回顾

许多软件开发者工作了很长一段时间可能都不会用到数学函数。然而，当我们在 MicroStation 中编写程序时却总是要用到数字函数。我们即要加减乘除，还要用其它的函数定位 MicroStation 中的元素，或者计算长度、角度等等。

China BDN



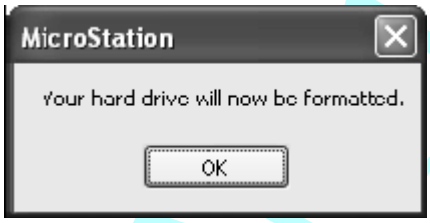
9. 标准 VBA 调用

在介绍各种概念的时候，我们使用了大量的标准 VBA 调用而没有讨论它们。现在，让我们逐一解读它们。同时，这些 VBA 调用还能在其它支持 VBA 的应用(如 Microsoft Excel)中使用。

消息框

我们用消息框来显示一些文字，这个框中带有一个 OK 按钮。缺省情况下，代码会暂停直到用户点击了 OK 按钮。

```
Sub TestMessageBox1()
    MsgBox "Your hard drive will now be formatted."
End Sub
```

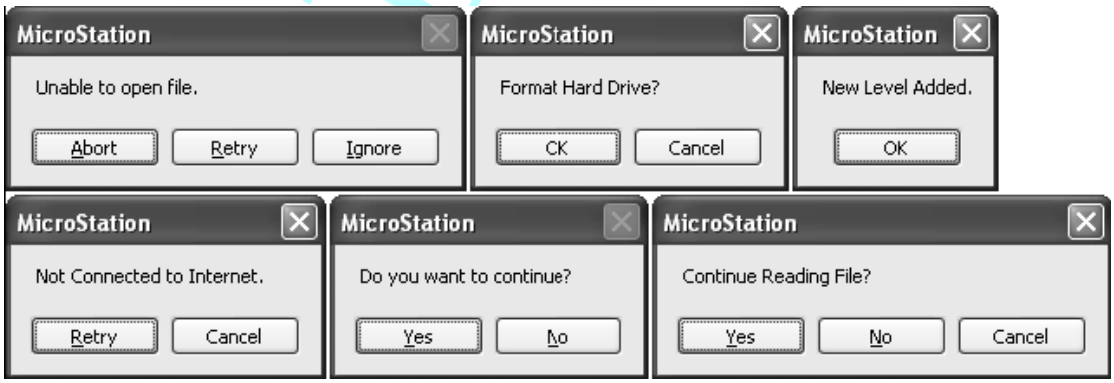


这正是我们想要看到的：一个消息框告知：某些重要的事情将要发生，我们所能做的就是点击 OK 按钮。

你能指定消息框的提示信息(就是图中所示的文字)，也能指定显示哪种按钮。

```
Sub TestMessageBox2()
    Dim MsgResp As VbMsgBoxResult
    MsgResp = MsgBox("Unable to open file.", vbAbortRetryIgnore)
    MsgResp = MsgBox("Format Hard Drive?", vbOKCancel)
    MsgResp = MsgBox("New Level Added.", vbOKOnly)
```

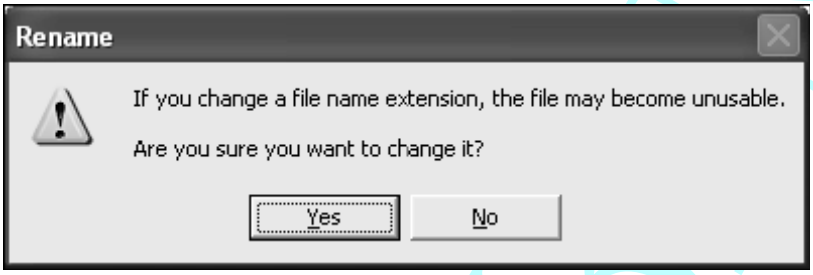
```
MsgResp = MsgBox("Not Connected to Internet.", vbRetryCancel)
MsgResp = MsgBox("Do you want to continue?", vbYesNo)
MsgResp = MsgBox("Continue Reading File?", vbYesNoCancel)
Select Case MsgResp
    Case VbMsgBoxResult.vbAbort
        '此处放置代码
    Case VbMsgBoxResult.vbCancel
        '此处放置代码
    Case VbMsgBoxResult.vbIgnore
        '此处放置代码
    Case VbMsgBoxResult.vbNo
        '此处放置代码
    Case VbMsgBoxResult.vbOK
        '此处放置代码
    Case VbMsgBoxResult.vbRetry
        '此处放置代码
    Case VbMsgBoxResult.vbYes
        '此处放置代码
End Select
End Sub
```



消息框可以带有多个按钮，这一点使得它功能很强大。现在，你正向用户提问：是(Yes)？否(No)？重试(Rety)？放弃(Abort)？忽略(Ignore)？取消(Cancel)？确定(OK)？当你问一个问题时，需要用户给出答案，此时就要把消息

框作为一个函数取得它的返回值。

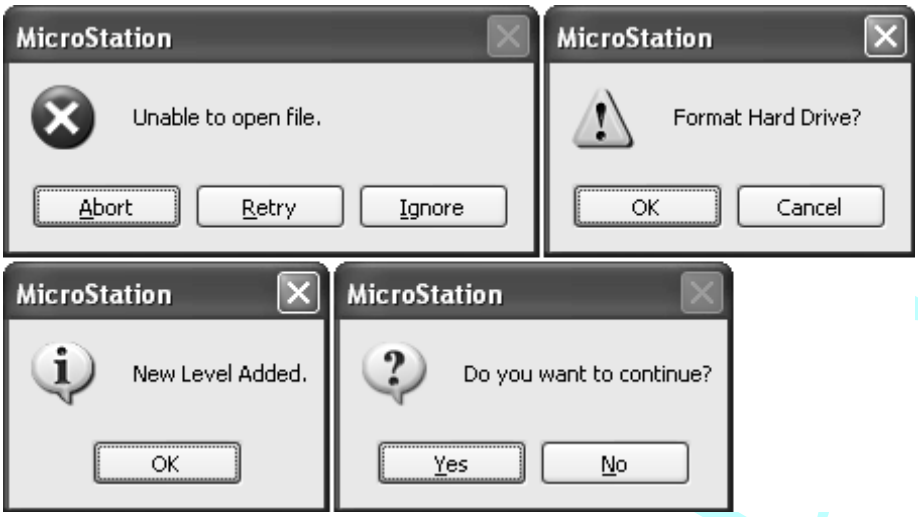
上面的例子仅用于演示，每个消息框都返回一个值给变量 **MsgResp**，该变量告知我们用户点击了哪个按钮。我们用 Select Case 语句来确定所按的按钮，然后基于这个按钮来执行代码。Select Case 语句位于“YesNoCancel”消息框之后，所以只有这个消息框的按钮被考虑了。



这里是一个 Microsoft Windows 的消息框，当你试图改变某个文件的扩展名(假设从.txt 到.dgn)时，则显示出该消息框，请注意 Yes 和 No 按钮。我们现在知道了怎样指定按钮，但是三角形中的感叹号又是什么意思呢？如何实现它？

当显示消息框时，用常数来指定框中显示哪个按钮，例如 **vbOKCancel**、**vbYesNo** 和 **vbOKOnly**，这些常数有一个对应的数值。有另外一些常数用来指定显示哪些图标，如 **vbExclamation**。在指定按钮常数时加上图标常数，VBA 会在消息框中同时显示按钮和图标。

```
Sub TestMessageBox3()
    Dim MsgResp As VbMsgBoxResult
    MsgResp = MsgBox("Unable to open file.", _
                vbAbortRetryIgnore + vbCritical)
    MsgResp = MsgBox("Format Hard Drive?", _
        vbOKCancel + vbExclamation)
    MsgResp = MsgBox("New Level Added.", vbOKOnly + vbInformation)
    MsgResp = MsgBox("Do you want to continue?", vbYesNo + _
    vbQuestion)
End Sub
```



在使用带有按钮常数的这些图标常数时，Windows 显示按钮和图标并播放特定的声音。

让我们再看一下消息框中另外一个参数。

```
Sub TestMessageBox4()
    MsgBox "Testing Title", vbCritical, "Title Goes Here"
    MsgBox "Testing Title", , "Title Goes Here"
End Sub
```



标题参数显示在消息框的顶部，它是第三个参数，消息框仅有一个参数是必须的，那就是提示参数。要显示提示、标题以及缺省按钮，在提示参数后放一个逗号、一个空格、另一个逗号，然后是标题。若要跳过一个可选参数，就让这个参数空着并用逗号指明您要提供下一个参数了。

输入框

输入框让用户输入文本。如果用户点击取消(Cancel)按钮或什么也没输入就点击确定(OK)按钮，InputBox(即输入框)将返回一个空串。VBA 中的空串表示中间什么内容也没有的两个双引号(“”)。

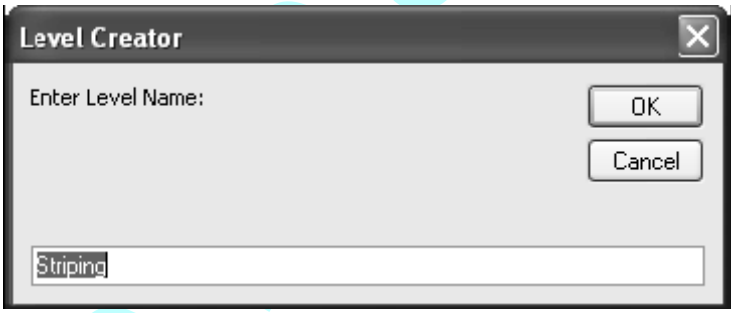
```
Sub TestInputBox1()
```

```
    Dim InpRet As String
    InpRet = InputBox("Enter Level Name:")
    Debug.Print "User entered " & InpRet
End Sub
```



InputBox 还有四个可用的参数，接下来就要讨论它们。

```
Sub TestInputBox2()
    Dim InpRet As String
    InpRet = InputBox("Enter Level Name:", _
                      "Level Creator", "Striping", 0, 0)
    Debug.Print "User entered " & InpRet
End Sub
```

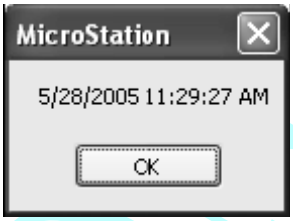


仔细观察代码和新参数引起的结果变化。在提示和标题后是向输入框提供的一个缺省值，然后是输入框显示的 X、Y 位置。X 和 Y 值以像素为单位且依赖于操作系统的坐标，这就意味着，如果您使用坐标 0,0，输入框将在监视器的左上角显示而与 MicroStation 窗口的位置无关。

小心使用 X 和 Y 位置参数，因为有可能把输入框完全置于屏幕之外。如果用户看不到输入框而代码又在等待用户点击按钮或按**<Enter>**键，这种情况一定会把人搞糊涂。

Now

Now(现在)函数给出当前的系统日期和时间。这在生成一个日期/时间戳时很有用。Now 返回一个 Date(日期)类型的值。

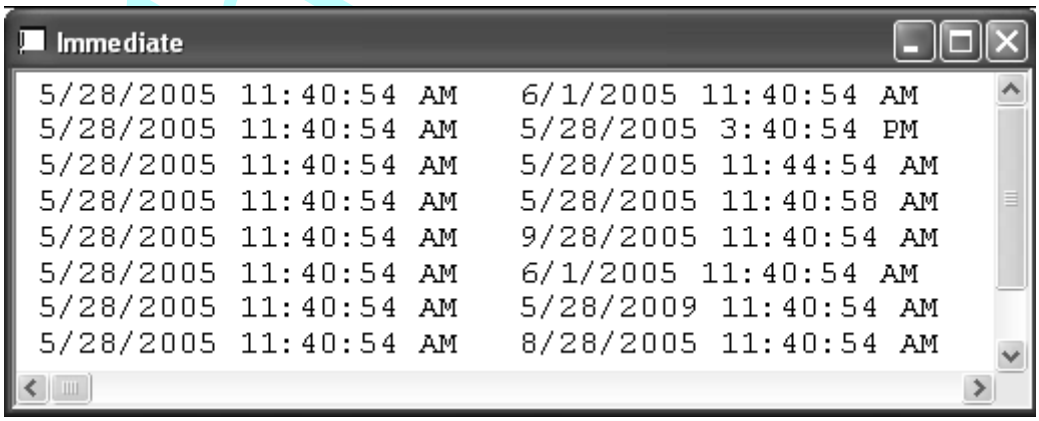



```
Sub TestNow()
    MsgBox Now
End Sub
```

DateAdd

Now 函数告诉我们当前日期/时间，DateAdd 函数能展望未来或回忆过去。下面是一些使用 DateAdd 的例子：

```
Sub TestDateAdd()
    Dim NowDate As Date
    NowDate = Now
    Debug.Print NowDate & vbTab & DateAdd("d", 4, NowDate) '日
    Debug.Print NowDate & vbTab & DateAdd("h", 4, NowDate) '时
    Debug.Print NowDate & vbTab & DateAdd("n", 4, NowDate) '分
    Debug.Print NowDate & vbTab & DateAdd("s", 4, NowDate) '秒
    Debug.Print NowDate & vbTab & DateAdd("m", 4, NowDate) '月
    Debug.Print NowDate & vbTab & DateAdd("w", 4, NowDate) '周
    Debug.Print NowDate & vbTab & DateAdd("yyyy", 4, NowDate)'年
    Debug.Print NowDate & vbTab & DateAdd("q", 1, NowDate)'季
End Sub
```



上例中我们声明了一个日期型的变量并把它的值设置为 Now。当然可以在

每个 DateAdd 函数中直接用函数 Now，但因为 Now 在一秒一秒地变化，所以设置一个变量到 Now 然后在整个过程中使用该变量是个好主意，这样能保证所有的计算都是基于相同的日期/时间。在第二个参数中用正数把结果移至未来，使用负数返回过去的一个值。

DateDiff

如果有两个日期并且想知道它们之间的时间差，请用 DateDiff 函数。DateAdd 和 DateDiff 使用相同的间隔参数。

```
Sub TestDateDiff()
    Dim NowDate As Date
    NowDate = Now
    Debug.Print "Days" & vbTab & DateDiff("d", NowDate, "1/1/3000")
    Debug.Print "Hours" & vbTab & DateDiff("h", NowDate, "1/1/3000")
    Debug.Print "Minutes" & vbTab & DateDiff("n", NowDate, "1/1/3000")
    Debug.Print "Seconds" & vbTab & DateDiff("s", NowDate, "1/1/3000")
    Debug.Print "Months" & vbTab & DateDiff("m", NowDate, "1/1/3000")
    Debug.Print "Weeks" & vbTab & DateDiff("w", NowDate, "1/1/3000")
    Debug.Print "Years" & vbTab & DateDiff("yyyy", NowDate, "1/1/3000")
    Debug.Print "Quarters" & vbTab & DateDiff("q", NowDate, "1/1/3000")
End Sub
```

Y2K(两千年)问题对我们大多数人来说已经不成为问题了。本例展望到了 Y3K(三千年)，显示出了从现在到令人担心的 3000 年 1 月 1 日有多少天、小时、分钟、秒、月、星期、年和季。其实只有 31,386,398,632 秒。

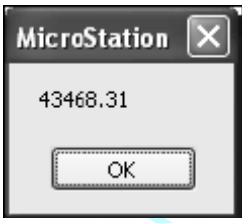
间隔标记是该函数的第一个参数，下两个参数是要比较的日期。如果第一个日期比第二个日期早，将得到正的返回值；如果第一个日期比第二个日期晚，将得到负的返回值。

Timer

Timer 函数告诉我们自午夜开始所经历的秒数。这个函数在测试应用以查找代码中的瓶颈时很有用。但如果你在晚上工作到很晚时就要小心了，当时钟到达

午夜时，Timer 函数将返回 0 值并从头开始计秒。

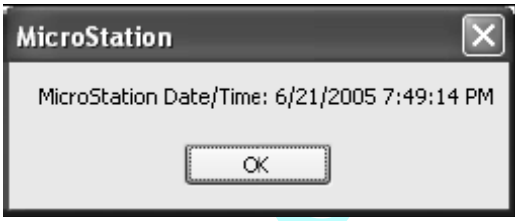
```
Sub TestTimer()
    MsgBox Timer
End Sub
```



FileDateTime

FileDateTime 给出指定文件最后修改的日期/时间。

```
Sub TestFileDateTime()
    Dim exeDate As Date
    exeDate = FileDateTime _
            ("C:\Program Files\Bentley\MicroStation\ustation.exe")
    MsgBox "MicroStation Date/Time: " & exeDate
End Sub
```



FileLen

FileLen 告知给定文件的大小(以字节为单位)。

```
Sub TestFileLen()
    Dim exeSize As Long
    exeSize = FileLen _
            ("C:\Program Files\Bentley\MicroStation\ustation.exe")
    MsgBox "MicroStation Size: " & exeSize
End Sub
```



MkDir

用 MkDir 建立新目录。要使 MkDir 正常工作，所有的父目录必须存在。例如，要建立一个目录(也叫文件夹) c:\MicroStation VBA\Chapters\09\Samples 但从 Chapters 目录以下都不存在，此时就必须先建立“Chapters”目录、然后建

立“09”目录，最后才能建立“Samples”目录。

```
Sub TestMkDir()
    MkDir "c:\MicroStation VBA\Source Code"
End Sub
```

RmDir

RmDir 从文件系统中删除一个目录。被删的目录必须是空的，否则会产生一个错误。

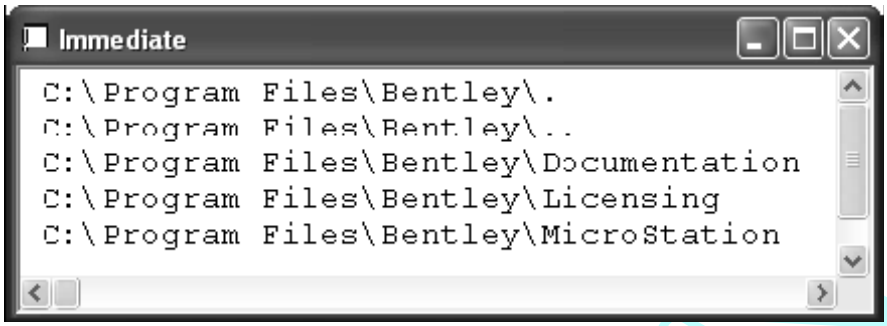
```
Sub TestRmDir()
    RmDir "c:\MicroStation VBA\Source Code"
End Sub
```

Dir

Dir 函数能查找文件和文件夹(目录)。第一次使用该函数时，指定一个路径和文件名(允许通配符)，Dir 每次只返回一个文件/路径。如果要查找一组文件或文件夹，反复调用不带参数的 Dir。当 Dir 返回一个空串(“”)时，表明已返回所有的文件或文件夹名。除了指定一个要查找的文件或文件夹路径/名外，还能指定文件/文件夹的类型。由于 Dir 函数还能完成大量的功能，所以我们将考察几个代码的例子及其结果。

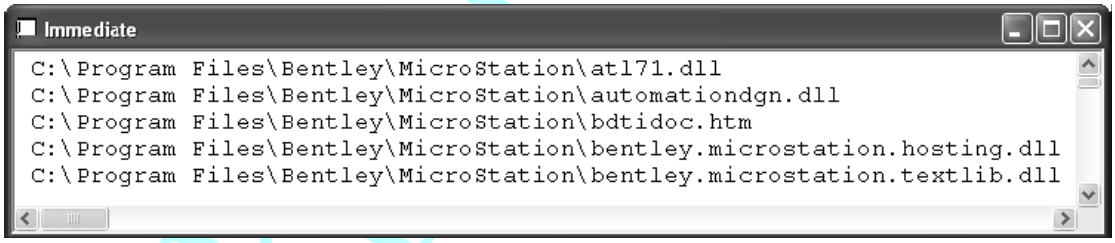
```
Sub TestDir1()
    Dim RootPath As String
    Dim DirReturn As String
    RootPath = "C:\Program Files\Bentley"
    DirReturn = Dir(RootPath & "\*.*", vbDirectory)
    While DirReturn <> ""
        Debug.Print RootPath & "\" & DirReturn
        DirReturn = Dir
    Wend
End Sub
```

China BDN



第一个例子在“C:\Program Files\Bentley”目录下查找所有目录。注意，查到的第一个目录叫“.”，第二个叫“..”。所有查找目录的调用情况都是这样，这两个目录应该在代码中被忽略掉。这两个目录分别指当前文件夹和父文件夹。

```
Sub TestDir2()
    Dim RootPath As String
    Dim DirReturn As String
    RootPath = "C:\Program Files\Bentley\MicroStation"
    DirReturn = Dir(RootPath & "\*.*")
    While DirReturn <> ""
        Debug.Print RootPath & "\" & DirReturn
        DirReturn = Dir
    Wend
End Sub
```

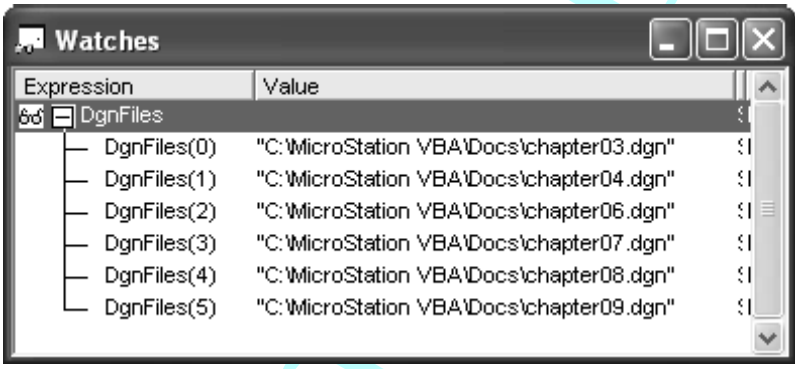


现在我们在“C:\Program Files\Bentley\MicroStation”文件夹里查找所有文件(*.*)。

把文件名写入立即窗口对演示来说还好，但不符合实际工作情况。这次让我们做一修改，把所有的文件名放入一个变量中。

```
Sub TestDir3()
    Dim RootPath As String
    Dim DirReturn As String
```

```
    Dim DgnFiles() As String
    ReDim DgnFiles(0) As String
    RootPath = "C:\MicroStation VBA\Docs"
    DirReturn = Dir(RootPath & "\*.dgn")
    While DirReturn <> ""
        DgnFiles(UBound(DgnFiles)) = RootPath & "\" & DirReturn
        ReDim Preserve DgnFiles(UBound(DgnFiles) + 1)
        DirReturn = Dir
    Wend
    ReDim Preserve DgnFiles(UBound(DgnFiles) - 1)
End Sub
```



我们在“C:\MicroStation VBA\Docs”目录中查找扩展名为.dgn 的文件，把这些文件的路径放入一个动态数组变量 **DgnFiles** 中。当代码运行到“End Sub”行时，找到了六个文件并被放入数组中。您应该在“End Sub”前写一些代码来处理这些文件。

Kill

警告：*Kill 函数的结果是永久性的。被“杀掉”的文件没有送到回收站，它们被完全删除。使用时要特别小心！*

```
Sub TestKill()
    Kill "C:\MicroStation VBA\Docs\killtest.txt"
End Sub
```

这段代码删除文件 C:\MicroStation VBA\Docs\killtest.txt。

删除文件的功能有用且必须，但使用时必须要小心。

Beep

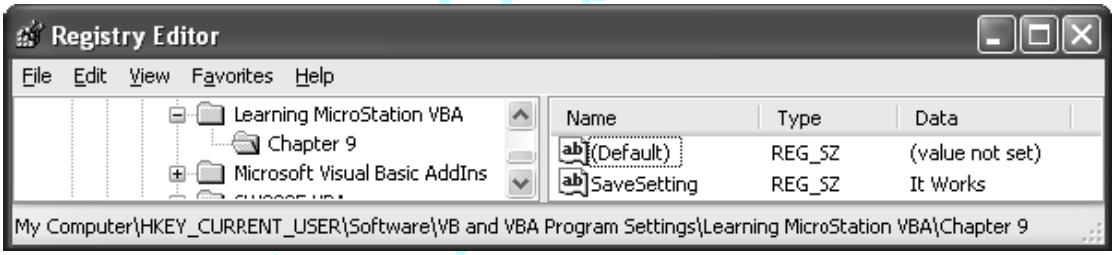
Beep 函数使电脑发出哔哔声，在代码运行过程中它能给用户一个快速的声音提示。使用 Beep 虽然有益于让用户关注程序，但每次操作到这里时都发出哔哔声也很烦人。

```
Sub TestBeep()
    Beep
End Sub
```

SaveSetting

使用 Windows 注册表能保存用户在软件中的设置。微软已经为 VBA 程序设置建立了一个注册表路径，我们能很容易地写、编辑和删除这个路径。

```
Sub TestSaveSetting()
    SaveSetting "Learning MicroStation VBA", "Chapter 9", _
                "SaveSetting", "It Works"
End Sub
```



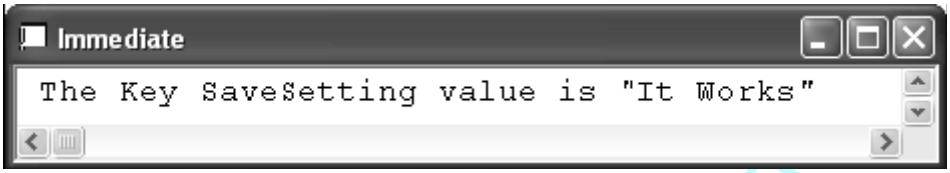
这段代码运行后，添加了所需的注册表文件夹，在该文件夹下建立了一个名叫“SaveSetting”的注册项，该注册项的值是“It Works”。

GetSetting

用 GetSetting 能取得注册表中的设置。

```
Sub TestGetSetting()
    Dim RegSetting As String
    RegSetting = GetSetting("Learning MicroStation VBA", "Chapter 9", _
                "SaveSetting")
    Debug.Print "The Key SaveSetting value is """ & RegSetting & """"
```

```
End Sub
```



DeleteSetting

我们能保存和获得设置，也能删除它们。正如其它删除文件或数据的 API 调用一样，也要小心使用这个函数。

```
Sub TestDeleteSetting1()
    DeleteSetting "Learning MicroStation VBA", _
                  "Chapter 9", "SaveSetting"
End Sub
```

以上代码运行后，“SaveSetting”键被删除。

```
Sub TestDeleteSetting2()
    DeleteSetting "Learning MicroStation VBA", "Chapter 9"
End Sub
```

TestDeleteSetting2 删除了注册表段“Chapter 9”。

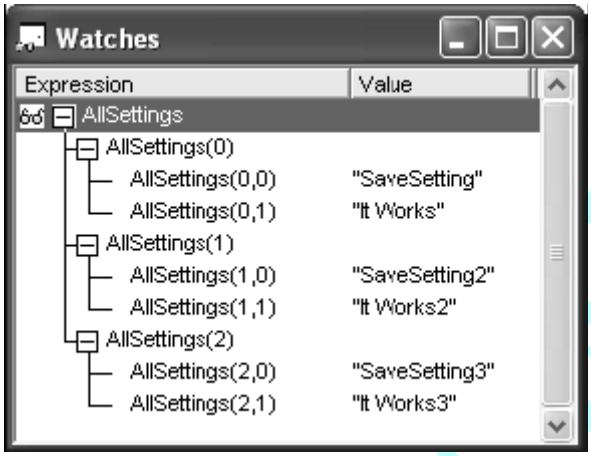
```
Sub TestDeleteSetting3()
    DeleteSetting "Learning MicroStation VBA"
End Sub
```

TestDeleteSetting3 从注册表中删除了整个“Learning MicroStation VBA”应用名及其所有子项。

GetAllSettings

顾名思义，GetAllSettings 从注册表中取得指定应用下的所有键，并把它们放入一个多维数组中。

```
Sub TestGetSettings()
    Dim AllSettings As Variant
    AllSettings = GetAllSettings("Learning MicroStation VBA", _
                  "Chapter 9")
End Sub
```



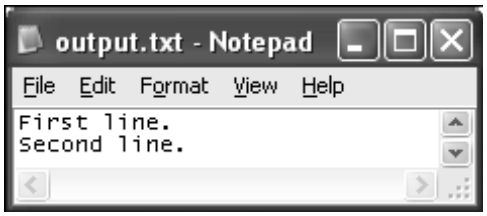
通过监视变量 AllSettings 能让我们看到调用完 GetAllSettings 后该变量的结构和值。

读写 ASCII 文件

给我一个文本文件，我就能移动这个世界！这个文本文件被假定成了“杠杆”？能够读写 ASCII 文本文件给我们编写程序提供了强大的支持。许多程序和数据库能够读写这种文件。那么，什么是 ASCII 文件呢？

一个 ASCII 文本文件完全由 ASCII 字符组成。它能被记事本或写字板打开，也能被人们直接阅读。

```
Sub TestWriteASCIIA()
    Open "C:\output.txt" For Output As #1
        Print #1, "First line."
        Print #1, "Second line."
    Close #1
End Sub
```



右图是用记事本打开的文件。现在，让我们分析一下代码。

首先，指定一个要操作的文件，上例中操作的是文件 C:\output.txt。你能使用任何文件扩展名，但是要小心！如果写了一个新的 ASCII 文件并给以.dgn 的扩展名，Microsoft Windows 会认为它是一个 MicroStation 的文件，当你双击此文件时，会试着用 MicroStation 打开它。

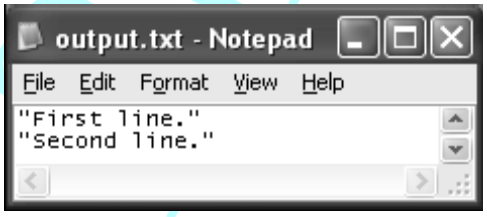
在写文件时有两种选择：使用“Output”关键字意思是如果指定的文件不存在就创建它，如果存在就覆盖它；或者使用“Append”关键字，意思是在现有文件的尾部添加，如果指定的文件不存在就创建新文件。

在打开一个文件时，需要赋给文件一个号，随后读取或写入文件时都要用到这个号。上例中我们使用了一个文件号“#1”。每次使用时井号符必须位于文件号之前。

接下来就是写一些文本到文件中，上例中使用了 Print 函数。还能用 Write 函数(出现在下面的例子中)来写，但结果稍有不同。

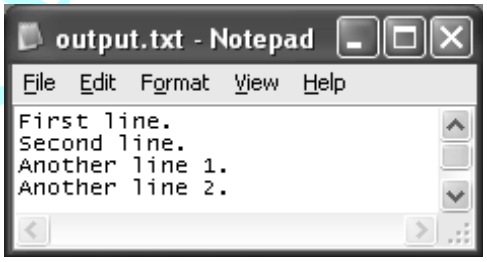
最后，用 Close 函数关闭文件。

```
Sub TestWriteASCIIB()
    Open "C:\output.txt" For Output As #1
    Write #1, "First line."
    Write #1, "Second line."
    Close #1
End Sub
```



Write 函数在每行的开头和结尾处放置了双引号，这一点在需要时很有用。

```
Sub TestWriteASCIIC()
    Open "C:\output.txt" For Append As #1
    Print #1, "Another line 1."
    Print #1, "Another line 2."
    Close #1
End Sub
```



要在现有的文件中添加文本就用“For Append”来打开文件。上图中的屏幕快照是运行完 TestWriteASCIIA 后又运行 TestWriteASCIIC 的结果。

FreeFile

向 VBA 提供一个指向文件的文件号很重要。前面的例子中使用了“#1”作为文件号，因为例子简单所以代码运行得很好。但如果你的程序同时打开多个文件，该用哪个号作为新的文件号可能就会引起困惑。此时 FreeFile 就该派上用场了。

```
Sub TestWriteASCIID()
    Dim FFileA As Long
```

```
    Dim FFileB As Long
    FFileA = FreeFile
    Open "C:\outputa.txt" For Append As #FFileA
        Print #FFileA, "Another line 1."
        Print #FFileA, "Another line 2."
        FFileB = FreeFile
        Open "C:\outputb.txt" For Append As #FFileB
            Print #FFileA, "Another line 3."
            Print #FFileB, "Another line 3."
            Print #FFileA, "Another line 4."
            Print #FFileB, "Another line 4."
        Close #FFileB
    Close #FFileA
End Sub
```

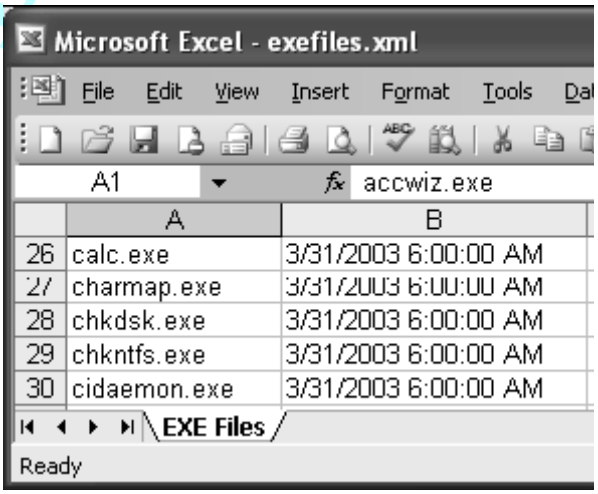
上例同时操作两个文件。当使用 FreeFile 时，把其返回值赋给变量，上例中用 **FFileA** 和 **FFileB** 作为变量名。

请注意 FreeFile 用于多个变量的情况。如果用 FreeFile 连续给 **FFileA** 和 **FFileB** 赋值，它们的值将会是相同的。FreeFile 仅当一个文件被打开后才会返回一个不同的值。所以调用 FreeFile 后要打开以它为文件号的文件，然后再调用 FreeFile 以打开下一个文件。这样做就能避免取得相同的文件号，否则，当你本意是想读/写两个不同的文件时，可能偶尔会出现读/写同一文件的情况。

下面是一个写 ASCII 文本文件的更高级一些的应用程序。XML 文档是具有特定格式的 ASCII 文本文件。下例建立了一个 Microsoft Excel XML 文档，该文档包含“C:\Windows\System32”文件夹中的所有 EXE 文件及其最后修改日期和时间。这个 XML 文件建立以后，可以在 Microsoft Excel 中打开它。

```
Sub TestWriteASCIIE()
    Dim FFileA As Long
    Dim exeFile As String
    FFileA = FreeFile
    Open "c:\exefiles.xml" For Output As #FFileA
```

```
    Print #FFileA, "<?xml version=""1.0""?>"
    Print #FFileA, "<?mso-application progid=""Excel.Sheet""?>"""
    Print #FFileA, _
        "<Workbook xmlns=""urn:schemas-microsoft-” & _
        “com:office:spreadsheet"">"
    Print #FFileA, " <Worksheet ss:Name=""EXE Files"">"
    Print #FFileA, " <Table>"
    exeFile = Dir("C:\Windows\System32\*.exe")
    While exeFile <> ""
        Print #FFileA, " <Row>"
        Print #FFileA, _
            " <Cell><Data ss:Type=""String"">" & exeFile & _
            "</Data></Cell>"
        Print #FFileA, _
            " <Cell><Data ss:Type=""String"">" & _
            FileDateTime("C:\Windows\System32\" & exeFile) & _
            "</Data></Cell>"
        Print #FFileA, " </Row>"
        exeFile = Dir
    Wend
    Print #FFileA, " </Table>"
    Print #FFileA, " </Worksheet>"
    Print #FFileA, "</Workbook>"
    Close FFileA
End Sub
```



Excel 对 XML 文档的格式很挑剔，如果从本书复制代码运行时出了问题，请打开并运行随书附送的 CD 里带的代码。

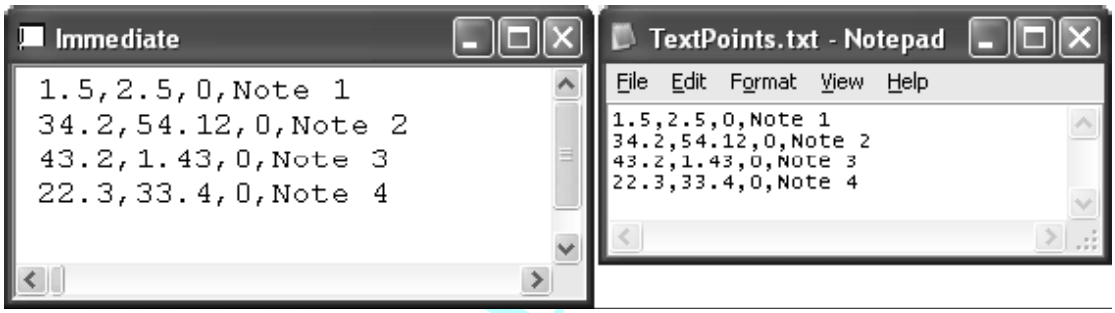
上图是在 Excel 中显示的该 XML 文件的部分结果。

China BDN

从 ASCII 文件中读

写 ASCII 文件很容易，读它们也很容易。让我们一次读取一行内容。

```
Sub ReadASCIIA()
    Dim FFile As Long
    Dim TextLine As String
    FFile = FreeFile
    Open "C:\MicroStation VBA\TextPoints.txt" For Input As #FFile
    While EOF(FFile) = False
        Line Input #FFile, TextLine
        Debug.Print TextLine
    Wend
    Close #FFile
End Sub
```



上图左侧是在立即窗口中显示所读文件的每一行，右侧是在记事本中打开的源文件。用 Line Input 和文件号每次读文本文件中的一行，一直读下去直到遇到文件尾(EOF)。下面是对该例子的扩展。

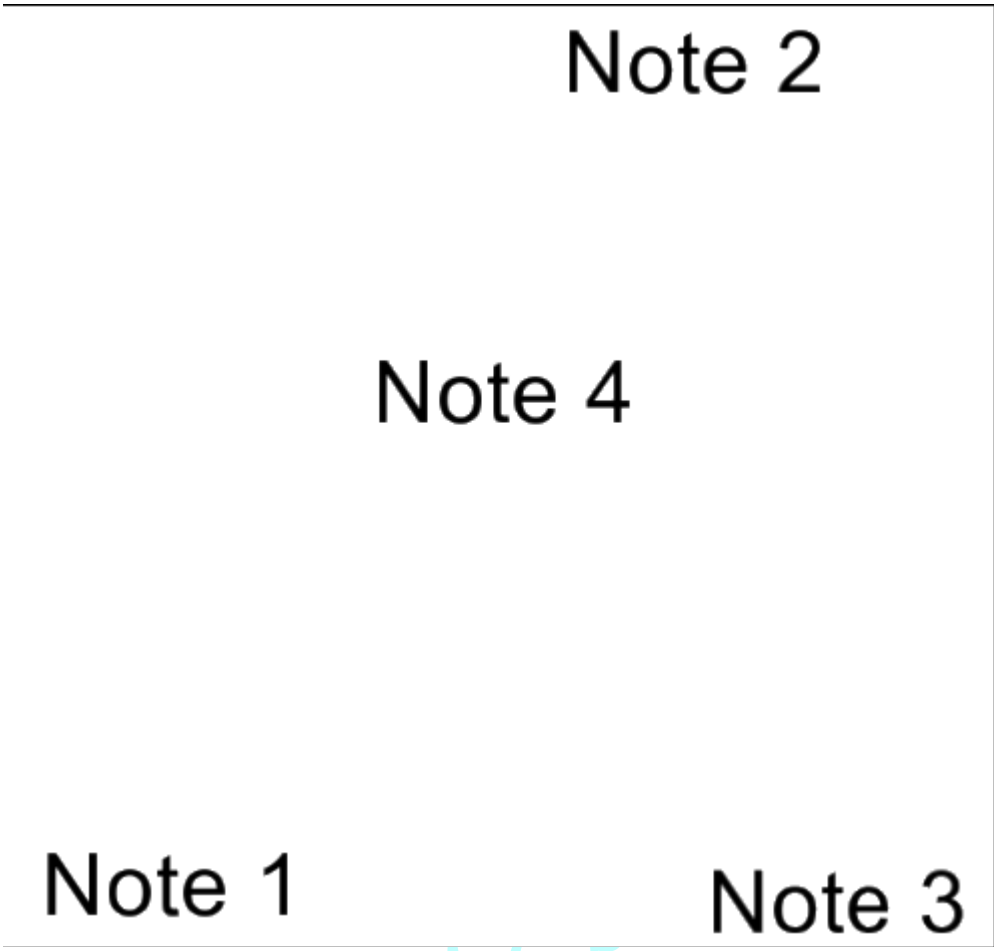
```
Sub ReadASCIIB()
    Dim FFile As Long
    Dim TextLine As String
    Dim TextPoint As Point3d
    Dim XSplit As Variant
    Dim TextElem As TextElement
    Dim RotMat As Matrix3d
    FFile = FreeFile
```

```
    Open "C:\MicroStation VBA\TextPoints.txt" For Input As #FFile
    While EOF(FFile) = False
        Line Input #FFile, TextLine
        XSplit = Split(TextLine, ",")
        TextPoint.X = XSplit(0)
        TextPoint.Y = XSplit(1)
        TextPoint.Z = XSplit(2)
        Set TextElem = Application.CreateTextElement1(Nothing, _
                        XSplit(3), TextPoint, RotMat)
        ActiveModelReference.AddElement TextElem
    Wend
    Close #FFile
End Sub
```

我们扩展了“ReadASCIIA”。现在，不是把文本文件中的信息写入立即窗口，而是按每行文本指定的 X、Y、Z 位置在 MicroStation 中建立新的文本元素。

下图是严格按照 ASCII 文件中指定的位置放置的注释文本。

China BDN



控制代码的执行

知道如何多次循环执行代码和基于某种条件来执行代码是最基本的要求了。

For … Next

当知道要循环执行一段代码多少次时，用一个 For … Next 语句。下面是个简单的样例：

```
Sub ForNextA()
    Dim I As Long
    For I = 1 To 10
        ActiveDesignFile.AddNewLevel "NewLevel " & I
    Next I
End Sub
```

这段代码运行后建立了 10 个新层，从“NewLevel 1”到“NewLevel 10”。For … Next 语句需要一个变量，本例中使用一个叫做 I 的长整型变量。第

一次建立新层时，I 的值是 1(一)，第二次时 I 的值是 2(二)，I 值一直从 1 变化到 10。I 的最终值为 11(十一)，因为 11 超出了指定的范围而使得程序退出 For … Next 循环，随后 VBA 继续执行 For … Next 循环后面的代码。

```
Sub ForNextB()
    Dim I As Long
    For I = 1 To 10 Step 2
        ActiveDesignFile.AddNewLevel "NewLevelB " & I
    Next I
End Sub
```

我给 For … Next 语句加了一个可选参数，那就是 Step(步长)参数。缺省情况下，For … Next 循环每次对索引参数加 1(一)。但上面这段代码运行时，I 变量将得到 1、3、5、7、9 然后以 11 结束并退出循环，因为我们使用了“Step 2”。

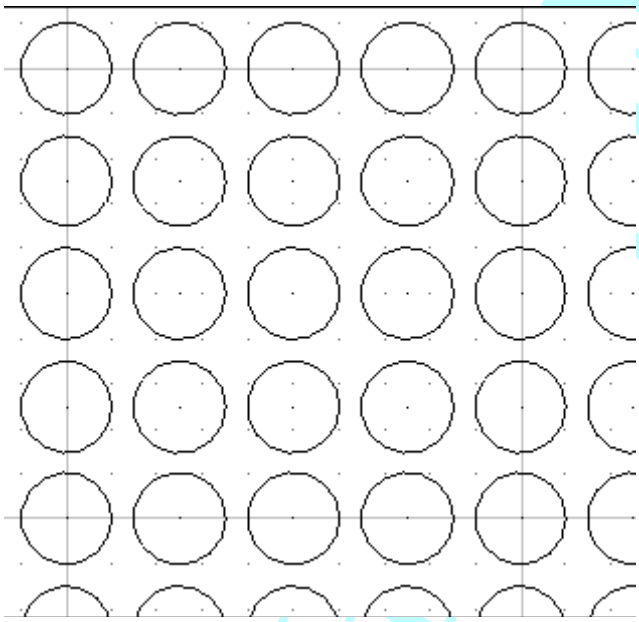
```
Sub ForNextC()
    Dim I As Long
    For I = 10 To 1 Step -1
        ActiveDesignFile.AddNewLevel "NewLevelC " & I
    Next I
End Sub
```

我刚刚又把步长参数改为-1，这就意味着 I 将得到如下值：10、9、8、7、6、5、4、3、2、1 然后是 0 并且退出循环，因为 0 超出了循环的边界。

```
Sub ForNextD()
    Dim X As Double
    Dim Y As Double
    Dim InsPt As Point3d
    Dim CellElem As CellElement
    For X = 0 To 10 Step 0.25
        For Y = 0 To 10 Step 0.25
            InsPt.X = X
            InsPt.Y = Y
            Set CellElem = Application.CreateCellElement3("Column", _
```


China BDN

```
                InsPt, True)
            ActiveModelReference.AddElement CellElem
        Next Y
    Next X
End Sub
```



在运行这段程序前要求定义有单元“Column”。一个 For … Next 循环嵌套在另一个循环中，步长语句每次对变量增加 0.25。注意，我把索引变量声明为双精度型以保证它们能存储小数值。

While … Wend

当不能确定要重复多少次代码块时使用 While … Wend。只要 While 语句为真，在 While 和 Wend 之间的代码语句就继续执行。下面是在本章已经看过的一个过程的一部分。

```
    While EOF(FFile) = False
        Line Input #FFile, TextLine
        XSplit = Split(TextLine, ",")
        TextPoint.X = XSplit(0)
        TextPoint.Y = XSplit(1)
        TextPoint.Z = XSplit(2)
        Set TextElem = Application.CreateTextElement1(Nothing, _
```

```
                XSplit(3), TextPoint, RotMat)
        TextElem.TextStyle.Height = 4
        TextElem.TextStyle.Width = 4
        ActiveModelReference.AddElement TextElem
    Wend
```

一个数据文件可能有 1 行、10 行、100 行或任意其它行，因此我们持续考察文件的 EOF(文件尾)，只要未到达文件尾就一直执行 While 和 Wend 间的代码。

Do … Loop

Do … Loop 语句与 While … Wend 非常相似，但它更通用些。下面是一个例子：

```
Sub TestDoWhileA()
    Dim CadMsg As CadInputMessage
    Dim InsPt As Point3d
    Dim CellElem As CellElement
    Do While True
        Set CadMsg = CadInputQueue.GetInput
        Select Case CadMsg.InputType
            Case msdCadInputTypeDataPoint
                InsPt = CadMsg.Point
                Exit Do
        End Select
    Loop
    Set CellElem = Application.CreateCellElement3("Column", _
                   InsPt, True)
    ActiveModelReference.AddElement CellElem
End Sub
```

可以使用“Exit Do”在任何时候退出 Do … Loop 语句。本例让用户在 MicroStation 中选择一个点，当用户选择了点以后就捕获这个点并退出 Do 循环，然后在捕获的点处插入一个新单元。

下一个例子是上面过程的修改版。程序代码会一直插入单元直到用户在键盘上按了一个键。

```
Sub TestDoWhileB()
    Dim CadMsg As CadInputMessage
    Dim InsPt As Point3d
    Dim CellElem As CellElement
    Do While True
        Set CadMsg = CadInputQueue.GetInput
        Select Case CadMsg.InputType
            Case msdCadInputTypeDataPoint
                InsPt = CadMsg.Point
                Set CellElem = _
                    Application.CreateCellElement3("Column", _
                        InsPt, True)
                ActiveModelReference.AddElement CellElem
            Case msdCadInputTypeCommand
                Exit Do
        End Select
    Loop
End Sub
```

使用“Do While True”时会使程序一直停留在循环中直到遇到“Exit Do”或“Exit Sub”或“Exit Function”。“Exit Do”退出循环继续执行过程或函数中的代码，而“Exit Sub”和“Exit Function”将退出过过程或函数。

```
Sub TestDoWhileC()
    Dim TextToPlace As String
    Dim LineNumber As Long
    Dim NotePt As Point3d
    Dim CadMsg As CadInputMessage
    Dim TextElem As TextElement
    Dim RotMat As Matrix3d
```


China BDN

```
    Do While True
        Set CadMsg = CadInputQueue.GetInput
        Select Case CadMsg.InputType
            Case msdCadInputTypeDataPoint
                NotePt = CadMsg.Point
                Set TextElem = Application.CreateTextElement1(Nothing, _
                                "# Note", NotePt, RotMat)
                ActiveModelReference.AddElement TextElem
                Exit Do
        End Select
    Loop
    TextToPlace = "The following notes supercede all prior notes."
    LineNumber = 1
    Do
        NotePt.Y = NotePt.Y - 0.375
        Set TextElem = Application.CreateTextElement1(Nothing, _
                        LineNumber & ". " & TextToPlace, _
                        NotePt, RotMat)
        ActiveModelReference.AddElement TextElem
        LineNumber = LineNumber + 1
        TextToPlace = InputBox("Enter Note:")
    Loop While TextToPlace <> ""
End Sub
```

该过程使用了两个单独的 Do Loop 语句，让我们集中考察第二个语句。当只有“Do”自己时，不是把条件语句放在了循环的开头而是放在了循环的结尾，循环体内的代码至少要执行一次。本例让用户选择一个点，程序在该点处自动放置“# Note”，又在“# Note”下面放置“1. The following notes supersede all prior notes.”文本。

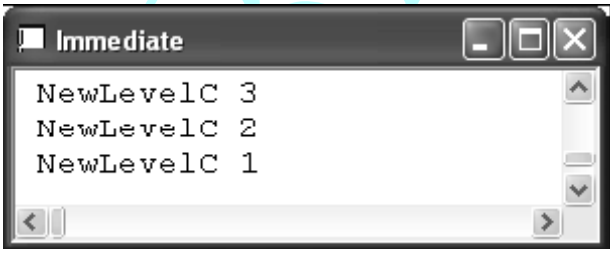
添加完标题和一条标准注释后开始让用户输入附加的注释。每行注释都放在

前一行下的 0.375 个单位处。当用户在输入框中什么也不输入就按 OK 钮的话，循环结束，因为此时 **TextToPlace** 为空串，循环条件不再为真。

For Each … Next

一些对象位于集合中，如每个 DGN 文档都有一个 Levels 集合，该集合由 Level 对象组成。使用 For Each … Next 语句在集合中逐个查看对象。

```
Sub TestForNextA()
    Dim dgnLevel As Level
    For Each dgnLevel In ActiveDesignFile.Levels
        Debug.Print
    dgnLevel.Name
    Next
End Sub
```



使用 For Each … Next 语句时，要指定每个对象使用的变量，然后是要搜索的集合。

If … Then

仅当指定的条件计算结果为真时才会执行 If … Then 语句中特定的代码块。

```
Sub TestIfThenA()
    Dim LevelName As String
    LevelName = InputBox("Enter Level Name (3 letters only)")
    If Len(LevelName) = 3 Then
        ActiveDesignFile.AddNewLevel LevelName
    End If
End Sub
```

在上例中，如果用户输入三个字符的名字，就建立新的层。这是 If … Then 语句非常简单的一个实现。

```
Sub TestIfThenB()
    Dim LevelName As String
    LevelName = InputBox("Enter Level Name (3 letters only)")
```

```
    If Len(LevelName) = 3 Then
        ActiveDesignFile.AddNewLevel LevelName
    Else
        MsgBox LevelName & " has " & Len(LevelName) & _
            " characters."
    End If
End Sub
```

在这个例子中，先查看字符数，如果是三个字符就建立新层。还添加了一个“Else”语句用来处理输入字符长度不等于三的情况。当输入内容不为三个字符时，显示一个消息框指明用户输入的字符数。

```
Sub TestIfThenC()
    Dim LevelName As String
    LevelName = InputBox("Enter Level Name (3 letters only)")
    If Len(LevelName) = 3 Then
        ActiveDesignFile.AddNewLevel LevelName
    ElseIf Len(LevelName) > 3 Then
        ActiveDesignFile.AddNewLevel Left(LevelName, 3)
    Else
        MsgBox LevelName & " has " & Len(LevelName) & _
            " characters."
    End If
End Sub
```

例子 TestIfThenC 介绍了一个 ElseIf 语句，可以在 If .. Then 语句中使用 ElseIf 语句以提供第二个 If 语句。能在 Else 之前使用多个 ElseIf 语句。

Select Case

假想这么一种情况：让用户输入一个层名，然后根据层名的第一个字符采取不同的动作。实现这个功能可以用带有多个 ElseIf 的 If … Then 语句或者干脆用一个 Select Case 语句。

Select Case 要求我们提供一个条件，然后是和该条件相匹配的多个可能值。

```
Sub TestSelectCaseA()
    Dim LevelName As String
    LevelName = InputBox("Enter Level Name:")
    Select Case UCase(Left(LevelName, 1))
        Case "A"
            ActiveDesignFile.AddNewLevel "A_" & LevelName
        Case "B"
            ActiveDesignFile.AddNewLevel "B_B_" & LevelName
        Case "C", "D", "E"
            ActiveDesignFile.AddNewLevel "CDE_" & LevelName
        Case Else
            MsgBox "Not a valid level name."
    End Select
End Sub
```

本例查找输入的层名的首字符，基于首字符有多个可能的代码块要执行。如果首字符不是 A、B、C、D 或 E，就显示一个消息框且不添加新层。如果首字符满足一个准则，就在输入的层名前附加上固定的字符后添加该层。

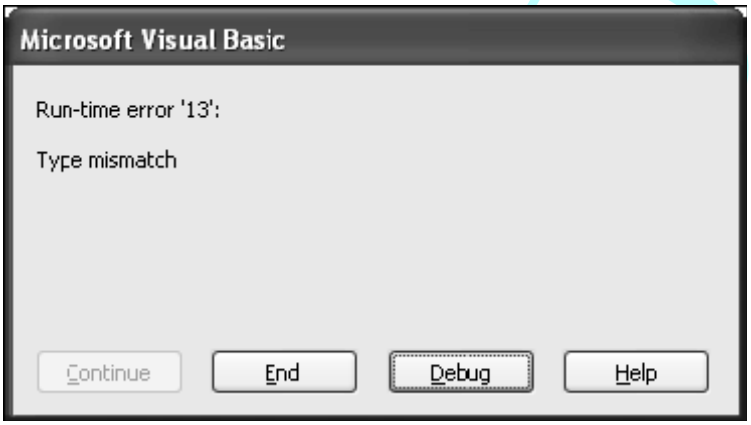
错误处理

在一个拥有完美开发者的完美世界里是从来都不会发出错误的。然而，我们并不完美，所以不时地会发生一些错误。VBA 为我们提供了一些处理错误的工具。

```
Sub TestErrorHndA()
    On Error GoTo errhnd
    Dim LineLength As Double
    LineLength = CDbl(InputBox("Enter Line Length:"))
    Exit Sub
errhnd:
    Select Case Err.Number
        Case 13 '类型不匹配
```

```
            MsgBox "Line Lengths must be numeric."
            Err.Clear
        End Select
    End Sub
```

TestErrorHndA 向用户提问一条线的长度。代码中假定用户知道要输入一个数值，但如果用户输入了像“10 meters”这样的东西将会出问题。若不处理这个错误，用户会看到这样一个对话框：



随着编程经验的丰富，您将能更好地预见到引起错误的一些潜在数据输入问题和其它事项。

所以，在要求用户输入一个长度时，如果不处理类型不匹配错误，用户将看到一个未经处理的错误消息框。此时用户点击调试(Debug)按钮后就会进入代码显示出错的行。然而，如果像上面的宏那样对错误加以处理，用户会看到这样：



在处理错误时，既可以显示消息框告知用户发生了错误，也可以后台处理特定的错误，这样用户就不会知道发生了什么。

现在，让我们仔细地再看一下代码。该过程做的第一件事是说明：

```
On Error GoTo errhnd
```

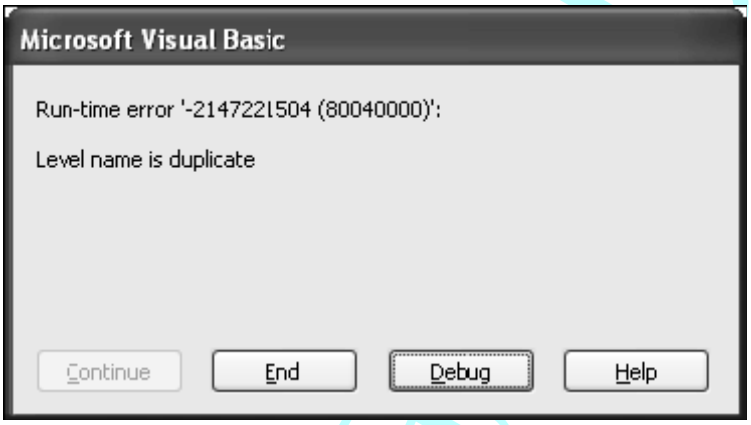
这一行告诉 VBA 如果遇到一个错误就跳转到标记为“errhnd”的代码区。下面就是错误处理代码区：

```
errhnd:
    Select Case Err.Number
        Case 13 '类型不匹配
            MsgBox "Line Lengths must be numeric."
```

```
            Err.Clear
        End Select
```

每个错误都有一个编号与之关联，用 Select Case 语句处理不同类型的错误。本例中仅处理编号 13 的错误，如果发生其它错误，并不会被 Select Case 语句处理，该过程将以 End Sub 结束。

那么，怎样知道需要处理的错误号呢？这是一个好问题。让我们回到几页前的 TestSelectCaseA，运行那个宏并输入“aaa”，一切运行正常。再次运行它并输入“aaa”，发生了什么？



如果两次输入相同的层名，代码将会试着建立重复的层，上面出现的这个消息框给予我们一些很有用的信息。它告诉我们错误号为-2147221504，还有一个描述是“Level name is duplicate(层名重复)”。知道错误号后就能把这个编号加入到代码的错误处理部分。还能通过点击调试(Debug)按钮转到问题代码行仔细地看一看错误发生的地方。

```
Sub TestErrorHndB()
    On Error GoTo errhnd
    Dim LineLength As Double
    LineLength = CDbl(InputBox("Enter Line Length:"))
    Exit Sub
errhnd:
    Select Case Err.Number
        Case 13 '类型不匹配
            MsgBox "Line Lengths must be numeric."
            Err.Clear
```

```
                Resume Next
        End Select
    End Sub
```

TestErrorHndB 除一行外与 TestErrorHndA 一致。在 TestErrorHndB 中加入 Resume Next 语句使得我们的过程发生错误后能继续执行。

```
    Sub TestErrorHndC()
        On Error GoTo errhnd
        Dim LineLength As Double
        LineLength = CDbl(InputBox("Enter Line Length:"))
        Exit Sub
    errhnd:
        Select Case Err.Number
            Case 13 '类型不匹配
                MsgBox "Line Lengths must be numeric."
                Err.Clear
                Resume
        End Select
    End Sub
```

上面这段代码有个小小的改动，用 Resume 语句代替了 Resume Next。Resume 让 VBA 重试出错的代码行，而 Resume Next 则是忽略出错代码行移到下一行执行。

```
    Sub TestErrorHndD()
        On Error Resume Next
        Dim LineLength As Double
        LineLength = CDbl(InputBox("Enter Line Length:"))
    End Sub
```

使用“On Error Resume Next”能告诉 VBA 完全忽略错误并移动到下一行继续执行，从而代替去捕获错误。

虽然“On Error Resume Next”显得有些随意(并且它是很随意)，但却很有

帮助。考虑下一个过程：

```
Sub TestErrHndE()
    On Error Resume Next
    Dim MyExcel As Object
    Set MyExcel = GetObject(, "Excel.Application")
    If Err.Number <> 0 Then
        Err.Clear
        Set MyExcel = CreateObject("Excel.Application")
    End If
    On Error GoTo errhnd
    MyExcel.Visible = True
    MsgBox MyExcel.ActiveSheet.Name
    Exit Sub
errhnd:
    MsgBox "Error " & Err.Number & " has occurred." & vbCr & _
            Err.Description, vbCritical, "Error In TestErrHndE"
    Err.Clear
End Sub
```

本例中使用了 On Error Resume Next，因为能预见到一个潜在的错误。

Set MyExcel = GetObject(, “Excel.Application”)

GetObject 假定要取得的对象已经通过启动 Excel(本例中)创建了。如果 Excel 未启动，在使用 GetObject 时通常会得到一个错误。用 Err 对象得到错误号和描述，在调用 GetObject 后检查 Err.Number 的值，如果非零就表明有错误发生。使用 Err.Clear 从内存中清除以前的错误。

再来讨论下一个操作 Excel 的方法，使用 CreateObject 启动 Excel 应用。On Error GoTo errhnd 通知 VBA 一旦遇到错误就转到“errhnd”区。如果 Excel 未启动，设置 Excel 应用的 Visible(可见)属性为真就会引发一个错误，Excel 不能启动的原因很可能是根本就没有安装。如果 Excel 运行了并且可见，就在消息框中显示当前工作表的名字。这也可能会抛出一个错误，因为即使 Excel 运行了，

还有可能没有打开一个 Excel 工作簿(.xls)文件。

现在再来看代码的 errhnd 部分。复习一下以上的建议，建议中提及了几种可能引起错误的情况。在消息框中显示错误号和描述能让我们知道发生了哪种错误。

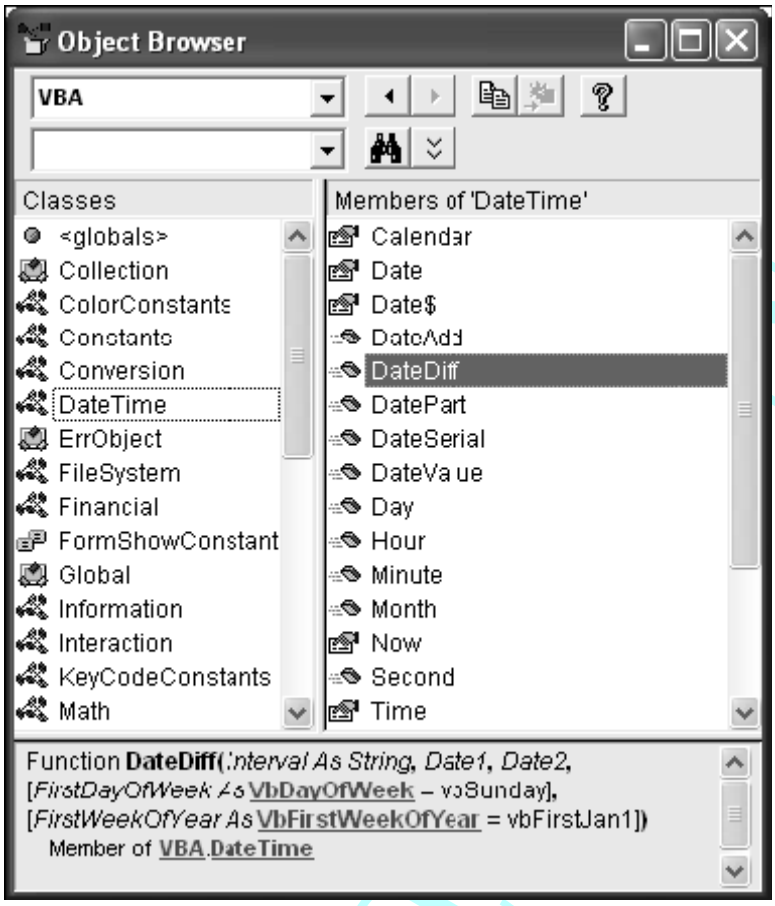
```
Sub TestErrorHndF()
    On Error GoTo errhnd
    Dim LineLength As Double
    On Error GoTo 0
    LineLength = CDbl(InputBox("Enter Line Length:"))
    Exit Sub
errhnd:
    MsgBox "Error " & Err.Number & " has occurred." & vbCr & _
    Err.Description, vbCritical, "Error In TestErrHndE"
    Err.Clear
End Sub
```

“On Error GoTo 0”(Goto 之后是一个零)告诉 VBA 忽略前面的“On Error”语句并继续运行就好像没有错误处理一样。这样做也有用处，你将看到一个显示错误号和描述外加调试(Debug)按钮的错误对话框。

点击 Debug 转到有问题的代码行。一旦找到并修正错误就可以注释掉“On Error GoTo 0”这一行而使你的错误处理代码再次起作用。

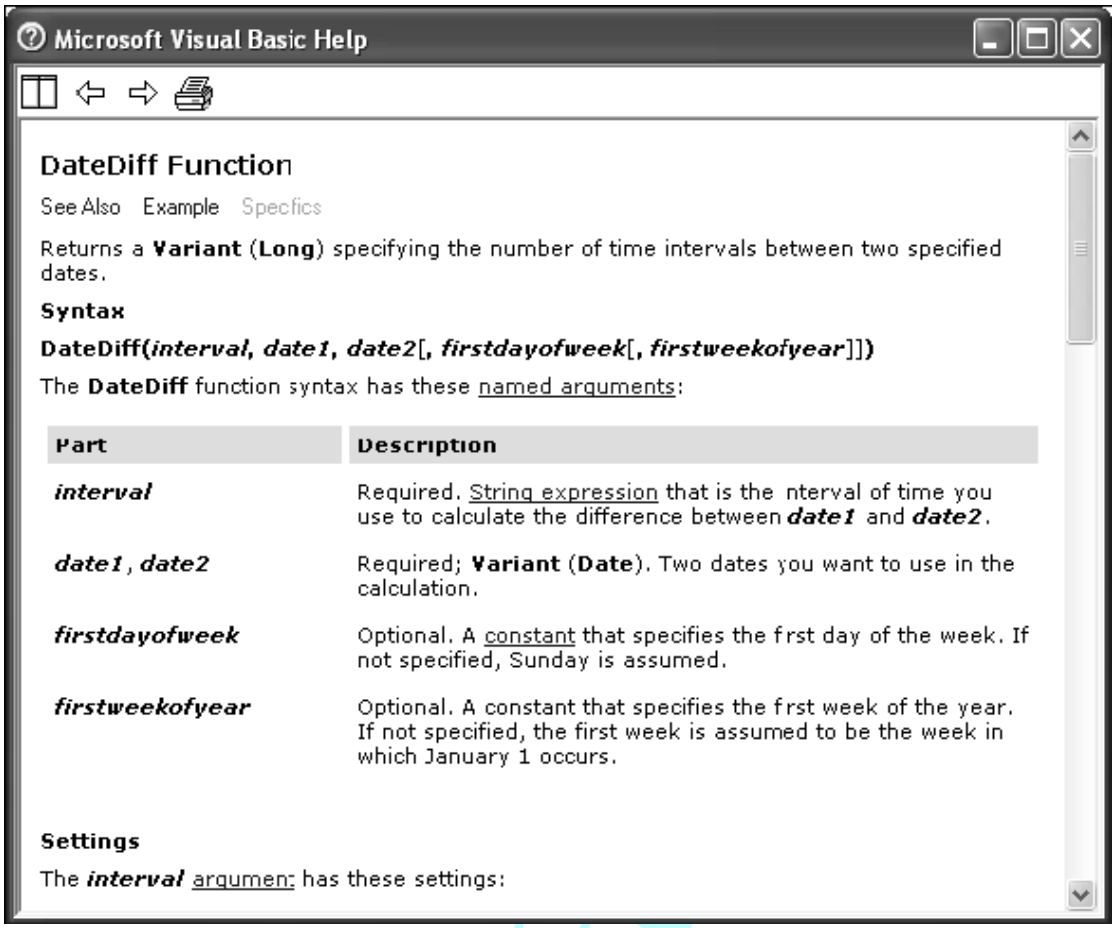
我们已经讨论了大量(尽管不是所有的)有用且通用的 VBA 调用。您可以用 VBA 中的对象浏览器来显示所有可用的 VBA 调用。本书后续部分会讨论对象浏览器。下图是一个对象浏览器的快照。如果您在引用复合框中选择 VBA，在类列表中选择 DateTime 类，再选中 DateDiff 成员就能看到这个快照。

China BDN



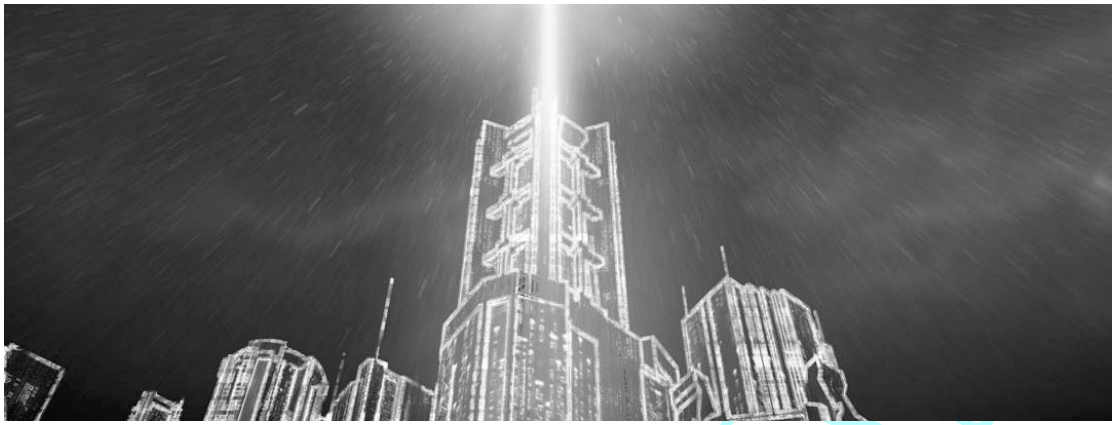
在对象浏览器中选中一个条目后，按**<F1>**键能获得更多的信息，有时甚至是样例代码。

China

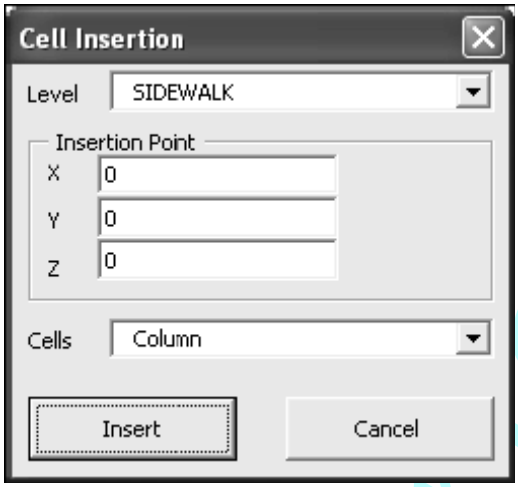


本章回顾

VBA 中内置了许多过程和函数。由于有了 Now 函数，您就不需要写取当前日期和时间的函数。类似地，您不需要写复杂的代码在注册表中存储应用信息，因为可以直接用 SaveSetting 和 GetSetting 过程。



10.可视界面

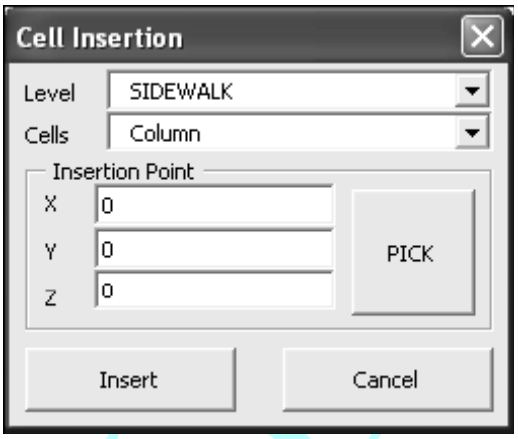


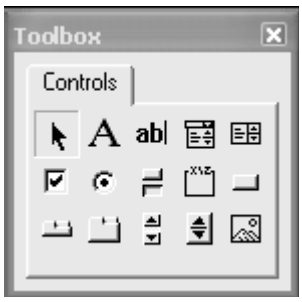
现在是开始与 VBA 的“Visual(可视)”打交道的时候了！让我们仔细看一下左侧所示的窗体。它由标签(Level 层、Cells 单元、X、Y 和 Z)、两个复合框(当前选中了 SIDEWALK 和 Column)、三个文本框(用于输入 X、Y 和 Z 值)以及两个标有 Insert(插入)和 Cancel(取消)的命令按钮组成。

如何建立良好的图形用户界面(GUI)是软件开发最具挑战的因素之一。谁都会把按钮随意放到窗体上，但要使界面用户友好和直观却需要花费思想、努力和借鉴别人的想法。

让我们讨论一下上图所示的窗体。其目的是让用户在指定点将一个单元插入到指定层上。哪一项应该出现在窗体的第一位呢？是单元名复合框吗？假如知道要插入哪个单元，也许它应该出现在第一位。如果单元列表依赖于所选择的层又会是什么样呢？这样做能避免把单元插入到错误的层上，如果是这样，那么层复合框或许应该在第一位。下面该考虑插入点了，或许插入点出现在层和单元复合框后是最好的。除了能手工输入插入点外，让用户也能拾取插入点会更好。让我们稍许改动一下界面吧。

这回看起来如何？好一点了吧？是的。我们已经安排好了可视化元素。在建立完基本界面后就开始编写界面背后的代码了，我们会稍后一些再进入这个主题。现在先让我们谈论一下能加入到用户窗体中的控件。





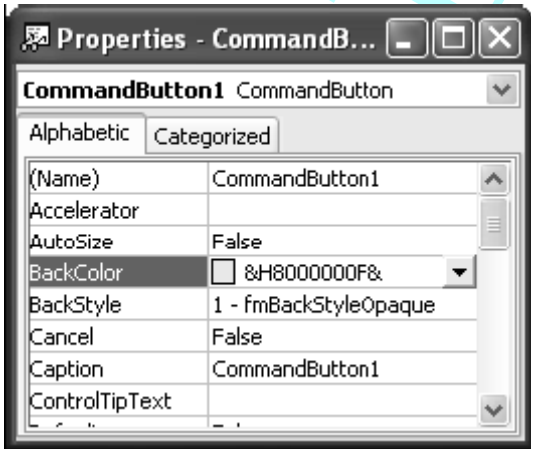
工具箱显示出了能放入窗体中的控件。除箭头外，所显示的每一个条目都是可用于界面设计的可视化元素。

属性、方法和事件

控件具有属性、方法和事件。属性描述控件的外观和或行为；方法让控件做某种事情。例如，用复合框的“AddItem”方法给其列表添加一项。当用户与 GUI 交互时会发生大多数的事件。例如，用户点击一个按钮后会触发该按钮的 click 事件并执行该事件下的任何代码。

属性

能在设计时(就是正在设计界面和编写代码的时候)或运行时(程序正在运行的时候)设置属性。用属性窗口可在设计时修改控件的属性。



属性窗口可被显示或隐藏。假如选择了一个命令按钮但没有看到属性窗口，可在该按钮上右击并从弹出的上下文菜单中选择“Properties(属性)”，或者直接按**<F4>**键。

在运行时，属性可如下设置：

```
CommandButton1.Enabled = True
```

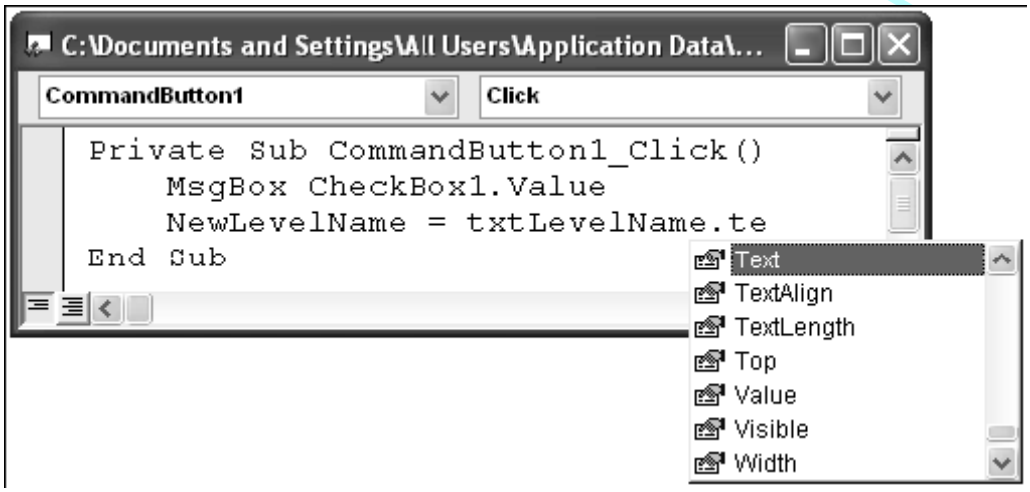
通过设置控件“CommandButton1”的 Enabled 属性为真使其有效。

也能在运行时取得属性值。

MsgBox CheckBox1.Value

NewLevelName = txtLevelName.Text

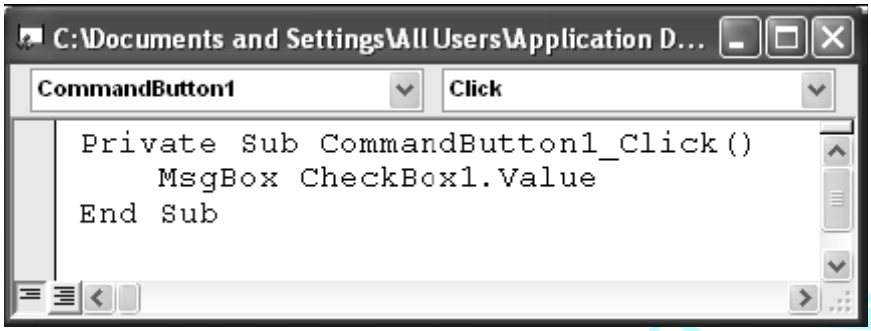
注意，我们先通过名字找到控件，又输入了一个点号，最后输入要操作的属性。在按了点号(.)键以后，VBA 能显示一个可用的属性和方法列表。



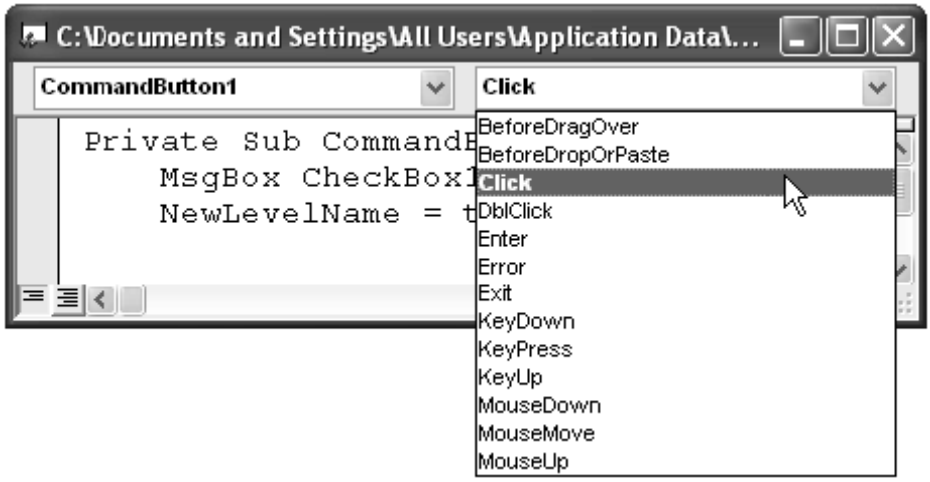
我在上例中键入了文本框名字 **txtLevelName** 和点键，助手窗口显示出可用的属性和方法。当您开始键入属性或方法名时，VBA 会自动在列表中选择第一个匹配项。当键入了“txtLevelName.te”后，如果按**<Enter>**键，VBA 会填补剩下的文字并将光标移动到下一行；如果按<Table>键，VBA 会填补剩下的文字并将光标移动到当前代码行的尾部。让 VBA 完成语句的输入能大大加快程序开发速度。

控件事件

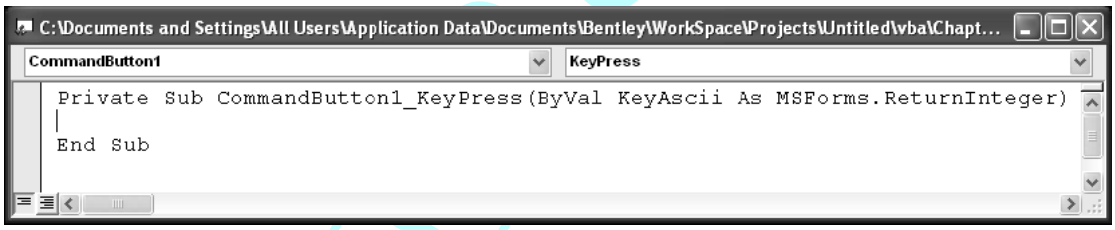
现在，让我们看一看如何使用控件的事件。我们在窗体的代码区为控件事件编写代码，窗体代码区看起来和代码模块一样，但有个微小的不同之处。插入到窗体中的所有控件都在左侧复合框中逐项列出，当在左侧复合框(下图选中的是**CommandButton1**)中选择一个控件后，就能在右侧复合框(下图选中的是 **Click** 事件)选择一个事件。



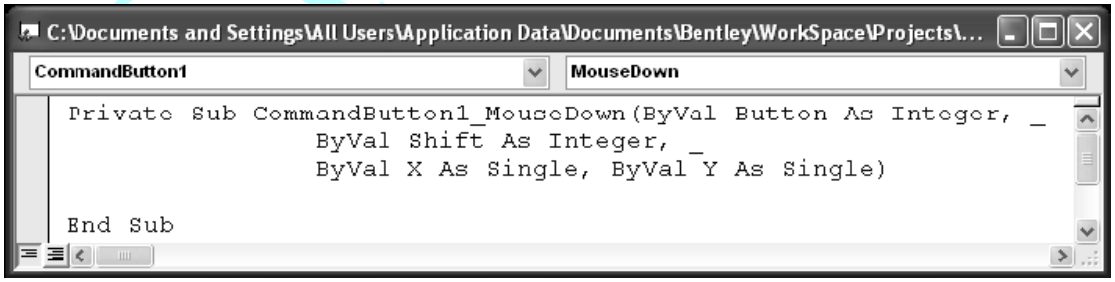
要操作其它事件，只需简单地下拉菜开过程复合框(右侧)并选择另一事件即可。正如下图中所看到的，可选事件相当多。



当选择了一个以前没有选择过的事件时，VBA 会为我们填补事件的框架。



这是一个 KeyPress 事件，当按了一个键后发生该事件。我们能考察某些事件传递的参数，如 KeyPress 事件传递的“KeyAscii”参数。把该参数作为变量了用来考察用户按下了哪个键。



这里的 MouseDown 事件说明了哪个鼠标键被按下(Button 参数)、鼠标键按

下时**<Shift>**、**<Ctrl>**和**<Alt>**键的状态(Shift 参数)以及鼠标键按下时鼠标的位置(X、Y 参数)。

事件参数除了向我们提供值以外，我们还可以修改它们。例如，在 KeyPress 事件内部将其 KeyAscii 参数赋值为 0(零)会使程序动作就像没有按下键盘一样。

通用控件属性

在单独讨论每个控件前，先来谈一谈几乎所有控件都具有的属性和事件。

Name(名称)

名称中有什么？我们通过名称来定位到控件，然后指定要取得或设置的属性，或者要使用的方法。名称和属性(或方法)用一个点号分割。请看：

Label1.Caption = “Enter Level Name:”

通过名字来调用控件、键入一个点号再键入要修改的属性“Caption”从而改变控件“Label1”的标题属性。当要设置一个属性的值时，“控件.属性”在等号的左边，要赋的值在右边，如上所示。要取得属性的值，就把“控件.属性”放在等号右边，在左边放一个变量，如下所示：

XVal = txtXValue.Text

变量 XVal 现在存有 txtXValue 控件的 Text 属性的值。

由于正在讨论的是控件名，所以应该对命名规则多说两句。控件名与变量名的规则一样，必须以字符开头，不能含空格等等。有些命名规则建议：文本框名应该以“txt”开头，标签名应该以“lbl”开头，复合框名应该以“cmb”开头等。正如变量命名一样，如果控件需要遵从一个命名规则，那就应该遵从它；如果不需要遵从，至少命名的控件要有意义。缺省情况下，控件被命名为像“TextBox1”、“TextBox2”、“TextBox3”等等这样。

Left(左)、Top(顶)

所有的控件都有一个 Left 和 Top 属性，这些属性规定了控件在窗体中的位置。窗体的左上角坐标是(0,0)，所以，如果一个文本框的 Left 值为 0 且 Top 值也为 0，则该文本框将在窗体的左上角出现。

Width(宽)、Height(高)

所有的控件都有 Width 和 Height 属性，这些属性确定了控件的大小。我们还能应当考虑所用控件的大小和形状，比如一个文本框就不应该宽 20、高 20。如果一个文本框被用来处理单行文本，那么把它的高设置为显示多于一行就没有多大的意义。另一方面，假如您想显示一个方形的命令按钮，就把它的宽和高属性设成一样。

Visible(可见)

为什么想要在窗体中放一个控件然后把它的 Visible 属性设置为假呢？控件就应该能看到，对吗？有时，一个控件的可见与不可见性是基于别的条件的。把控件的 Visible 属性设置为假使其在运行时不可见但在设计时仍可见。Visible 属性能在运行时按需更改为真和假。

Enabled(有效)

当把一个控件的 Enabled 属性设置为真时，可以在运行时与该控件交互。当 Enabled 为假时，控件转为灰色，用户不能与之交互。Enabled 属性不影响控件的可见性，仅影响控件的交互。

TabStop(制表符站)

运行时按**<Tab>**键能从一个控件移到下一个控件。如果控件的 TabStop 属性为真，该控件将能接收到光标聚焦；如果控件的 TabStop 为假，则该控件在用户按 Tab 键切换时将不能接收到聚焦。

TabIndex(制表符索引)

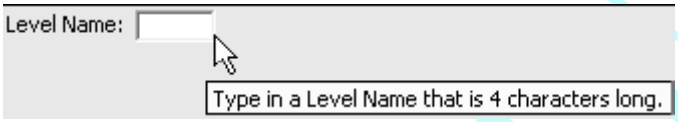
TabIndex 属性确定了您按<Tab>键从一个控件移到另一个控件时本控件接收聚焦的顺序。

Tag(附加信息)

Tag 属性没有专指，用它可以做你觉着合适的任何工作。一种用法是：把文本框的缺省值赋给 Tag，当用户点击“Load Defaults(装载缺省值)”按钮时，把 Tag 属性值赋给 Text 属性。

ControlTipText(控件提示文本)

完美的界面能提供给用户执行正确功能的控件，而且每次使用程序时不需要查阅用户手册。你可以在每个控件旁放上长的标签来详细解释该控件为何在此以及如何使用它，但这样使得界面乱糟糟。把鼠标放在控件上呆一到两秒，VBA 就会显示控件的提示文本。



这个例子显示了 ControlTipText 属性的值。有经验的用户知道层名应该是四个字符长，同时，新用户能从这个小的帮助中受益，知道应该如何输入。

通用控件属性了解得已经足够了。让我们每次谈论一个标准控件吧，突出其基本的属性、方法和事件，同时也显示它在工具箱中的图标。


标签

标签帮助用户知道要输入或选择什么。需要注意的属性如下：

Caption(标题)	显示给用户的文本
Font(字体)	显示标题所用的字体


文本框

文本框让用户输入文本或显示文本，常以单行模式出现，所以文本也显示在一行上。可以将文本框在垂直方向上拉伸以显示多行内容。

属性

Text(文本)	在文本框中显示的文本
Locked(锁定)	为“False”时，用户能在文本框中键入内容；为“True”时则不能。当设置为“True”时，虽然其中的文本不能被用户修改但能被选取并复制到 Windows 的剪切板中。锁定时代码也能修改 Text 属性。
MaxLength (最大长度)	能输入的最大字符个数。缺省值为 0，表示没有最大值。如果是在提问加拿大的邮政编码，可以把 MaxLength 属性设置

	为 6，允许输入“V4A5M2”但不允许输入“85302-1234”。
MultiLine(多行)和 WordWrap(字折行)	当为 True 时，文本框中的文本折到下一行。当为 False 时，文本在一行内滚动。
PasswordChar(密码字符)	为了在文本框中键入且不让其他人看到键入的内容而提供的密码字符。在该属性中输入一个星号(*)使得每次按下键盘键时出现一个星号。这能不让其他人看到您键入的内容(如果他们从我们的肩膀上看过来的话)，但安全性并不高。对于一个老练的开发者来说，用很少的代码就能找出密码字符背后的真正字符。

事件

KeyPress(按键)	给出在键盘上按下的字符的 ASCII 码。使用该事件能限制用户输入特定字符，如仅允许输入数字值。
KeyDown(键按下)、KeyUp(键抬起)	给出按下和抬起时的键盘键值，也包括**<Shift>**、**<Ctrl>**和**<Alt>**键的状态。例如，在 KeyDown 或 KeyUp 事件中知道是否按了**<F1>**，但在 KeyPress 事件中看不到**<F1>**键，因为它不是一个 ASCII 字符。这两个事件不同于 KeyPress 事件，KeyPress 事件给出的是所按的字符(A 或 a)。



复合框

复合框让用户下拉出一个项目列表来从中选择，或者基于 Style(风格)属性，用户还能在复合框中键入未列出的项。

属性

Text(文本)	在复合框中选中或输入的文本。
Style(风格)	0 = 组合(从列表中选择或在复合框中键入) 或 2 = 列表(用户必须从列表中选择)

ListIndex(列表索引)	复合框中选中项目的索引。 -1 = 什么也没有选择；0 = 选中了列表中的第一项； 1 = 选中了列表的第二项，以此类推。
ListCount(列表计数)	复合框中的项目个数。
ColumnCount (列计数)	列表里每行所含的列数。

方法

AddItem(加项)	向复合框中添加一个列表项。
RemoveItem (删项)	从复合框中删除一项。指定要删除的项的列表索引。
Clear	从复合框列表中清除所有项。

事件

Click(单击)	在复合框的一个项上单击时发生。
Change(改变)	当选中的项改变时发生。这与 Click 事件不同，因为还能用箭头键和键盘上其它的键来改变复合框中所选的项。



列表框

列表框允许从一个列表中选择一项或多项。复合框与列表框类似，但每次只能选择一项。当然了，列表框不支持“下拉”。

属性

Text	列表中所选项的文本。
MultiSelect	列表索引 – 列表中所选项的索引。
Selected	指定要检查的列表项索引，根据是否被选中而返回真或假。
ColumnCount	列表里每行所含的列数。

方法

AddItem	与复合框相同。
RemoveItem	与复合框相同。
Clear	与复合框相同。

事件

Click	与复合框相同。
Change	与复合框相同。



复选框

复选框让我们指定对一个项的选择。窗体中能含有多个复选框且它们之间的动作相互独立。设计一个比萨饼菜单模拟程序的话，就能用复选框来指定每种浇头(食物或菜上的一种点缀物)。

属性

Caption	复选框后面显示的文本。
TrippleState	True 或 False。True 时复选框的可能值为 True、False 或 Null。当 TrippleState(三态)为 False 时，可能的值不是 True 就是 False。
Value	True、False 或 Null。

事件

Click	当用户点击复选框时，其值被设置为真或假。当用户点击复选框且其值被设置为 Null(处于三态模式)时不触发 Click 事件。
Change	复选框的值发生变化。



选项按钮

当想让用户在几个可能项中作出一个选择时使用选项按钮，如在大、中或小

中来选择，可以使用三个选项按钮，每个选择一个选项按钮。

属性

Caption	选项按钮后面显示的文本。
GroupName (组名)	选项按钮成组工作，组中仅有一个能被选择。为了让用户在“大”、“中”、“小”中选大小和在“红”、“白”“蓝”中选颜色，需要为每组选项按钮起一个组名(GroupName)。

事件

Click	参见复选框的 Click 事件。
Change	参见复选框的 Change 事件。
DblClick(双击)	当用户双击控件时触发该事件。



切换按钮

切换按钮看起来像命令按钮但其动作更像复选框。选中该按钮时，它看上去凹了进去。切换按钮的一个典型应用就是用来指定字体是否为粗体、下划线或倾斜。

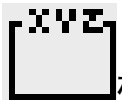
属性

Caption	切换按钮上显示的文本。
TrippleState	参见复选框。
Value	参见复选框。

事件

Click	参见选项按钮。
Change	参见选项按钮。
DblClick(双击)	参见选项按钮。

China BDN

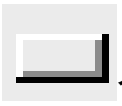


框架

框架是控件的容器，意思是控件能置于其中。如果框架的 Visible 属性设置为假，则其中的所有控件都不可见。移动框架时，其中的所有控件也都随之移动。使用框架来组织控件组。

属性

Caption	标题(Caption)显示在框架的左上角。
Visible	True 或 False。



命令按钮

命令按钮是用来让用户点击的，如(通用的一些标题)：“确定”、“取消”、“打印”、“打开”和“关闭”。

属性

Caption	按钮中显示的文本。
TakeFocusOnClick	确定当用户点击按钮时该按钮是接收聚焦还是保持聚焦在以前选择的控件上。

事件

Click	用户点击按钮时触发。



TabStrip(选项卡条)

用 TabStrip 表现“Tab”选择。虽然 TabStrip 与多页控件看起来很像，但不要把它们搞混了。TabStrip 不是控件容器，TabStrip 按钮是切换按钮和选项按钮交集的产物。在任何给定的时间内，TabStrip 中仅有一个 Tab 被选择。

属性

MultiRow(多行)	设为 True 时，Tab 个数超出 TabStrip 宽度的情况下将以多行显示。设为 False 时，所有 Tab 都在一行内显示。

SelectedItem	选择了哪个 Tab?
Style(风格)	0 = 选项卡式, 1 = 按钮式。风格控制 TabStrip 如何显示。
TabOrientation	选项卡条的位置，0 = 顶，1 = 底，2 = 左，3 = 右。

方法

Tabs.Add	用于向 TabStrip 中添加 Tab。

事件

Change	当前 Tab 变化时触发。



多页

多页控件是一个控件容器，其每一页都有自己的控件集合。右击一个 Tab(选项卡标签)并选择一个功能来添加、重命名、删除和排序选项卡。

属性

Value	0 = 第一页，1 = 第二页，等等。
Pages.Count	页的个数。
MultiRow	参见 TabStrip。

方法

Pages.Add	用于向多页控件中添加选项卡。

事件

Change	当前页变化时触发。



滚动条

滚动条能让用户选择并改变数字值。随值的改变在移动的那个矩形框叫做拇指(Thumb)。

属性

LargeChange	当用户点击滚动条区时 Value 属性的改变量。

China BDN

SmallChange	当用户点击滚动箭头时 Value 属性的改变量。
Min	当滚动条为水平时滚动条能具有的最小值。
Max	滚动条能具有的最大值。
Value	滚动条的数值。

事件

Change	点击拇指与箭头间的区域或箭头并释放拇指后改变了滚动条的值，此时触发该事件。
Scroll(滚动)	拖拽拇指时触发。例如，假如我们让用户输入一个数并在一个标签中显示该数，在拇指未被释放前标签值不会改变，此时也没有发生 Scroll 事件。Scroll 事件能让我们看到拇指从滚动条的一段拖拽到另一端时数字的变化过程。



旋转按钮

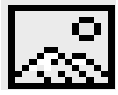
旋转按钮能让用户改变数值。它与滚动条类似但没有拇指。

属性

Delay(延迟)	用户按下按钮以后、值上下滚动变化之前经过的毫秒数。
Min	旋转按钮的最小值。
Max	旋转按钮的最大值。
SmallChange	用户点击按钮时值的改变量。
Value	旋转按钮的值。

事件

Change	值改变时触发该事件。
SpinUp	值增加时触发该事件。
SpinDown	值减少时触发该事件。



图像

用图像控件在界面中显示图像。可接受的文件格式是：.bmp、.gif、.jpg、.wmf

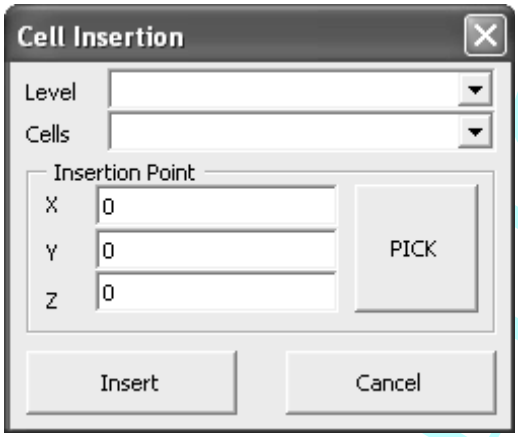
和.ico。

属性

BorderStyle	边框风格，0 = 无，1 = 单边框。
Picture	在图像控件中显示的文件。
PictureSizeMode	图像大小方式，0 = 剪切，1 = 拉伸，3 = 缩放。
PictureTiling	True 将图像控件中的图片平铺，False 不平铺。

用户界面练习

本章一开始讨论了建立一个有用且直观的界面的重要性，然后介绍了标准控件。现在让我们建立一些界面来演示刚说明过的属性、方法和事件。我们将在一个新的工程中建立一些新的窗体来实现这些演示。一开始先插入一个新窗体并操作之，完成后再插入另一个新窗体，如此继续下去。



这是您将操作的第一个界面，从添加最顶上的控件开始向下工作。

第一组控件是两个复合框，缺省情况下，它们以名字 **ComboBox1** 和 **ComboBox2** 插入，把它们的名字改为 **cmbLevels** 和 **cmbCells**。接下来在复合框的左侧插入两个标签，给这些标签的 Caption 属性赋值为“Level”和“Cells”。

下一段是位于一个框架中的一组控件。插入框架并将其标题改为“Insertion Point”，再插入文本框、标签和命令按钮，改变文本框名字为 txtX、txtY 和 txtZ，命名命令按钮为 cmdPick，标题为“Pick”。

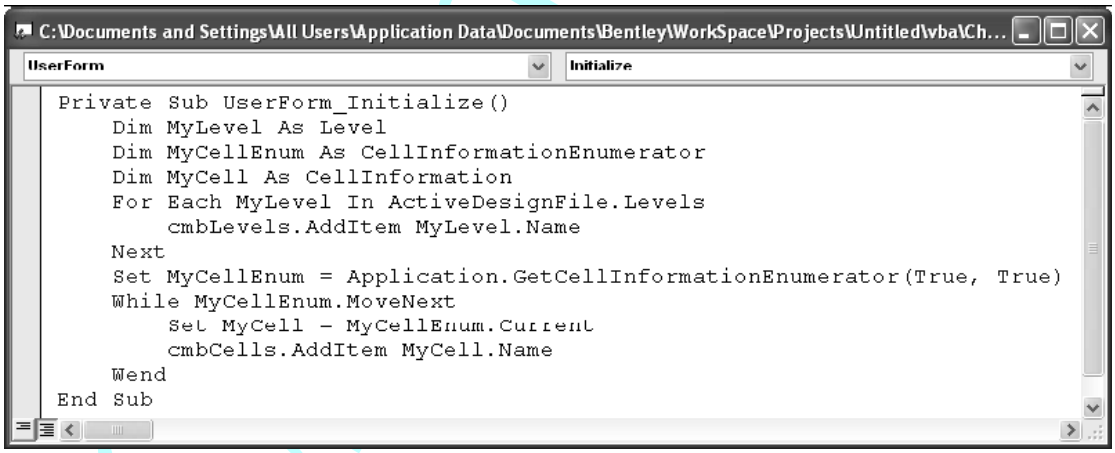
最后两个要加入的控件是命令按钮 cmdInsert 和 cmdCancel，它们的标题分别为“Insert”和“Cancel”。

在窗体中放入控件仅仅是个开始。Level 复合框需要用当前图形中所有的层来填充，Cells 复合框需要用所有有效的单元来填充。这些复合框要在用户看到窗体之前填充，要实现这个功能，需转到用户窗体的 Initialize(初始化)事件。

右击窗体，在弹出的菜单中选择 **View Code(查看代码)**。缺省情况下进入的是窗体的 Click 事件，在过程复合框中选 Initialize。

```
Private Sub UserForm_Initialize()
    Dim MyLevel As Level
    Dim MyCellEnum As CellInformationEnumerator
    Dim MyCell As CellInformation
    For Each MyLevel In ActiveDesignFile.Levels
        cmbLevels.AddItem MyLevel.Name
    Next
    Set MyCellEnum = _
        Application.GetCellInformationEnumerator(True, True)
    While MyCellEnum.MoveNext
        Set MyCell = MyCellEnum.Current
        cmbCells.AddItem MyCell.Name
    Wend
End Sub
```

下面是 UserForm 中 Initialize 事件的代码。



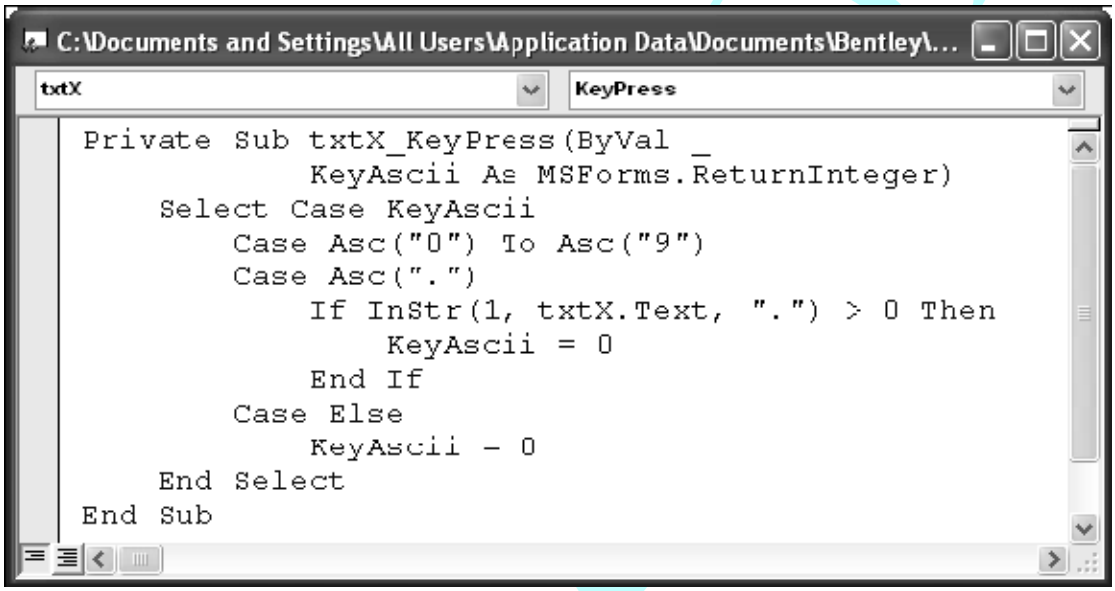
通过 AddItem 方法将 ActiveDesignFile 中的层名和单元名分别赋给 Level 和 Cells 复合框。

既然构成复合框的代码已经就位，则可以按**<F5>**来运行代码并确认一切正常。复合框应该含有层名和单元名。点击窗体右上角的“X”关闭窗体回到 VBA 中。

还需要对复合框进行加工。我们让用户选择层和单元但不想让用户在这两个复合框中键入任何内容。把复合框的 Style(风格)属性改为 2 – fmStyleDropDownList 就能实现这一点。

下一步是编写代码让文本框内只能输入数字。这是靠操作文本框的 KeyPress 事件完成的。

右击最顶上的那个文本框再选择 **View Code(查看代码)**。缺省情况下进入的是文本框的 Change 事件，在过程复合框中选 KeyPress 事件。



KeyPress 事件传递给我们“KeyAscii”，该参数告诉我们用户所按字符的 ASCII 值。如果 ASCII 码在数字 0 到 9 之间，代码不对 KeyAscii 参数作任何处理，这会让输入的值进入文本框，这些值和键入时的一样。

下面看一看对小数点符号的处理情况。如果文本框中已经有了一个小数点，就把 KeyAscii 设置为零以阻止输入另一个小数点；如果文本框中没有小数点，就不对 KeyAscii 参数作任何处理，这样，小数点就能被加入进来。

如果遇到任何其它的 KeyAscii 值，就设置 KeyAscii 参数为零(0)，这会使事件认为什么键也没有按下。

需要把同样的代码放入 txtY 和 txtZ 控件的 KeyPress 事件中。简单地复制和粘贴代码，然后作个小的改动就行了。当查找小数点符号时要用到控件名，所以复制和粘贴完 Select Case 代码后，把 InStr 函数中的控件名改成与 KeyPress 事件中的控件一致。

Private Sub txtX_KeyPress(ByVal KeyAscii As MSForms.ReturnInteger)

```
Select Case KeyAscii
    Case Asc("0") To Asc("9")
    Case Asc(".")
        If InStr(1, txtX.Text, ".") > 0 Then
            KeyAscii = 0
        End If
    Case Else
        KeyAscii = 0
    End Select
End Sub
Private Sub txtY_KeyPress(ByVal KeyAscii As MSForms.ReturnInteger)
Select Case KeyAscii
    Case Asc("0") To Asc("9")
    Case Asc(".")
        If InStr(1, txtY.Text, ".") > 0 Then
            KeyAscii = 0
        End If
    Case Else
        KeyAscii = 0
End Select
End Sub
Private Sub txtZ_KeyPress(ByVal KeyAscii As MSForms.ReturnInteger)
Select Case KeyAscii
    Case Asc("0") To Asc("9")
    Case Asc(".")
        If InStr(1, txtZ.Text, ".") > 0 Then
            KeyAscii = 0
        End If
    Case Else
        KeyAscii = 0
```

China BDN

```
End Select
End Sub
```

下一步处理 Cancel(取消)按钮。当用户点击 Cancel 按钮时，应该关闭窗体。我们以前都是通过右击控件选择 **View Code** 来进入控件事件的，现在请双击 Cancel 按钮，这是另一种进入窗体代码区的方法。

```
Private Sub cmdCancel_Click()
    Unload Me
End Sub
```

关闭窗体仅需要一行代码。

以上是一些较容易的功能实现，现在有些较难的功能要做。当用户点击命令按钮“Insert”时，我们需要在选择的层、输入的点处插入选择的单元。

```
Private Sub cmdInsert_Click()
    If cmbLevels.Text = "" Then
        MsgBox "Please select a level."
        Exit Sub
    End If
    If cmbCells.Text = "" Then
        MsgBox "Please select a cell."
        Exit Sub
    End If
    Dim InsPt As Point3d
    Dim CellElem As CellElement
    InsPt.X = CDbl(txtX.Text)
    InsPt.Y = CDbl(txtY.Text)
    InsPt.Z = CDbl(txtZ.Text)
    Set CellElem = CreateCellElement3(cmbCells.Text, InsPt, True)
    CellElem.Level = ActiveDesignFile.Levels(cmbLevels.Text)
    ActiveModelReference.AddElement CellElem
End Sub
```

China BDN

插入单元前先确保用户选择了一个层和一个单元。用 **txtX**、**txtY** 和 **txtZ** 文本框取得单元原点 X、Y 和 Z 值。建立单元后设置单元的层为 cmbLevels 复合框的 Text 属性值。最后一件事是把元素加入当前模型中。

还剩下一个按钮“PICK”，它用于选择单元原点。我们想让它干什么？这个按钮应该用于让用户在 MicroStation 中选择一个点而不是手工输入 X、Y 和 Z 值。为了让程序工作得更好，如果用户已经选择了层和单元，将会在选择点自动插入单元。这样做省去了用户在点击完“PICK”按钮后还需要点击“Insert”按钮。

简而言之，我们想让用户在 MicroStation 中点击。但缺省条件下会出问题，因为 VBA 窗体是模态的(modal)。也就是说，窗体激活后将阻止用户与 MicroStation 交互直到窗体被卸载为止。要解决这一潜在的问题，需要以非模态(modeless)方式显示窗体。

```
Sub DoCellInsertion()
    frmCellInsertion.Show vbModeless
End Sub
```

把“DoCellInsertion”放入一个代码模块，我们将在那里启动显示窗体的命令。

对于“PICK”命令按钮，我们想让用户拾取一个点，在层和单元复合框都不空的条件下，在选择的层上、在选择的点处插入选择的单元。

```
Private Sub cmdPick_Click()
    Dim MyMsg As CadInputMessage
    Dim MyQue As CadInputQueue
    Dim SelPt As Point3d
    Dim CellElem As CellElement
    On Error GoTo errhnd
    Set MyQue = Application.CadInputQueue
    Do
        Set MyMsg = MyQue.GetInput
        Select Case MyMsg.InputType
            Case msdCadInputTypeDataPoint
                SelPt = MyMsg.Point
```

```
                txtX.Text = SelPt.X
                txtY.Text = SelPt.Y
                txtZ.Text = SelPt.Z
                If cmbLevels.Text <> "" And cmbCells.Text <> "" Then
                    Set CellElem = _
                    CreateCellElement3(cmbCells.Text, SelPt, True)
                    CellElem.Level = _
                    ActiveDesignFile.Levels(cmbLevels.Text)
                    ActiveModelReference.AddElement CellElem
                End If
                Exit Do
            Case Else
                Exit Do
        End Select
    Loop
    Exit Sub
errhnd:
    Err.Clear
End Sub
```

让我们慢慢地通读这段代码。首先声明了一些变量，这很简单；其次开始监听输入队列，如果用户拾取了一个点，就做如下动作：

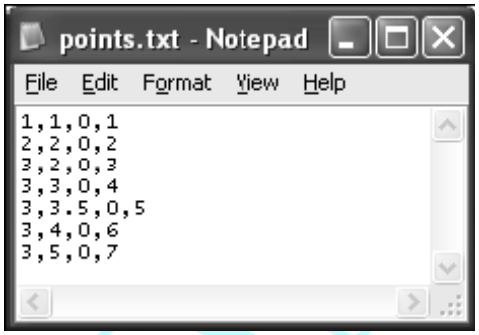
- 把选择的点元素的 X、Y 和 Z 值放入三个文本框中；
- 如果层复合框和单元复合框都不空，就在选择的点插入选择的单元，然后改变其 Level 属性以反映选择的层。

如果有其它输入或发生错误就退出该过程。

本书配套的 CD 上有名字为 Chapter10.mvba 的 VBA 工程，里面有所有以上介绍的代码。

点列表读取器

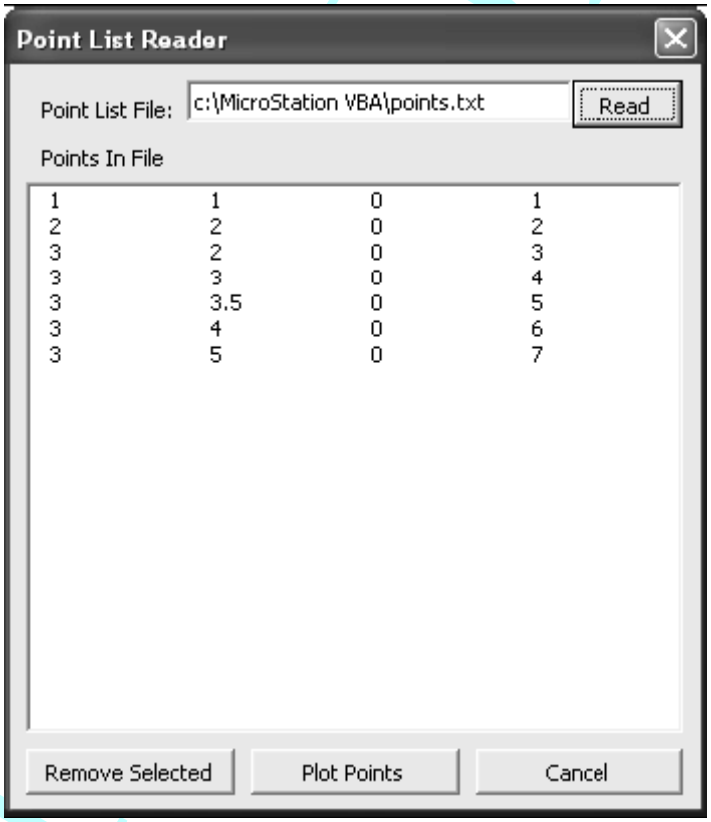
本程序集中于列表控件。用 AddItem 方法向列表框添加条目，然后用 List 属性在列表框的其它列放置值，使用“Remove”让用户手工从列表框中移去条目。



我们读一个文本文件取得点，再把这些点放入界面中的列表框里。文本文件如右图所示：

文本文件中的每行都给出了文本插入点 X、Y、Z，还有在 X、Y、Z 点要放的标签。

右图就是界面。



我们使用两个标签、一个叫 **txtPointFile** 的文本框、一个叫 **lstPoints** 的列表框，四个分别叫做 **btnRead**、**btnRemove**、**btnPlotPoints** 和 **btnCancel** 的命令按钮。

缺省情况下，列表框仅使用一列。我们想要四列，所以设置 ColumnCount 属性为 4。在 ColumnWidths 属性中指定每个列的宽度为“60 pt;60 pt;60 pt;60 pt”。把最后一个属性 MultiSelect 设置为“2 - fmMultiSelectExtended”，设置这个属性后就能让用户在列表框中选择多个条目，多选条目的方法是在列表框中点击条目的同时按住 **<Shift>**和**<Ctrl>**键。

先从 Read 按钮开始考察代码。

```
Private Sub btnRead_Click()
    Dim PointText As String
    Dim PointSplit As Variant
```

```
    Dim FFile As Long
    FFile = FreeFile
    Open txtPointFile.Text For Input As #FFile
    While EOF(FFile) = False
        Line Input #FFile, PointText
        If PointText <> "" Then
            PointSplit = Split(PointText, ",")
            lstPoints.AddItem PointSplit(0)
            lstPoints.List(lstPoints.ListCount - 1, 1) = PointSplit(1)
            lstPoints.List(lstPoints.ListCount - 1, 2) = PointSplit(2)
            lstPoints.List(lstPoints.ListCount - 1, 3) = PointSplit(3)
        End If
    Wend
End Sub
```

当用户点击 Read 按钮时，我们打开由文本框 txtPointFile 指定的文件用于输入(意思是将读取该文件)。由于未到达文件尾，就从文件中读出一行，分割一行的内容为独立的元素再把这些元素加入到列表框中。注意是如何用 AddItem 把点的 X 分量加入到列表框中的，AddItem 仅用于添加列表框第一列中的条目。每个附加列的值是通过 List 属性设置的，在使用 List 时，指定行索引和列，然后给出想放入该列的值。

Remove 按钮的意思是移除列表框中选择的任何条目，因为一次能选择多个条目，所以移除条目时要小心。

```
Private Sub btnRemove_Click()
    Dim I As Long
    For I = lstPoints.ListCount To 1 Step -1
        If lstPoints.Selected(I - 1) Then
            lstPoints.RemoveItem I - 1
        End If
    Next I
End Sub
```

从列表框的最后一项开始到第一项，可避免移去选择项时出现潜在的问题。

```
Private Sub btnPlotPoints_Click()
    Dim TextIns As Point3d
    Dim TextVal As String
    Dim I As Long
    Dim PT As TextElement
    Dim RotMat As Matrix3d
    For I = 1 To lstPoints.ListCount
        TextIns.X = lstPoints.List(I - 1, 0)
        TextIns.Y = lstPoints.List(I - 1, 1)
        TextIns.Z = lstPoints.List(I - 1, 2)
        Set PT = Application.CreateTextElement1(Nothing, _
                lstPoints.List(I - 1, 3), TextIns, RotMat)
        ActiveModelReference.AddElement PT
    Next I
End Sub
```

btnPlotPoints 按钮考察列表中的每一项，从中取得文本原点坐标 X、Y 和 Z 和要显示的文本。

当用户点击 Cancel 按钮时执行如下代码：

```
Private Sub btnCancel_Click()
    Unload frmPointList
End Sub
```

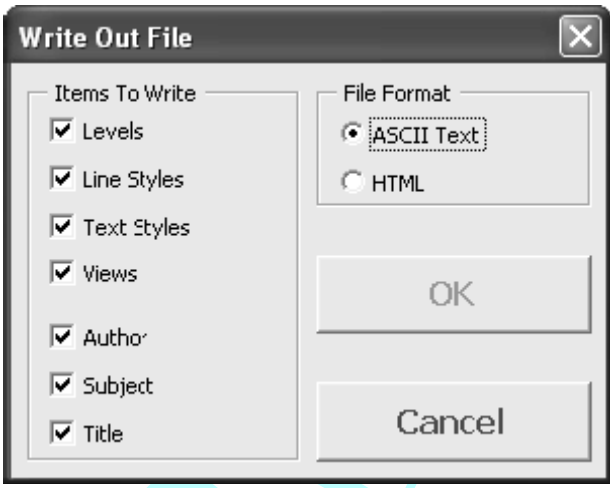
以上是窗体中的按钮部分。那么，首先是如何显示窗体呢？在代码模块中放入如下代码：

```
Sub DoPointListReader()
    frmPointList.Show
End Sub
```

现在，宏 DoPointListReader 用于显示窗体以及执行我们放入其中的所有功能。

China BDN

写出到文件



我们将要编写的下一个宏使用复选框和选项按钮。这里是其可视化界面：

有七个复选框，名字为：**chkLevels**、**chkLineStyles**、**chkTextStyles**、**chkViews**、**chkAuthor** 、 **chkSubject** 和 **chkTitle** 。有两个选项按钮：**optASCII** 和 **optHTML**。还有两个命令按钮：**cmdOK**、**cmdCancel**。

我们想让用户选择任意条目写出到文件，复选框正适合于此种情况。

对于文件格式，应该仅有一个选择，所以用了选项按钮。

在进入代码前，一定要明白本程序不像前两个那样简单，让我们做个规划。

我们想写出两种文件格式：ASCII 和 HTML。这仅满足今天的需要，明天将会怎样呢？在我们开发具有可扩展性的应用程序时，应该考虑将来的使用。我们能将所有的代码都放入 comdOK 按钮的 Click 事件中，然而，把代码分成更多可管理的块能使将来添加文件格式时更容易。

让我们考察一下写文件的三个部分的过程，文件的三个部分是：页眉、内容行、页脚。

```
Sub PrintHeader(HeaderIn As String, FileNum As Long, _
                        Optional Columns As Long = 1)
    If optASCII.Value = True Then
        Print #FileNum, "[" & HeaderIn & "]"
    ElseIf optHTML.Value = True Then
        Print #FileNum, "<table width=660>"
        Print #FileNum, "<tr><td colspan=" & Columns & _
                " align=center><b>" & HeaderIn & "</td></tr>"
    End If
End Sub
```

现在用 If 和 ElseIf 语句处理两种文件格式，将来可以用另一个 ElseIf 语句添加另一种文件格式。

```
Sub PrintLine(LineIn As String, FileNum As Long)
    If optASCII.Value = True Then
        Print #FileNum, LineIn
    ElseIf optHTML.Value = True Then
        Dim XSplit As Variant
        Dim I As Long
        XSplit = Split(LineIn, vbTab)
        Print #FileNum, "<tr>"
        For I = LBound(XSplit) To UBound(XSplit)
            Print #FileNum, vbTab & "<td>" & XSplit(I) & "</td>"
        Next I
        Print #FileNum, "</tr>"
    End If
End Sub
```

对每个选择的条目都使用过程 PrintLine 进行处理。使用 If 和 ElseIf 语句区分文件格式。

```
Sub PrintFooter(FileNum As Long)
    If optHTML.Value = True Then
        Print #FileNum, "</table>" & vbCrLf
    End If
End Sub
```

仅当选择了 HTML 选项时才需要打印页脚。

现在来看 cmdOK 按钮的 Click 事件。在 Click 事件中有两部分：第一部分设置导出；第二部分是一系列的 If … Then 语句，每个语句直接关联到一个复选框。下面是代码：

```
Private Sub cmdOK_Click()
    Dim MyFile As String
    Dim FFile As Long
```

```
Dim myLevel As Level
Dim myLStyle As LineStyle
Dim myTStyle As TextStyle
Dim myView As View
FFile = FreeFile
If optASCII.Value = True Then
    MyFile = "c:\output.txt"
ElseIf optHTML.Value = True Then
    MyFile = "c:\output.htm"
End If
Open MyFile For Output As #FFile
PrintHeader "FILE NAME", FFile, 1
PrintLine ActiveDesignFile.FullName, FFile
PrintFooter FFile

If chkLevels.Value = True Then
    PrintHeader "LEVELS", FFile, 3
    For Each myLevel In ActiveDesignFile.Levels
        PrintLine myLevel.Name & vbTab & _
                  myLevel.Description & vbTab & _
                  myLevel.ElementColor, FFile
    Next
    PrintFooter FFile
End If

If chkLineStyles.Value = True Then
    PrintHeader "LINE STYLES", FFile, 2
    For Each myLStyle In ActiveDesignFile.LineStyles
        PrintLine myLStyle.Name & vbTab & _
                  myLStyle.Number, FFile
```


China BDN

```
        Next
        PrintFooter FFile
    End If

    If chkTextStyles.Value = True Then
        PrintHeader "TEXT STYLES", FFile, 3
        For Each myTStyle In ActiveDesignFile.TextStyles
            PrintLine myTStyle.Name & vbTab & _
                      myTStyle.Color & vbTab & _
                      myTStyle.BackgroundFillColor, FFile
        Next
        PrintFooter FFile
    End If

    If chkViews.Value = True Then
        PrintHeader "VIEWS", FFile, 5
        For Each myView In ActiveDesignFile.Views
            PrintLine myView.Origin.X & vbTab & _
                      myView.Origin.Y & vbTab & _
                      myView.Origin.Z & vbTab & _
                      myView.CameraAngle & vbTab & _
                      myView.CameraFocalLength, FFile
        Next
        PrintFooter FFile
    End If

    If chkAuthor.Value = True Then
        PrintHeader "AUTHOR", FFile
        PrintLine ActiveDesignFile.Author, FFile
        PrintFooter FFile
```

```
        End If

        If chkSubject.Value = True Then
            PrintHeader "SUBJECT", FFile
            PrintLine ActiveDesignFile.Subject, FFile
            PrintFooter FFile
        End If
        If chkTitle.Value = True Then
            PrintHeader "TITLE", FFile
            PrintLine ActiveDesignFile.Title, FFile
            PrintFooter FFile
        End If
        Close #FFile
    End Sub
```

最后再看一下最容易的事件。

```
    Private Sub cmdCancel_Click()
        Unload frmWriteDgnSettings
    End Sub
```

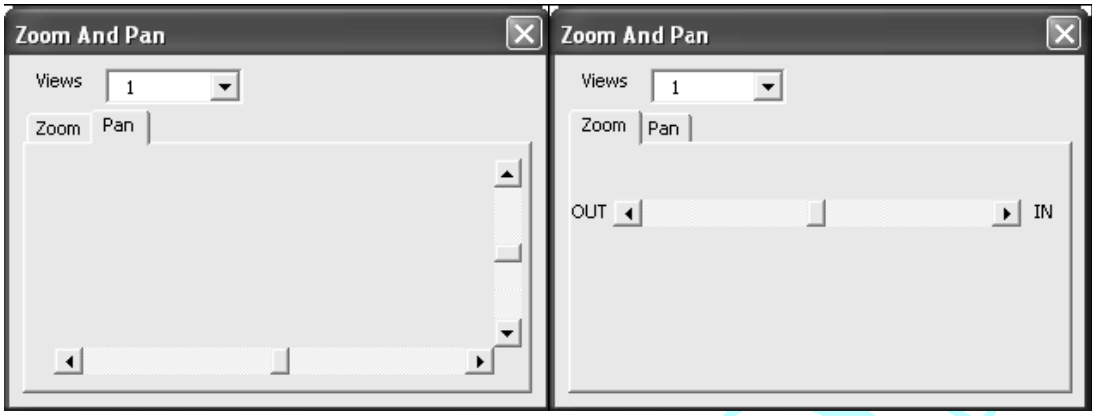
把下列过程加入到一个代码模块中来显示窗体。

```
    Sub DoWriteOutFile()
        frmWriteDgnSettings.Show
    End Sub
```

缩放和平移

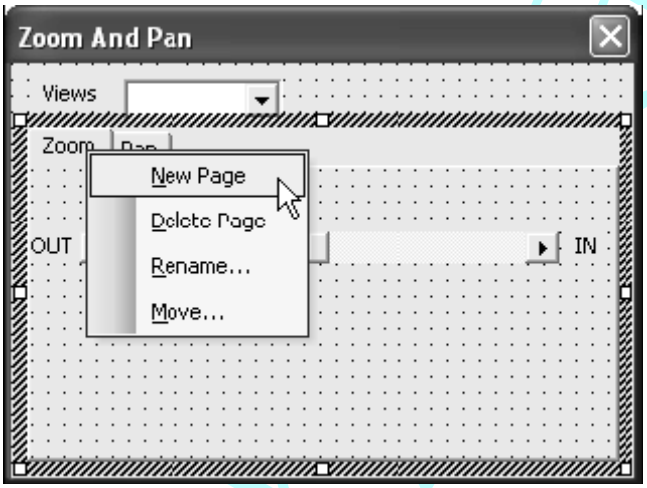
这是一个实时交互式平移和缩放当前设计文件中视图的小程序。

China BDN



该程序中使用了多页控件，多页控件有多个选项卡，每个选项卡有独立的界面。还使用了一些标签、一个复合框和三个滚动棒，滚动棒的最小值为-500，最大值为 500。窗体初始化时，用视图索引填充复合框，还设置了平移滚动棒的初始值。

当在多页控件一个现有的页上右击时，能添加页(选择“New Page(新建页)”)、对页改名、删除页或移动页的顺序。



这是在已经存在的一页上执行右击时多页控件的样子。

让我们先从用户窗体的初始化事件开始。

```
Private Sub UserForm_Initialize()
    Dim ViewCen As Point3d
    Dim MyView As View
    For Each MyView In ActiveDesignFile.Views
        cmbViews.AddItem MyView.Index
    Next
```

```
    cmbViews.ListIndex = 0
    ViewCen = ActiveDesignFile.Views(1).Center
    scrX.Value = ViewCen.X
    scrY.Value = ViewCen.Y
End Sub
```

这就是用户窗体的初始化事件。我们把每个视图索引都加入到了复合框 **cmbViews** 中。通过把 ListIndex 属性值赋为 0 来选择第一个元素。最后一步是取得视图 1 的当前中心点并把其 X 和 Y 值赋给滚动棒的 **srcX** 和 **srcY**。

滚动棒有两个事件需要处理：第一个是 Change 事件，除拇指正被滚动外每次滚动棒值的变化都会触发该事件；第二个是 Scroll 事件，当拇指在最小值和最大值之间托动时被触发。

我们将建立两个过程来执行缩放和平移操作：

```
Sub SetZoom(ZoomValue As Long, OldZoomValue As Long)
    ActiveDesignFile.Views(cmbViews.Text).Zoom 1 + _
                    (ZoomValue - OldZoomValue) / 100
    ActiveDesignFile.Views(cmbViews.Text).Redraw
End Sub
Sub SetPan(XPan As Long, YPan As Long)
    Dim ViewOrigin As Point3d
    ViewOrigin.X = XPan
    ViewOrigin.Y = YPan
    ViewOrigin.Z = 0
    ActiveDesignFile.Views(cmbViews.Text).Center = ViewOrigin
    ActiveDesignFile.Views(cmbViews.Text).Redraw
End Sub
```

当使用视图的 Zoom(缩放)方法时，大于 1 的数表示放大，小于 1 的数表示缩小。Zoom 方法的缩放是相对的，如果以系数 1.1 缩放三次，视图每次都被放大。从当前值中减去前一个值再除以 100，把结果值加 1 作为滚动棒左右移动时放大缩小的量。执行完缩放动作后，发出一个 Redraw(重绘)方法来看缩放的结

果。

通过调整视图的中心来执行平移。

我们已经看了用于放大和缩小的代码，现在来考察一下调用这些过程的事件。

```
Private Sub scrZoom_Change()
    SetZoom scrZoom.Value, scrZoom.Tag
    scrZoom.Tag = scrZoom.Value
End Sub
Private Sub scrZoom_Scroll()
    SetZoom scrZoom.Value, scrZoom.Tag
    scrZoom.Tag = scrZoom.Value
End Sub
```

scrZoom 滚动棒的 Change 和 Scroll 事件如上所示，这些事件中的代码是相同的。Tag 属性(如前讨论的那样)在这里为我们所用，这里就是使用它的一种方法：用 Tag 保存前一个值，调用 SetZoom 后，再把当前值设置到 Tag 中。

现在讨论一下平移。我们使用了两个滚动棒来设置视图中心的 X 和 Y 分量。

```
Private Sub scrX_Change()
    SetPan scrX.Value, scrY.Value
End Sub
Private Sub scrX_Scroll()
    SetPan scrX.Value, scrY.Value
End Sub
Private Sub scrY_Change()
    SetPan scrX.Value, scrY.Value
End Sub
Private Sub scrY_Scroll()
    SetPan scrX.Value, scrY.Value
End Sub
```

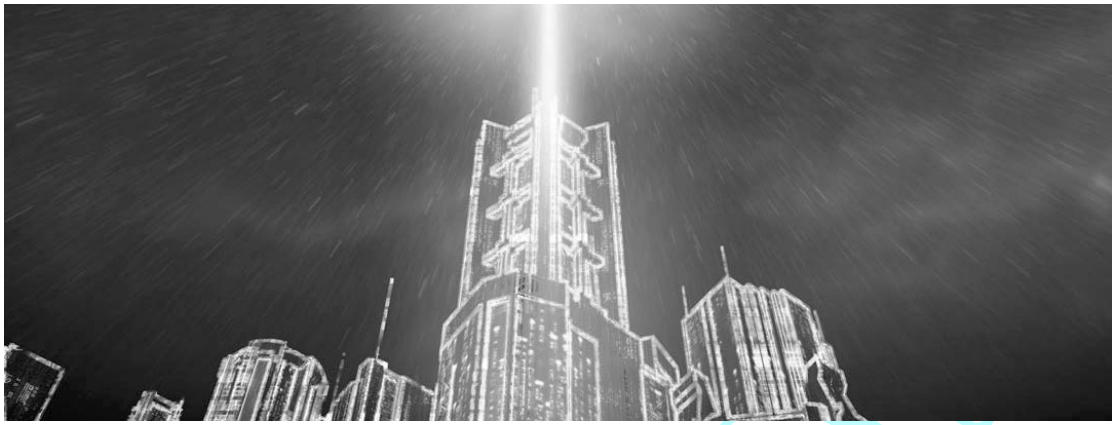
China BDN

本章回顾

随着我们更多地学习 MicroStation VBA，本书其余章节的许多地方都还会用到用户界面。记住下面几点：

- 所有控件都由属性、方法和事件；
- 访问控件的属性和方法的格式是：控件名，一个点号，属性或方法，需要时提供参数；
- 程序运行时，用户与界面间的交互触发事件；
- 用 Show 方法显示用户窗体；
- 在窗体显示前，用 Initialize 事件设置值和增加控件。

China BDI



11. MicroStation 对象模型—对象

对象是大多数 VBA 编程的基础。对象模型是对象的层次结构。本章没有分析所有的对象，而是先考察操作 MicroStation 对象模型的工具。在考察完工具后，再来考察使用 MicroStation VBA 时经常用到的一些对象。

- 对象浏览器
- 自动列出成员
- MicroStation VBA帮助文件
- 添加监控
- MicroStation对象模型

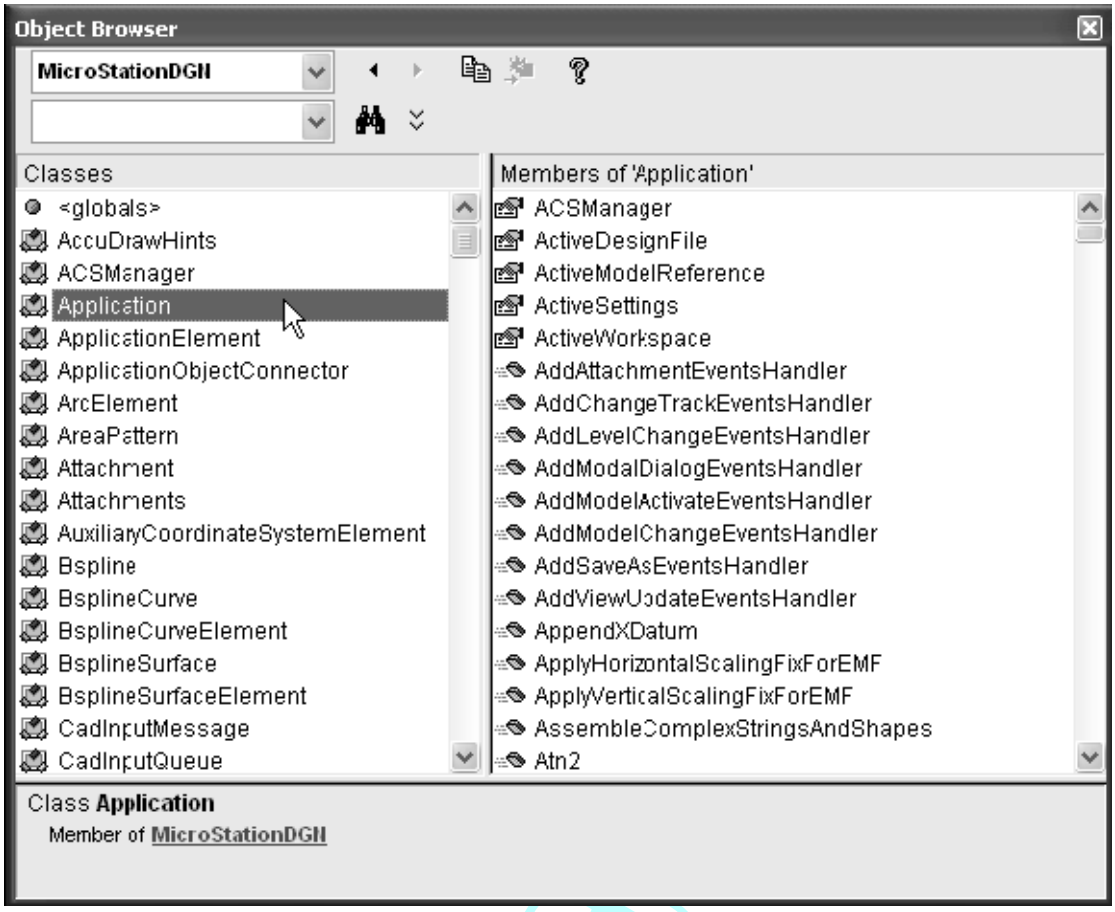
对象浏览器



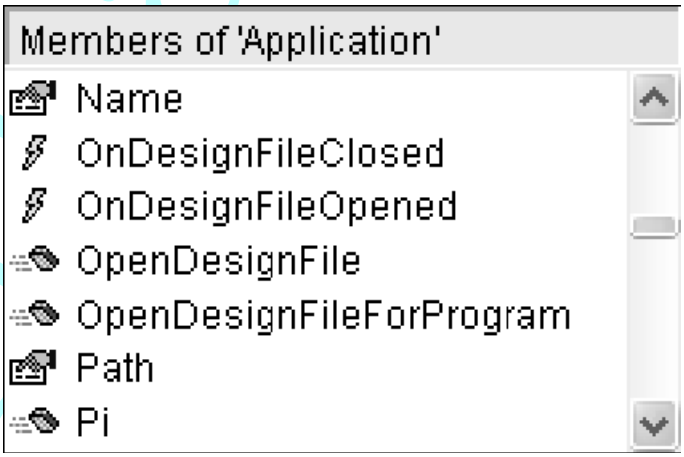
操作对象模型的最佳工具之一是对象浏览器。

点击对象浏览器工具栏按钮来显示对象浏览器。对象浏览器还能通过 VBA 菜单 **View > Object Browser(视图 > 对象浏览器)**或按键盘上的**<F2>**键来显示。

China



对象浏览器在顶部有两个复合框。最顶上的复合框能让我们缩小类的范围到一个指定的库。上图选中的是 MicroStationDGN 库。所显示的类都属于 MicroStationDGN 库。



当在类列表框中选择“Application”时，成员列表框中将显示出“Application”的成员。成员列表框能显示所选类的属性、方法和事件。

成员列表框中显示三种基本的成员类型。

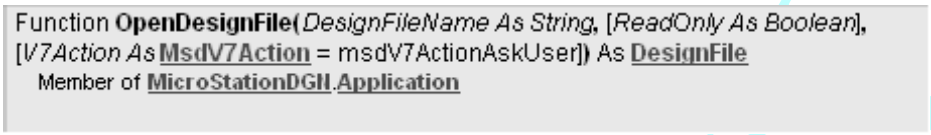
第一种是**属性**。“Name”和“Path”都是 Application 对象的属性。

方法“OpenDesignFile”、“OpenDesignFileForProgram”和“Pi”属于 Application 对象。

事件“OnDesignFileClosed”和“OnDesignFileOpened”也属于 Application

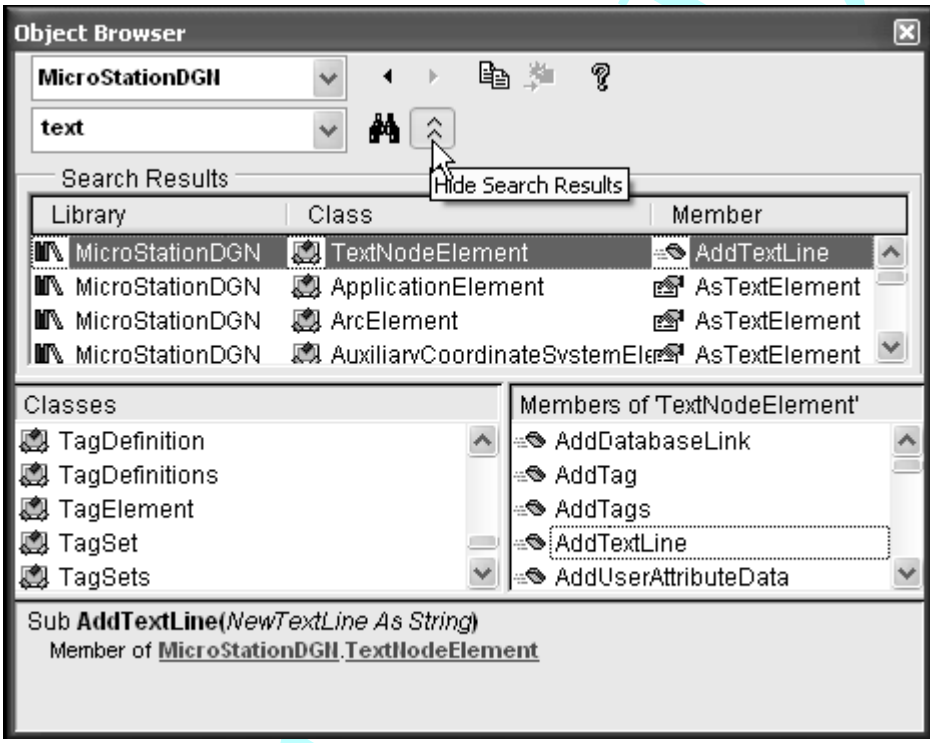
对象。

当在列表中选择了一个成员后，该成员的声明就会显示在对象浏览器的底部。



声明中显示了函数或过程的参数，还有函数的返回值类型。

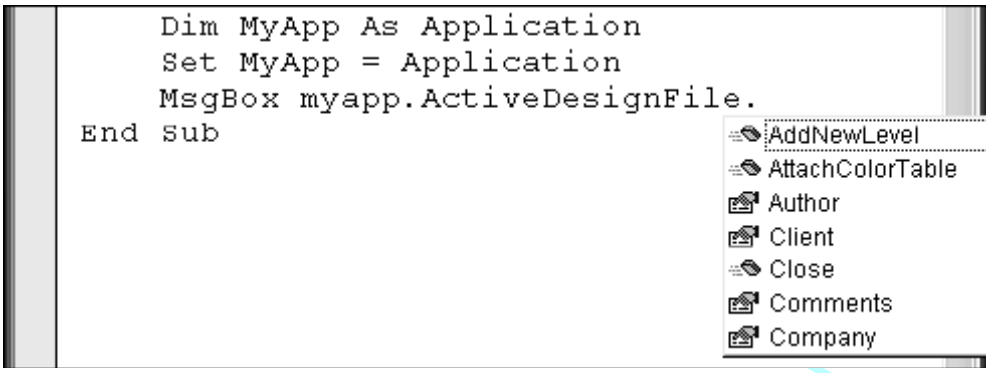
除了点击熟悉的类和成员外，还能用对象浏览器搜索对象模型。



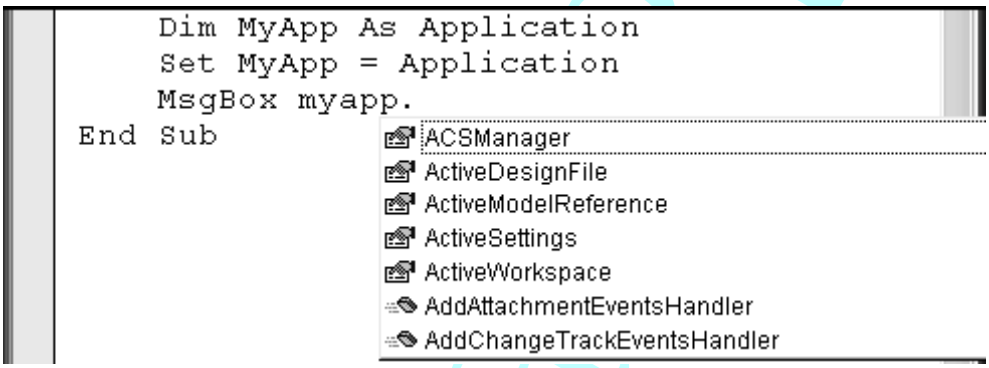
注意对象浏览器中 Hide/Show Search Results(隐藏/显示搜索结果)按钮上的光标。在 MicroStationDGN 类型库中对“text”的搜索会出现大量结果。所以，如果不知道想要的类或成员时，可以用对象浏览器来搜索它。

自动列出成员

当我们在设计中书写代码时，VBA 能给我们提供帮助。



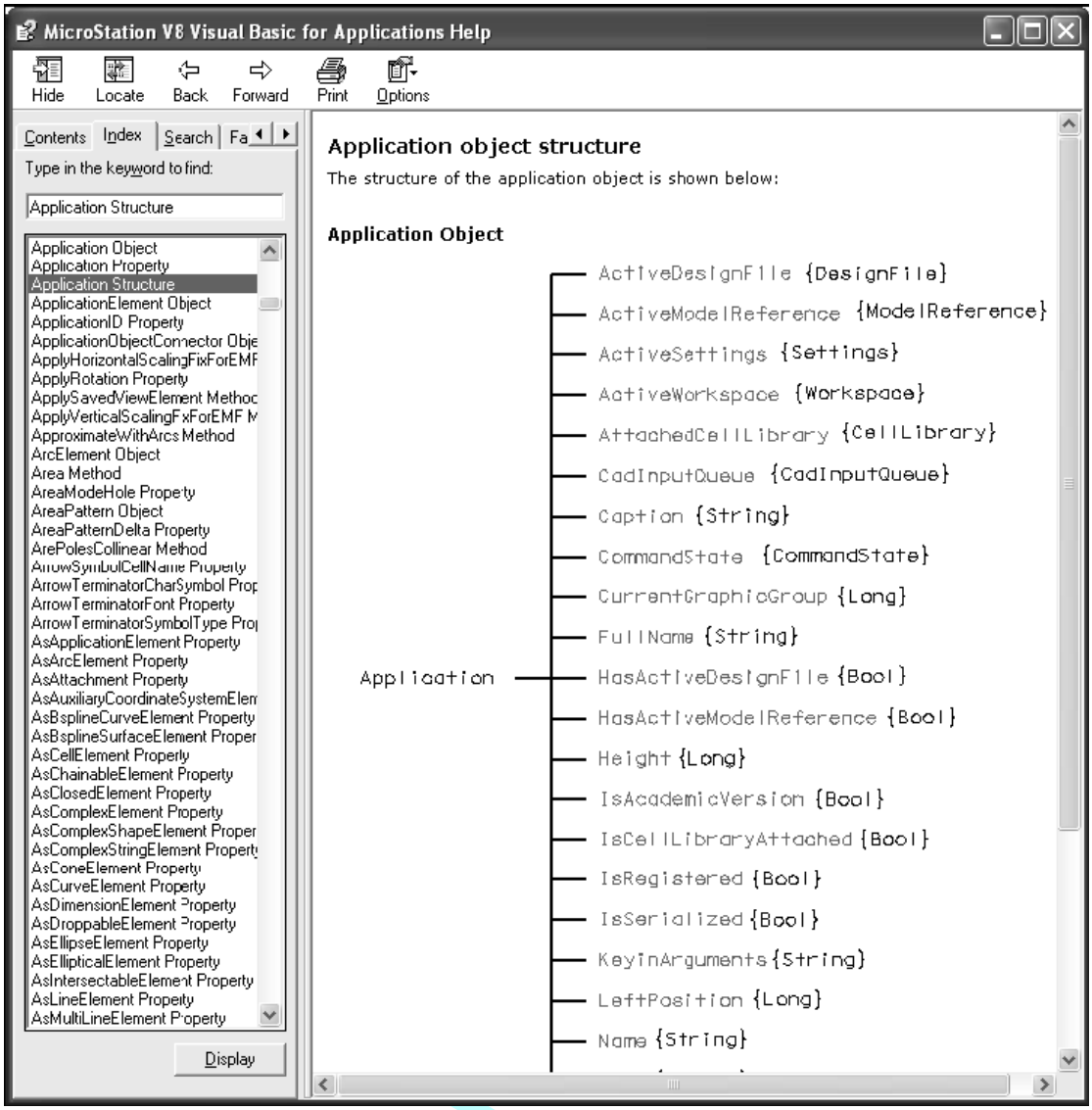
在 VBA 中工作时，能显示“列出成员”列表。当该列表显示时，我们能用箭头键和翻页键在列表中滚动。假如我们此时选择“ActiveDesignFile”并按点号键时能看到下面的样子：



“自动列出成员”列表能让我们“钻探”对象模型。

MicroStation VBA 帮助文件

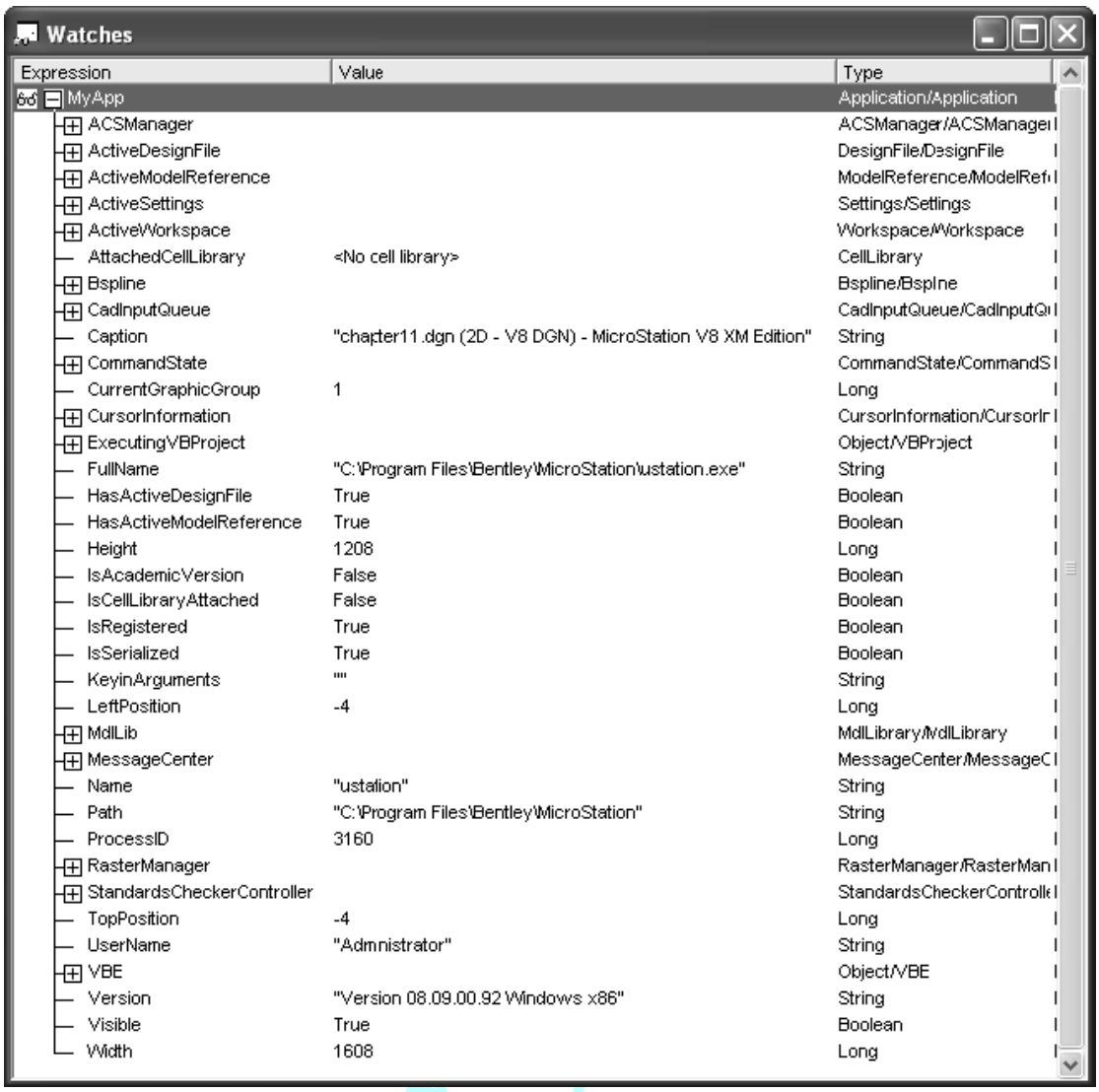
如果想进一步了解在对象浏览器中看到的某些东西，可以在对象浏览器中选中它后按键盘上的**<F1>**键，此时所选择的对象、属性、方法或事件的详细信息就会在 VBA 帮助窗口中显示出来。



在 MicroStation VBA 帮助文件中点击 Index(索引)选项卡，在“Search(搜索)”框中键入“Application Structure”。从索引列表中选择“Application Structure”显示出 MicroStation 应用对象(Application Object)结构。进一步从列表中选择“Application Object”显示出该对象的描述，还带有属性(Properties)、方法(Method)、事件(Event)、样例代码 Example Code)和参见(See Also)的超链接，点击“参见”又能显示出一个与当前主题相关联对象的列表。

添加监控

我们以前已经介绍过添加监控。对变量添加一个监控是查看其属性的最佳途径。有些属性实际上是可以继续展开的其它对象，列表中的另一些是对象的集合，在监控窗口中能查看它们。



MicroStation 对象模型

通过研究 Application(应用)对象，让我们开始查看 MicroStation 对象模型吧。

Application 对象

Application 对象指向 MicroStation 应用。

补充内容

```
Sub TestApplicationA()
    Dim MyApp As New Application
    MsgBox MyApp.Path
End Sub
Sub TestApplicationB()
```

```
    Dim MyApp As Application
    Set MyApp = Application
    MsgBox MyApp.Path
End Sub
```

这里显示的两个例子都能使 **MyApp** 指向 MicroStation 的 Application 对象。一旦一个变量指向了 Application，我们就能用该变量来操作 Application 对象，当然是通过 Application 的属性和方法来完成的。

Application 对象总是能通过公开的名字为“Application”的对象获得。这就意味着，只要我们在 VBA 中，随时都能使用名字为 **Application** 的对象。

除了能访问 Application 的属性和方法外，还能通过遍历对象模型来访问 Application 对象下的其它对象和集合。键入“Application”、点号，然后是对象模型的下一层来实现这一点。

```
Sub TestApplicationC()
    MsgBox Application.Path
End Sub
```

在这个例子中，我们没有声明任何变量或设置任何变量，我们就直接使用名字为“Application”的对象，因为它已经向我们公开。

MicroStation 对象模型的完整对象列表在本书配套的 CD 上。以打印形式在本书中给出整个对象模型有些不切实际，但你将明白对象模型有多么庞大。让我们看一眼在 MicroStation 中经常使用的几个对象的属性和方法。一些只读属性被标以“{read-only}”。

Application

- Property ACSManager As ACSManager {read-only}
- Property ActiveDesignFile As DesignFile {read-only}
- Property ActiveModelReference As ModelReference {read-only}
- Property ActiveSettings As Settings {readonly}
- Property ActiveWorkspace As Workspace {readonly}
- Sub AddAttachmentEventsHandler(EventHandler As IAttachmentEvents)
- Sub AddChangeTrackEventsHandler(EventHandler

As IChangeTrackEvents)

- Sub AddLevelChangeEventsHandler(EventHandler As ILevelChangeEvents)
- Sub AddModalDialogEventsHandler(EventHandler As IModalDialogEvents)
- Sub AddModelActivateEventsHandler(EventHandler As IModelActivateEvents)
- Sub AddModelChangeEventsHandler(EventHandler As IModelChangeEvents)
- Sub AddSaveAsEventsHandler(EventsHandler As ISaveAsEvents)
- Sub AddViewUpdateEventsHandler(EventHandler As IViewUpdateEvents)
- Sub AppendXDatum(XData() As XDatum, Type As MsdXDatumType, Value As Variant)
- Function ApplyHorizontalScalingFixForEMF(PixelCoordinate As Double) As Long
- Function ApplyVerticalScalingFixForEMF(PixelCoordinate As Double) As Long
- Function AssembleComplexStringsAndShapes(ChainableElements() As ChainableElement, [GapTolerance As Double = -1]) As ElementEnumerator
- Function Atn2(Y As Double, X As Double) As Double
- Sub AttachCellLibrary(CellLibraryName As String, [ConvertFromV7 As MsdConversionMode = msdConversionModeAlways])
- Property AttachedCellLibrary As CellLibrary {read-only}
- Property Bspline As Bspline {read-only}
- Function ByCellColor() As Long

- Function ByCellLineStyle() As LineStyle
- Function ByCellLineWeight() As Long
- Function ByLevelColor() As Long
- Function ByLevelLineStyle() As LineStyle
- Function ByLevelLineWeight() As Long
- Property CadInputQueue As CadInputQueue {read-only}
- Property Caption As String
- Property CommandState As CommandState {readonly}
- Function ConstructCirclesTangentToThreeElements(Element1 As Element, Element2 As Element, Element3 As Element, Template As Element, [OutputType As MsdTangentElementOutputType = msdTangentCircles], [SamplesCount As Long = 10]) As ElementEnumerator
- Sub CopyDesignFile(ExistingDesignFileName As String, NewDesignFileName As String, [Overwrite As Boolean])
- Function CreateApplicationElement(ApplicationID As Long, ApplicationData As DataBlock) As ApplicationElement
- Function CreateArcElement1(Template As Element, StartPoint As Point3d, CenterPoint As Point3d, EndPoint As Point3d) As ArcElement
- Function CreateArcElement2(Template As Element, CenterPoint As Point3d, PrimaryRadius As Double, SecondaryRadius As Double, Rotation As Matrix3d, StartAngle As Double, SweepAngle As Double) As ArcElement
- Function CreateArcElement3(Template As Element, StartPoint As Point3d, PointOnCurve As Point3d, EndPoint As Point3d) As ArcElement

- Function CreateArcElement4(Template As Element, StartTangent As Ray3d, EndPoint As Point3d) As ArcElement
- Function CreateArcElement5(Template As Element, Chord As Segment3d, ArcLength As Double, PlanePoint As Point3d) As ArcElement
- Function CreateAreaPattern(RowSpacing As Double, ColSpacing As Double, Angle As Double, CellName As String, Scale As Double) As AreaPattern
- Function CreateBsplineCurveElement1(Template As Element, Curve As BsplineCurve) As BsplineCurveElement
- Function CreateBsplineCurveElement2(Template As Element, Curve As InterpolationCurve) As BsplineCurveElement
- Function CreateBsplineSurfaceElement1(Template As Element, Surface As BsplineSurface) As BsplineSurfaceElement
- Function CreateCellElement1(Name As String, Elements() As _Element, Origin As Point3d, [IsPointCell As Boolean]) As CellElement
- Function CreateCellElement2(CellName As String, Origin As Point3d, Scale As Point3d, TrueScale As Boolean, Rotation As Matrix3d) As CellElement
- Function CreateCellElement3(CellName As String, Origin As Point3d, TrueScale As Boolean) As CellElement
- Function CreateComplexShapeElement1(ChainableElements() As ChainableElement, [FillMode As MsdFillMode = msdFillModeUseActive]) As ComplexShapeElement
- Function CreateComplexShapeElement2(ChainableElements()

As ChainableElement, [FillMode As MsdFillMode = msdFillModeUseActive], [GapTolerance As Double = -1]) As ComplexShapeElement

- Function CreateComplexStringElement1(ChainableElements() As ChainableElement) As ComplexStringElement
- Function CreateComplexStringElement2(ChainableElements() As ChainableElement, [GapTolerance As Double = -1]) As ComplexStringElement
- Function CreateConeElement1(Template As Element, BaseRadius As Double, BaseCenterPoint As Point3d, TopRadius As Double, TopCenterPoint As Point3d, Rotation As Matrix3d) As ConeElement
- Function CreateConeElement2(Template As Element, Radius As Double, BaseCenterPoint As Point3d, TopCenterPoint As Point3d) As ConeElement
- Function CreateCrossHatchPattern(Space1 As Double, Space2 As Double, Angle1 As Double, Angle2 As Double) As CrossHatchPattern
- Function CreateCurveElement1(Template As Element, Points() As Point3d) As CurveElement
- Function CreateDatabaseLink(Mslink As Long, Entity As Long, LinkType As MsdDatabaseLinkage, IsInformation As Boolean, DisplayableAttributeType As Long) As DatabaseLink
- Function CreateDesignFile(SeedFileName As String, NewDesignFileName As String, Open As Boolean) As DesignFile
- Function CreateDimensionElement1(Template As

Element, Rotation As Matrix3d, Type As MsdDimType, [TextOrientationView As View]) As DimensionElement

- Function CreateEllipseElement1(Template As Element, PerimeterPoint1 As Point3d, PerimeterPoint2 As Point3d, PerimeterPoint3 As Point3d, [FillMode As MsdFillMode = msdFillModeUseActive]) As EllipseElement
- Function CreateEllipseElement2(Template As Element, Origin As Point3d, PrimaryRadius As Double, SecondaryRadius As Double, Rotation As Matrix3d, [FillMode As MsdFillMode = msdFillModeUseActive]) As EllipseElement
- Function CreateEllipticalElement1(Template As Element, Ellipse As Ellipse3d, [FillMode As MsdFillMode = msdFillModeUseActive]) As Element
- Function CreateHatchPattern1(Space As Double, Angle As Double) As HatchPattern
- Function CreateLineElement1(Template As Element, Vertices() As Point3d) As LineElement
- Function CreateLineElement2(Template As Element, StartPoint As Point3d, EndPoint As Point3d) As LineElement
- Function CreateObjectInMicroStation(ProgID As String) As Unknown
- Function CreatePointStringElement1(Template As Element, Vertices() As Point3d, Disjoint As Boolean) As PointStringElement
- Function CreateSavedViewElement(ViewSpecifier As Variant, Name As String, [Description As

Chinaa BDN

String]) As SavedViewElement

- Function CreateShapeElement1(Template As Element, Vertices() As Point3d, [FillMode As MsdFillMode = msdFillModeUseActive]) As ShapeElement
- Function CreateSharedCellElement1(Name As String, Elements() As _Element, Origin As Point3d, [IsPointCell As Boolean]) As SharedCellElement
- Function CreateSharedCellElement2(CellName As String, Origin As Point3d, Scale As Point3d, TrueScale As Boolean, Rotation As Matrix3d) As SharedCellElement
- Function CreateSharedCellElement3(CellName As String, Origin As Point3d, TrueScale As Boolean) As SharedCellElement
- Function CreateTextElement1(Template As Element, Text As String, Origin As Point3d, Rotation As Matrix3d) As TextElement
- Function CreateTextNodeElement1(Template As Element, Origin As Point3d, Rotation As Matrix3d) As TextNodeElement
- Function CreateTextNodeElement2(Template As Element, Origin As Point3d, Rotation As Matrix3d, [IncrementNodeNumber As Boolean = True], [Reserved As Unknown]) As TextNodeElement
- Property CurrentGraphicGroup As Long {read-only}
- Property CursorInformation As CursorInformation {read-only}
- Function DataEntryRegionFromCriteria(StartPosition As Long, Length As Long, Justification As MsdDataEntryRegionJustification) As DataEntryRegion
- Function Degrees(Radians As Double) As Double

- Sub DeleteXDatum(XData() As XDatum, Index As Long)
- Sub DetachCellLibrary()
- Function DLongAbs(Value As DLong) As DLong
- Function DLongAdd(Term1 As DLong, Term2 As DLong) As DLong
- Function DLongComp(Value1 As DLong, Value2 As DLong) As Long
- Function DLongDivide(Numerator As DLong, Denominator As DLong) As DLong
- Function DLongFromDouble(Value As Double) As DLong
- Function DLongFromHexString(Value As String) As DLong
- Function DLongFromInt64(Value As Empty) As DLong
- Function DLongFromLong(Value As Long) As DLong
- Function DLongFromString(Value As String) As DLong
- Function DLongMod(Numerator As DLong, Denominator As DLong) As DLong
- Function DLongMultiply(Factor1 As DLong, Factor2 As DLong) As DLong
- Function DLongNegate(Value As DLong) As DLong
- Function DLongSubtract(Minuend As DLong, Subtrahend As DLong) As DLong
- Function DLongToHexString(Value As DLong) As String
- Sub DLongToInt64(Value As DLong)
- Function DLongToLong(Value As DLong) As Long
- Function DLongToString(Value As DLong) As String
- Function Ellipse3dFromEllipticalElement(Element As EllipticalElement) As Ellipse3d
- Property ExecutingVBProject As Object {readonly}
- Property FullName As String {read-only}

- Function GetCellInformationEnumerator(IncludeSharedCells As Boolean, IncludeFullPath As Boolean) As CellInformationEnumerator
- Function GetCExpressionValue(CExpression As String, [MdlApplicationName As String]) As Variant
- Function GetCExpressionValueAsDLong(CExpression As String, [MdlApplicationName As String]) As DLong
- Function GetFloodBoundary(CandidateElements() As _Element, Template As Element, SeedPoint As Point3d, [ViewSpecifier As Variant], [FindHoles As Boolean = True], [Tolerance As Double = -1], [FillMode As MsdFillMode = msdFillModeUseActive]) As Element
- Function GetRegionDifference(RegionSolid() As _Element, RegionHoles() As _Element, Template As Element, [FillMode As MsdFillMode = msdFillModeUseActive]) As ElementEnumerator
- Function GetRegionIntersection(Region1() As _Element, Region2() As _Element, Template As Element, [FillMode As MsdFillMode = msdFillModeUseActive]) As ElementEnumerator
- Function GetRegionUnion(Region1() As _Element, Region2() As _Element, Template As Element, [FillMode As MsdFillMode = msdFillModeUseActive]) As ElementEnumerator
- Property HasActiveDesignFile As Boolean {read-only}
- Property HasActiveModelReference As Boolean {read-only}
- Property Height As Long

- Sub InsertXDatum(XData() As XDatum, Index As Long, Type As MsdXDatumType, Value As Variant)
- Property IsAcademicVersion As Boolean {readonly}
- Property IsCellLibraryAttached As Boolean {read-only}
- Property IsRegistered As Boolean {read-only}
- Property IsSerialized As Boolean {read-only}
- Property KeyinArguments As String
- Property LeftPosition As Long
- Function Matrix3dAdd2Scaled(Matrix0 As Matrix3d, Matrix1 As Matrix3d, Scale1 As Double, Matrix2 As Matrix3d, Scale2 As Double) As Matrix3d
- Function Matrix3dDeterminant(Matrix As Matrix3d) As Double
- Function Matrix3dEqual(Matrix1 As Matrix3d, Matrix2 As Matrix3d) As Boolean
- Function Matrix3dEqualTolerance(Matrix1 As Matrix3d, Matrix2 As Matrix3d, Tolerance As Double) As Boolean
- Function Matrix3dFromAxisAndRotationAngle(Axis As Long, Radians As Double) As Matrix3d
- Function Matrix3dFromDirectionAndScale(Vector As Point3d, Scale As Double) As Matrix3d
- Function Matrix3dFromMatrix3dTimesMatrix3d(A As Matrix3d, B As Matrix3d) As Matrix3d
- Function Matrix3dFromMatrix3dTimesMatrix3dTimesMatrix3d (A As Matrix3d, B As Matrix3d, C As Matrix3d) As Matrix3d
- Function Matrix3dFromPoint3dColumns(XVector As Point3d, YVector As Point3d, ZVector As

Scuta BDN

Point3d) As Matrix3d

- Function Matrix3dFromPoint3dRows(XVector As Point3d, YVector As Point3d, ZVector As Point3d) As Matrix3d
- Function Matrix3dFromRotationBetweenVectors(Vector0 As Point3d, Vector1 As Point3d) As Matrix3d
- Function Matrix3dFromRowValues(X00 As Double, X01 As Double, X02 As Double, X10 As Double, X11 As Double, X12 As Double, X20 As Double, X21 As Double, X22 As Double) As Matrix3d
- Function Matrix3dFromScale(Scale As Double) As Matrix3d
- Function Matrix3dFromScaleFactors(Xscale As Double, Yscale As Double, Zscale As Double) As Matrix3d
- Function Matrix3dFromTransform3d(Transform As Transform3d) As Matrix3d
- Function Matrix3dFromVectorAndRotationAngle(Axis As Point3d, Radians As Double) As Matrix3d
- Function Matrix3dFromXYRotationSkewAndScale(XAxisAngle As Double, YAxisSkewAngle As Double, Xscale As Double, Yscale As Double, Zscale As Double) As Matrix3d
- Function Matrix3dGetComponentByRowAndColumn(Matrix As Matrix3d, Row As Long, Col As Long) As Double
- Function Matrix3dHasInverse(Matrix As Matrix3d) As Boolean
- Function Matrix3dIdentity() As Matrix3d
- Function Matrix3dInverse(Forward As Matrix3d) As Matrix3d

- Function Matrix3dIsIdentity(Matrix As Matrix3d) As Boolean
- Function Matrix3dIsOrthogonal(Matrix As Matrix3d) As Boolean
- Function Matrix3dIsRigid(Matrix As Matrix3d) As Boolean
- Function Matrix3dIsRotateScaleRotate(Matrix As Matrix3d, Rotation1 As Matrix3d, ScaleFactors As Point3d, Rotation2 As Matrix3d) As Boolean
- Function Matrix3dIsSignedPermutation(Matrix As Matrix3d) As Boolean
- Function Matrix3dIsXRotationYRotationZRotationScale(Matrix As Matrix3d, RadiansX As Double, RadiansY As Double, RadiansZ As Double, Scale As Double) As Boolean
- Function Matrix3dIsXYRotation(Matrix As Matrix3d, XYRotationRadians As Double) As Boolean
- Function Matrix3dIsXYRotationSkewAndScale(Matrix As Matrix3d, XAxisAngle As Double, YAxisSkewAngle As Double, Xscale As Double, Yscale As Double, Zscale As Double) As Boolean
- Function Matrix3dMaxAbs(Matrix As Matrix3d) As Double
- Function Matrix3dMaxDiff(Matrix1 As Matrix3d, Matrix2 As Matrix3d) As Double
- Function Matrix3dRotationFromColumnZ(Normal As Point3d) As Matrix3d
- Function Matrix3dRotationFromPoint3dOriginXY(Origin As Point3d, XPoint As Point3d, YPoint As Point3d) As Matrix3d
- Function Matrix3dRotationFromRowZ(Normal As

Point3d) As Matrix3d

- Sub Matrix3dSetComponentByRowAndColumn(Matrix As Matrix3d, RowIndex As Long, ColumnIndex As Long, Value As Double)
- Function Matrix3dSumSquares(Matrix As Matrix3d) As Double
- Function Matrix3dTranspose(Matrix As Matrix3d) As Matrix3d
- Function Matrix3dZero() As Matrix3d
- Function MdlCreateElementFromElementDescrP(ElementDescrP As Long) As Element
- Function MdlGetDesignFileFromModelRefP(ModelRefP As Long) As DesignFile
- Function MdlGetModelReferenceFromModelRefP(ModelRefP As Long) As ModelReference
- Property MessageCenter As MessageCenter {read-only}
- Property Name As String {read-only}
- Sub OnDesignFileClosed(DesignFileName As String)
- Sub OnDesignFileOpened(DesignFileName As String)
- Function OpenDesignFile(DesignFileName As String, [ReadOnly As Boolean], [V7Action As MsdV7Action = msdV7ActionAskUser]) As DesignFile
- Function OpenDesignFileForProgram(DesignFileName As String, [ReadOnly As Boolean]) As DesignFile
- Property Path As String {read-only}
- Function Pi() As Double
- Function

China BDN

Plane3dIntersectsPlane3d(IntersectionRay As Ray3d, Plane0 As Plane3d, Plane1 As Plane3d) As Boolean

- Function Plane3dIntersectsRay3d(IntersectionPoint As Point3d, Parameter As Double, Plane As Plane3d, Ray As Ray3d) As Boolean
- Function Point2dAdd(Point1 As Point2d, Point2 As Point2d) As Point2d
- Function Point2dAdd2Scaled(Origin As Point2d, Vector1 As Point2d, Scale1 As Double, Vector2 As Point2d, Scale2 As Double) As Point2d
- Function Point2dAdd3Scaled(Origin As Point2d, Vector1 As Point2d, Scale1 As Double, Vector2 As Point2d, Scale2 As Double, Vector3 As Point2d, Scale3 As Double) As Point2d
- Function Point2dAddScaled(Origin As Point2d, Vector As Point2d, Scale As Double) As Point2d
- Function Point2dAreVectorsParallel(Vector1 As Point2d, Vector2 As Point2d) As Boolean
- Function Point2dAreVectorsPerpendicular(Vector1 As Point2d, Vector2 As Point2d) As Boolean
- Function Point2dCrossProduct(Vector1 As Point2d, Vector2 As Point2d) As Double
- Function Point2dCrossProduct3Points(Origin As Point2d, Target1 As Point2d, Target2 As Point2d) As Double
- Function Point2dDistance(Point0 As Point2d, Point1 As Point2d) As Double
- Function Point2dDistanceSquared(Point1 As

Point2d, Point2 As Point2d) As Double

- Function Point2dDotDifference(TargetPoint As Point2d, Origin As Point2d, Vector As Point2d) As Double
- Function Point2dDotProduct(Vector1 As Point2d, Vector2 As Point2d) As Double
- Function Point2dDotProduct3Points(Origin As Point2d, Target1 As Point2d, Target2 As Point2d) As Double
- Function Point2dEqual(Vector1 As Point2d, Vector2 As Point2d) As Boolean
- Function Point2dEqualTolerance(Vector1 As Point2d, Vector2 As Point2d, Tolerance As Double) As Boolean
- Function Point2dFromXY(X As Double, Y As Double) As Point2d
- Function Point2dGetComponent(Point As Point2d, Index As Long) As Double
- Function Point2dInterpolate(Point0 As Point2d, S As Double, Point1 As Point2d) As Point2d
- Function Point2dMagnitude(Vector As Point2d) As Double
- Function Point2dMagnitudeSquared(Vector As Point2d) As Double
- Function Point2dMaxAbs(Vector As Point2d) As Double
- Function Point2dNegate(Vector As Point2d) As Point2d
- Function Point2dNormalize(Vector As Point2d) As Point2d
- Function Point2dOne() As Point2d
- Function Point2dSignedAngleBetweenVectors(Vector1 As Point2d, Vector2 As Point2d) As Double
- Function Point2dSubtract(Point1 As Point2d,

China BDN

Point2 As Point2d) As Point2d

- Function Point2dZero() As Point2d
- Function Point3dAdd(Point1 As Point3d, Point2 As Point3d) As Point3d
- Function Point3dAdd2Scaled(Origin As Point3d, Vector1 As Point3d, Scale1 As Double, Vector2 As Point3d, Scale2 As Double) As Point3d
- Function Point3dAdd2ScaledVector3d(Origin As Point3d, Vector1 As Vector3d, Scale1 As Double, Vector2 As Vector3d, Scale2 As Double) As Point3d
- Function Point3dAdd3Scaled(Origin As Point3d, Vector1 As Point3d, Scale1 As Double, Vector2 As Point3d, Scale2 As Double, Vector3 As Point3d, Scale3 As Double) As Point3d
- Function Point3dAdd3ScaledVector3d(Origin As Point3d, Vector1 As Vector3d, Scale1 As Double, Vector2 As Vector3d, Scale2 As Double, Vector3 As Vector3d, Scale3 As Double) As Point3d
- Function Point3dAddAngleDistance(Point1 As Point3d, AngleRadians As Double, DistanceXY As Double, Dz As Double) As Point3d
- Function Point3dAddPoint3dVector3d(Base As Point3d, Vector As Vector3d) As Point3d
- Function Point3dAddScaled(Origin As Point3d, Vector As Point3d, Scale As Double) As Point3d
- Function Point3dAddScaledVector3d(Origin As Point3d, Vector As Vector3d, Scale As Double) As Point3d
- Function Point3dAngleBetweenVectors(Vector1 As Point3d, Vector2 As Point3d) As Double
- Function Point3dAngleBetweenVectorsXY(Vector1

China BDN

As Point3d, Vector2 As Point3d) As Double

- Function Point3dAreVectorsParallel(Vector1 As Point3d, Vector2 As Point3d) As Boolean
- Function Point3dAreVectorsPerpendicular(Vector1 As Point3d, Vector2 As Point3d) As Boolean
- Function Point3dCrossProduct(Vector1 As Point3d, Vector2 As Point3d) As Point3d
- Function Point3dCrossProduct3Points(Origin As Point3d, Target1 As Point3d, Target2 As Point3d) As Point3d
- Function Point3dCrossProduct3PointsXY(Origin As Point3d, Target1 As Point3d, Target2 As Point3d) As Double
- Function Point3dCrossProductXY(Vector1 As Point3d, Vector2 As Point3d) As Double
- Function Point3dDistance(Point1 As Point3d, Point2 As Point3d) As Double
- Function Point3dDistanceSquared(Point1 As Point3d, Point2 As Point3d) As Double
- Function Point3dDistanceSquaredXY(Point1 As Point3d, Point2 As Point3d) As Double
- Function Point3dDistanceXY(Point1 As Point3d, Point2 As Point3d) As Double
- Function Point3dDotDifference(Target As Point3d, Origin As Point3d, Vector As Point3d) As Double
- Function Point3dDotDifferenceVector3d(Target As Point3d, Origin As Point3d, Vector As Vector3d) As Double
- Function Point3dDotProduct(Vector1 As Point3d,

China BDN

Vector2 As Point3d) As Double

- Function Point3dDotProduct3Points(Origin As Point3d, Target1 As Point3d, Target2 As Point3d) As Double
- Function Point3dDotProduct3PointsXY(Origin As Point3d, Target1 As Point3d, Target2 As Point3d) As Double
- Function Point3dDotProductXY(Vector1 As Point3d, Vector2 As Point3d) As Double
- Function Point3dEqual(Vector1 As Point3d, Vector2 As Point3d) As Boolean
- Function Point3dEqualTolerance(Vector1 As Point3d, Vector2 As Point3d, Tolerance As Double) As Boolean
- Function Point3dFromAngleDistance(AngleRadians As Double, DistanceXY As Double, Z As Double) As Point3d
- Function Point3dFromMatrix3dColumn(Matrix As Matrix3d, Col As Long) As Point3d
- Function Point3dFromMatrix3dInverseTimesPoint3d(Matrix As Matrix3d, Point As Point3d) As Point3d
- Function Point3dFromMatrix3dInverseTransposeTimesPoint3d(Matrix As Matrix3d, Point As Point3d) As Point3d
- Function Point3dFromMatrix3dRow(Matrix As Matrix3d, Row As Long) As Point3d
- Function Point3dFromMatrix3dTimesPoint3d(Matrix As Matrix3d, Point As Point3d) As Point3d

- Function Point3dFromMatrix3dTimesXYZ(Matrix As Matrix3d, X As Double, Y As Double, Z As Double) As Point3d
- Function Point3dFromMatrix3dTransposeTimesPoint3d(Matrix As Matrix3d, Point As Point3d) As Point3d
- Function Point3dFromMatrix3dTransposeTimesXYZ(Matrix As Matrix3d, X As Double, Y As Double, Z As Double) As Point3d
- Function Point3dFromRay3dFractionParameter(Ray As Ray3d, Fraction As Double) As Point3d
- Function Point3dFromRay3dTangent(Ray As Ray3d) As Point3d
- Function Point3dFromSegment3dFractionParameter(Segment As Segment3d, Fraction As Double) As Point3d
- Function Point3dFromSegment3dTangent(Segment As Segment3d) As Point3d
- Function Point3dFromTransform3d(Transform As Transform3d) As Point3d
- Function Point3dFromTransform3dTimesPoint3d(Transform As Transform3d, Point As Point3d) As Point3d
- Function Point3dFromTransform3dTimesXYZ(Transform As Transform3d, X As Double, Y As Double, Z As Double) As Point3d
- Function Point3dFromVector3d(Vector As Vector3d) As Point3d
- Function Point3dFromXY(Ax As Double, Ay As

China BDN

Double) As Point3d

- Function Point3dFromXYZ(Ax As Double, Ay As Double, Az As Double) As Point3d
- Function Point3dGetComponent(Point As Point3d, Index As Long) As Double
- Function Point3dInPolygonXY(Point As Point3d, PolygonVertices() As Point3d, [Tolerance As Double = -1]) As Long
- Function Point3dInterpolate(Point0 As Point3d, FractionParameter As Double, Point1 As Point3d) As Point3d
- Function Point3dIsPointInCCWSector(TestPoint As Point3d, Origin As Point3d, Target0 As Point3d, Target1 As Point3d, UpVector As Point3d) As Boolean
- Function Point3dIsPointInSmallerSector(TestPoint As Point3d, Origin As Point3d, Target1 As Point3d, Target2 As Point3d) As Boolean
- Function Point3dIsVectorInCCWSector(TestVector As Point3d, Vector0 As Point3d, Vector1 As Point3d, UpVector As Point3d) As Boolean
- Function Point3dIsVectorInSmallerSector(TestVector As Point3d, Vector0 As Point3d, Vector1 As Point3d) As Boolean
- Function Point3dMagnitude(Vector As Point3d) As Double
- Function Point3dMagnitudeSquared(Vector As Point3d) As Double
- Function Point3dMaxAbs(Vector As Point3d) As Double

- Function Point3dNegate(Vector As Point3d) As Point3d
- unction Point3dNormalize(Vector As Point3d) As Point3d
- Function Point3dOne() As Point3d
- Function Point3dPlanarAngleBetweenVectors(Vector1 As Point3d, Vector2 As Point3d, PlaneNormal As Point3d) As Double
- Function Point3dPolarAngle(Vector As Point3d) As Double
- Function Point3dProjectToPlane3d(Point As Point3d, Plane As Plane3d, [ViewSpecifier As Variant], [UseAuxiliaryCoordinateSystem As Boolean = False]) As Point3d
- Function Point3dProjectToRay3d(Parameter As Double, Point As Point3d, Ray As Ray3d, [ViewSpecifier As Variant], [UseAuxiliaryCoordinateSystem As Boolean = False]) As Point3d
- Function Point3dRotateXY(Vector As Point3d, Theta As Double) As Point3d
- Function Point3dScale(Vector As Point3d, Scale As Double) As Point3d
- Sub Point3dSetComponent(Point As Point3d, Index As Long, Value As Double)
- Function Point3dSignedAngleBetweenVectors(Vector1 As Point3d, Vector2 As Point3d, OrientationVector As Point3d) As Double
- Function Point3dSmallerAngleBetweenUnorientedVectors(Vector1 As Point3d, Vector2 As Point3d) As Double

China BDN

- Function Point3dSmallerAngleBetweenUnorientedVectorsXY(Vector1 As Point3d, Vector2 As Point3d) As Double
- Function Point3dSubtract(Point1 As Point3d, Point2 As Point3d) As Point3d
- Function Point3dSubtractPoint3dVector3d(Base As Point3d, Vector As Vector3d) As Point3d
- Function Point3dTripleProduct(Vector1 As Point3d, Vector2 As Point3d, Vector3 As Point3d) As Double
- Function Point3dTripleProduct4Points(Origin As Point3d, Target1 As Point3d, Target2 As Point3d, Target3 As Point3d) As Double
- Function Point3dZero() As Point3d
- Function PointsToPixelsX(PointCoordinate As Double) As Long
- Function PointsToPixelsY(PointCoordinate As Double) As Long
- Property ProcessID As Long {read-only}
- Sub Quit()
- Function Radians(Degrees As Double) As Double
- Function Range3dContainsPoint3d(Range As Range3d, Point As Point3d) As Boolean
- Function Range3dContainsXYZ(Range As Range3d, X As Double, Y As Double, Z As Double) As Boolean
- Function Range3dEqual(Range1 As Range3d, Range2 As Range3d) As Boolean
- Function Range3dEqualTolerance(Range0 As Range3d, Range1 As Range3d, Tolerance As Double) As Boolean

- Function Range3dExtentSquared(Range As Range3d) As Double
- Function Range3dFromPoint3d(Point As Point3d) As Range3d
- Function Range3dFromPoint3dPoint3d(Point0 As Point3d, Point1 As Point3d) As Range3d
- Function Range3dFromPoint3dPoint3dPoint3d(Point0 As Point3d, Point1 As Point3d, Point2 As Point3d) As Range3d
- Function Range3dFromRange3dMargin(Range As Range3d, Margin As Double) As Range3d
- Function Range3dFromXYZ(X As Double, Y As Double, Z As Double) As Range3d
- Function Range3dFromXYZXYZ(X1 As Double, Y1 As Double, Z1 As Double, X2 As Double, Y2 As Double, Z2 As Double) As Range3d
- Function Range3dInit() As Range3d
- Function Range3dIntersect(Range1 As Range3d, Range2 As Range3d) As Range3d
- Function Range3dIntersect2(ResultRange As Range3d, Range1 As Range3d, Range2 As Range3d) As Boolean
- Function Range3dIsContainedInRange3d(InnerRange As Range3d, OuterRange As Range3d) As Boolean
- Function Range3dIsNull(Range As Range3d) As Boolean
- Function Range3dScaleAboutCenter(RangeIn As Range3d, Scale As Double) As Range3d
- Function Range3dUnion(Range0 As Range3d, Range1 As Range3d) As Range3d

China BDN

- Function Range3dUnionPoint3d(Range As Range3d, Point As Point3d) As Range3d
- Function Range3dUnionXYZ(Range As Range3d, X As Double, Y As Double, Z As Double) As Range3d
- Property RasterManager As RasterManager {read-only}
- Sub Ray3dClosestPoint(Ray As Ray3d, SpacePoint As Point3d, ClosePoint As Point3d, CloseFraction As Double)
- Sub Ray3dClosestPointBounded(Ray As Ray3d, SpacePoint As Point3d, ClosePoint As Point3d, CloseFraction As Double)
- Sub Ray3dClosestPointBoundedXY(Ray As Ray3d, SpacePoint As Point3d, ClosePoint As Point3d, CloseFraction As Double)
- Sub Ray3dClosestPointXY(Ray As Ray3d, SpacePoint As Point3d, ClosePoint As Point3d, CloseFraction As Double)
- Function Ray3dFromPoint3dStartEnd(Point0 As Point3d, Point1 As Point3d) As Ray3d
- Function Ray3dFromPoint3dStartTangent(Point0 As Point3d, Tangent As Point3d) As Ray3d
- Function Ray3dFromSegment3d(Segment As Segment3d) As Ray3d
- Function Ray3dFromTransform3dTimesRay3d(Transform As Transform3d, Ray As Ray3d) As Ray3d
- Function Ray3dFromXYZXYZStartEnd(X0 As Double, Y0 As Double, Z0 As Double, X1 As Double, Y1 As Double, Z1 As Double) As Ray3d
- Function Ray3dLength(Ray As Ray3d) As Double

China BDN

- Function Ray3dLengthSquared(Ray As Ray3d) As Double
- Function Ray3dPlane3dIntersect(Ray As Ray3d, Plane As Plane3d, Point As Point3d, Fraction As Double) As Boolean
- Function Ray3dRay3dClosestApproach(Ray0 As Ray3d, Ray1 As Ray3d, Point0 As Point3d, Fraction0 As Double, Point1 As Point3d, Fraction1 As Double) As Boolean
- Function Ray3dRay3dIntersectXY(Ray0 As Ray3d, Ray1 As Ray3d, Point0 As Point3d, Fraction0 As Double, Point1 As Point3d, Fraction1 As Double) As Boolean
- Sub RedrawAllViews([DrawMode As MsdDrawingMode = msdDrawingModeNormal])
- Sub RegisterV8ToV7Filter(Handler As IConvertV8ToV7)
- Sub RemoveAttachmentEventsHandler(EventHandler As IAttachmentEvents)
- Sub RemoveChangeTrackEventsHandler(EventHandler As IChangeTrackEvents)
- Sub RemoveLevelChangeEventsHandler(EventHandler As ILevelChangeEvents)
- Sub RemoveModalDialogEventsHandler(EventHandler As IModalDialogEvents)
- Sub RemoveModelActivateEventsHandler(EventHandler As IModelActivateEvents)
- Sub

China BDN

RemoveModelChangeEventsHandler(EventHandler As IModelChangeEvents)

- Sub RemoveSaveAsEventsHandler(EventsHandler As ISaveAsEvents)
- Sub RemoveViewUpdateEventsHandler(EventHandler As IViewUpdateEvents)
- Sub ResetDisplaySet(ShowEverything As Boolean)
- Sub SaveSettings()
- Sub Segment3dClosestPoint(Segment As Segment3d, SpacePoint As Point3d, ClosePoint As Point3d, CloseFraction As Double)
- Sub Segment3dClosestPointBounded(Segment As Segment3d, SpacePoint As Point3d, ClosePoint As Point3d, CloseFraction As Double)
- Sub Segment3dClosestPointBoundedXY(Segment As Segment3d, SpacePoint As Point3d, ClosePoint As Point3d, CloseFraction As Double)
- Sub Segment3dClosestPointXY(Segment As Segment3d, SpacePoint As Point3d, ClosePoint As Point3d, CloseFraction As Double)
- Function Segment3dFromPoint3dStartEnd(Point0 As Point3d, Point1 As Point3d) As Segment3d
- Function Segment3dFromPoint3dStartTangent(Point0 As Point3d, Tangent As Point3d) As Segment3d
- Function Segment3dFromRay3d(Ray As Ray3d) As Segment3d
- Function Segment3dFromTransform3dTimesSegment3d(Transform As Transform3d, Segment As Segment3d) As Segment3d
- Function Segment3dFromXYZXYZStartEnd(X0 As

China BDN

Double, Y0 As Double, Z0 As Double, X1 As Double, Y1 As Double, Z1 As Double) As Segment3d

- Function Segment3dLength(Segment As Segment3d) As Double
- Function Segment3dLengthSquared(Segment As Segment3d) As Double
- Function Segment3dPlane3dIntersect(Segment As Segment3d, Plane As Plane3d, Point As Point3d, Fraction As Double) As Boolean
- Function Segment3dSegment3dClosestApproach(Segment0 As Segment3d, Segment1 As Segment3d, Point0 As Point3d, Fraction0 As Double, Point1 As Point3d, Fraction1 As Double) As Boolean
- Function Segment3dSegment3dIntersectXY(Segment0 As Segment3d, Segment1 As Segment3d, Point0 As Point3d, Fraction0 As Double, Point1 As Point3d, Fraction1 As Double) As Boolean
- Sub SetCExpressionValue(CExpression As String, NewValue As Variant, [MdlApplicationName As String])
- Sub SetCExpressionValueAsDLong(CExpression As String, NewValue As DLong, [MdlApplicationName As String])
- Sub ShowCommand([Command As String])
- Sub ShowError([Error As String])
- Sub ShowPrompt([Prompt As String])
- Sub ShowStatus([Status As String])
- Sub ShowTempMessage(Area As MsdStatusBarArea, Message As String, [Details As String])
- Property StandardsCheckerController As StandardsCheckerController {read-only}

- Sub StartBusyCursor()
- Sub StopBusyCursor()
- Property TopPosition As Long
- Function Transform3dEqual(Transform1 As Transform3d, Transform2 As Transform3d) As Boolean
- Function Transform3dEqualTolerance(Transform1 As Transform3d, Transform2 As Transform3d, MatrixTolerance As Double, PointTolerance As Double) As Boolean
- Function Transform3dFactorMirror(Transform As Transform3d, ResidualTransform As Transform3d, MirrorTransform As Transform3d, FixedPoint As Point3d, PlaneNormal As Point3d) As Boolean
- Function Transform3dFromLineAndRotationAngle(Point0 As Point3d, Point1 As Point3d, Radians As Double) As Transform3d
- Function Transform3dFromMatrix3d(Matrix As Matrix3d) As Transform3d
- Function Transform3dFromMatrix3dAndFixedPoint3d(Matrix As Matrix3d, Origin As Point3d) As Transform3d
- Function Transform3dFromMatrix3dPoint3d(Matrix As Matrix3d, Translation As Point3d) As Transform3d
- Function Transform3dFromMatrix3dTimesTransform3d(Matrix As Matrix3d, Transform As Transform3d) As Transform3d
- Function Transform3dFromMirrorPlane(Origin As Point3d, Normal As Point3d) As Transform3d
- Function Transform3dFromPlane3dToWorld(Plane

China BDN

As Plane3d) As Transform3d

- Function Transform3dFromPoint3d(Translation As Point3d) As Transform3d
- Function Transform3dFromRowValues(X00 As Double, X01 As Double, X02 As Double, Tx As Double, X10 As Double, X11 As Double, X12 As Double, Ty As Double, X20 As Double, X21 As Double, X22 As Double, Tz As Double) As Transform3d
- Function Transform3dFromSquaredTransform3d(Transform As Transform3d, PrimaryAxis As Long, SecondaryAxis As Long) As Transform3d
- Function Transform3dFromTransform3dTimesMatrix3d(Transform As Transform3d, Matrix As Matrix3d) As Transform3d
- Function Transform3dFromTransform3dTimesTransform3d(Transform1 As Transform3d, Transform2 As Transform3d) As Transform3d
- Function Transform3dFromTransform3dTimesTransform3dTimesTransform3d(Transform1 As Transform3d, Transform2 As Transform3d, Transform3 As Transform3d) As Transform3d
- Function Transform3dFromWorldToPlane3d(Plane As Plane3d) As Transform3d
- Function Transform3dFromXYZ(X As Double, Y As Double, Z As Double) As Transform3d
- Function Transform3dGetMatrixComponentByRowAndColumn(Tr

ansform As Transform3d, Row As Long, Col As Long) As Double

- Function Transform3dGetPointComponent(Transform As Transform3d, Row As Long) As Double
- Function Transform3dHasInverse(Transform As Transform3d) As Boolean
- Function Transform3dIdentity() As Transform3d
- Function Transform3dInverse(In As Transform3d) As Transform3d
- Function Transform3dIsIdentity(Transform As Transform3d) As Boolean
- Function Transform3dIsMirrorAboutPlane(Transform As Transform3d, PlanePoint As Point3d, PlaneNormal As Point3d) As Boolean
- Function Transform3dIsPlanar(Transform As Transform3d, Normal As Point3d) As Boolean
- Function Transform3dIsRigid(Transform As Transform3d) As Boolean
- Function Transform3dIsRotateAroundLine(Transform As Transform3d, FixedPoint As Point3d, DirectionVector As Point3d, Radians As Double) As Boolean
- Function Transform3dIsTranslate(Transform As Transform3d, Translation As Point3d) As Boolean
- Function Transform3dIsTranslateRotateScaleRotate(Transform As Transform3d, Translation As Point3d, Rotation1 As Matrix3d, ScaleFactors As Point3d, Rotation2 As Matrix3d) As Boolean
- Function Transform3dIsUniformScale(Transform

Cnlia BDN

As Transform3d, FixedPoint As Point3d, Scale As Double) As Boolean

- Function Transform3dIsUniformScaleAndRotateAroundLine(Transform As Transform3d, FixedPoint As Point3d, DirectionVector As Point3d, Radians As Double, Scale As Double) As Boolean
- Sub Transform3dSetMatrixComponentByRowAndColumn(Transform As Transform3d, RowIndex As Long, ColumnIndex As Long, Value As Double)
- Sub Transform3dSetPointComponent(Transform As Transform3d, RowIndex As Long, Value As Double)
- Function Transform3dZero() As Transform3d
- Function UpdateGraphicGroupNumber() As Long
- Property UserName As String {read-only}
- Property VBE As Object {read-only}
- Function Vector3dAdd(Vector1 As Vector3d, Vector2 As Vector3d) As Vector3d
- Function Vector3dAdd2Scaled(Origin As Vector3d, Vector1 As Vector3d, Scale1 As Double, Vector2 As Vector3d, Scale2 As Double) As Vector3d
- Function Vector3dAdd3Scaled(Origin As Vector3d, Vector1 As Vector3d, Scale1 As Double, Vector2 As Vector3d, Scale2 As Double, Vector3 As Vector3d, Scale3 As Double) As Vector3d
- Function Vector3dAddScaled(Origin As Vector3d, Vector As Vector3d, Scale As Double) As Vector3d
- Function Vector3dAngleBetweenVectors(Vector1 As Vector3d, Vector2 As Vector3d) As Double
- Function Vector3dAngleBetweenVectorsXY(Vector1

As Vector3d, Vector2 As Vector3d) As Double

- Function Vector3dAreVectorsParallel(Vector1 As Vector3d, Vector2 As Vector3d) As Boolean
- Function Vector3dAreVectorsPerpendicular(Vector1 As Vector3d, Vector2 As Vector3d) As Boolean
- Function Vector3dCrossProduct(Vector1 As Vector3d, Vector2 As Vector3d) As Vector3d
- Function Vector3dCrossProduct3Points(Origin As Point3d, Target1 As Point3d, Target2 As Point3d) As Vector3d
- Function Vector3dCrossProductXY(Vector1 As Vector3d, Vector2 As Vector3d) As Double
- Function Vector3dDistance(Vector1 As Vector3d, Vector2 As Vector3d) As Double
- Function Vector3dDistanceSquared(Vector1 As Vector3d, Vector2 As Vector3d) As Double
- Function Vector3dDistanceSquaredXY(Vector1 As Vector3d, Vector2 As Vector3d) As Double
- Function Vector3dDistanceXY(Vector1 As Vector3d, Vector2 As Vector3d) As Double
- Function Vector3dDotProduct(Vector1 As Vector3d, Vector2 As Vector3d) As Double
- b Function Vector3dDotProductXY(Vector1 As Vector3d, Vector2 As Vector3d) As Double
- Function Vector3dDotProductXYZ(Vector As Vector3d, Ax As Double, Ay As Double, Az As Double) As Double
- Function Vector3dEqual(Vector1 As Vector3d, Vector2 As Vector3d) As Boolean
- Function Vector3dEqualTolerance(Vector1 As

Vector3d, Vector2 As Vector3d, Tolerance As Double) As Boolean

- Function Vector3dFromMatrix3dColumn(Matrix As Matrix3d, Col As Long) As Vector3d
- Function Vector3dFromMatrix3dRow(Matrix As Matrix3d, Row As Long) As Vector3d
- Function Vector3dFromMatrix3dTimesVector3d(Matrix As Matrix3d, Vector As Vector3d) As Vector3d
- Function Vector3dFromMatrix3dTimesXYZ(Matrix As Matrix3d, X As Double, Y As Double, Z As Double) As Vector3d
- Function Vector3dFromMatrix3dTransposeTimesVector3d(Matrix As Matrix3d, Vector As Vector3d) As Vector3d
- Function Vector3dFromMatrix3dTransposeTimesXYZ(Matrix As Matrix3d, X As Double, Y As Double, Z As Double) As Vector3d
- Function Vector3dFromPoint3d(Point As Point3d) As Vector3d
- Function Vector3dFromTransform3dColumn(Transform As Transform3d, Col As Long) As Vector3d
- Function Vector3dFromTransform3dRow(Transform As Transform3d, Row As Long) As Vector3d
- Function Vector3dFromTransform3dTimesVector3d(Transform As Transform3d, Vector As Vector3d) As Vector3d
- Function Vector3dFromTransform3dTimesXYZ(Transform As Transform3d, X As Double, Y As Double, Z As Double) As Vector3d
- Function

Vector3dFromTransform3dTranslation(Transform As Transform3d) As Vector3d

- Function Vector3dFromTransform3dTransposeTimesVector3d(Transform As Transform3d, Vector As Vector3d) As Vector3d
- Function Vector3dFromTransform3dTransposeTimesXYZ(Trans form As Transform3d, X As Double, Y As Double, Z As Double) As Vector3d
- Function Vector3dFromXY(Ax As Double, Ay As Double) As Vector3d
- Function Vector3dFromXYAngleAndMagnitude(Theta As Double, Magnitude As Double) As Vector3d
- Function Vector3dFromXYZ(Ax As Double, Ay As Double, Az As Double) As Vector3d
- Function Vector3dGetComponent(Vector As Vector3d, Index As Long) As Double
- Function Vector3dInterpolate(Vector0 As Vector3d, FractionParameter As Double, Vector1 As Vector3d) As Vector3d
- Function Vector3dIsVectorInCCWSector(TestVector As Vector3d, Vector0 As Vector3d, Vector1 As Vector3d, UpVector As Vector3d) As Boolean
- Function Vector3dIsVectorInCCWXYSector(TestVector As Vector3d, Vector0 As Vector3d, Vector1 As Vector3d) As Boolean
- Function Vector3dIsVectorInSmallerSector(TestVector As

Cuiing BDN

Vector3d, Vector0 As Vector3d, Vector1 As Vector3d) As Boolean

- Function Vector3dMagnitude(Vector As Vector3d) As Double
- Function Vector3dMagnitudeSquared(Vector As Vector3d) As Double
- Function Vector3dMagnitudeSquaredXY(Vector As Vector3d) As Double
- Function Vector3dMagnitudeXY(Vector As Vector3d) As Double
- Function Vector3dMaxAbs(Vector As Vector3d) As Double
- Function Vector3dMaxAbsDifference(Vector1 As Vector3d, Vector2 As Vector3d) As Double
- Function Vector3dNegate(Vector As Vector3d) As Vector3d
- Function Vector3dNormalize(Vector As Vector3d) As Vector3d
- Function Vector3dOne() As Vector3d
- Function Vector3dPlanarAngleBetweenVectors(Vector1 As Vector3d, Vector2 As Vector3d, PlaneNormal As Vector3d) As Double
- Function Vector3dPolarAngle(Vector As Vector3d) As Double
- Function Vector3dRotateXY(Vector As Vector3d, Theta As Double) As Vector3d
- Function Vector3dScale(Vector As Vector3d, Scale As Double) As Vector3d
- Function Vector3dSignedAngleBetweenVectors(Vector1 As Vector3d, Vector2 As Vector3d, OrientationVector As Vector3d) As Double
- Function Vector3dSmallerAngleBetweenUnorientedVectors(Vector1 As Vector3d, Vector2 As Vector3d) As Double
- Function Vector3dSmallerAngleBetweenUnorientedVectorsXY

(Vector1 As Vector3d, Vector2 As Vector3d) As Double

- Function Vector3dSubtract(Vector1 As Vector3d, Vector2 As Vector3d) As Vector3d
- Function Vector3dSubtractPoint3dPoint3d(Target As Point3d, Base As Point3d) As Vector3d
- Function Vector3dTripleProduct(Vector1 As Vector3d, Vector2 As Vector3d, Vector3 As Vector3d) As Double
- Function Vector3dUnitPerpendicularXY(Vector As Vector3d) As Vector3d
- Function Vector3dZero() As Vector3d
- Property Version As String {read-only}
- Property Visible As Boolean
- Property Width As Long

ApplicationObjectConnector

- Property Application As Application {readonly}

Attachment

- Sub Activate()
- Sub AddElement(Element As Element)
- Sub AddElements(Elements() As _Element)
- Function AddNewNamedGroup([Name As String], [Description As String]) As NamedGroupElement
- Sub AddUserAttributeData(AttributeID As Long, AttributeData As DataBlock)
- Property AnyElementsSelected As Boolean {read-only}
- Property AsAttachment As Attachment {readonly}
- Property AttachmentOrigin As Point3d {readonly}
- Property Attachments As Attachments {readonly}
- Property AttachName As String {read-only}
- Property CanBePlacedAsCell As Boolean
- Property CellType As MsdCellType

China BDN

- Property ControlElementCache As ElementCache {read-only}
- Function Copy() As Attachment
- Function CopyElement(Element As Element,
  [CopyContext As CopyContext]) As Element
- Sub DeleteAllXData()
- Function DeleteUserAttributeData(AttributeID
  As Long, Index As Integer) As Integer
- Sub DeleteXData(ApplicationName As String)
- Property Description As String
- Property DesignFile As DesignFile {read-only}
- Property DisplayAsNested As Boolean
- Property DisplayFlag As Boolean
- Property DisplayPriority As Long
- Property DisplaysRasterReferences As Boolean
- Function DoubleToWorkingUnits(Value As Double) As String
- Function
  ElementCacheContainingFilePosition(FilePosition
  As Long, [CacheIndex As Long]) As ElementCache
- Property ElementID As DLong {read-only}
- Property ElementsLocatable As Boolean
- Property ElementsSnappable As Boolean
- Property ElementsVisible As Boolean
- Function GetElementByID(ElementID As DLong) As Element
- Function GetElementByID64(ElementID64 As Empty) As Element
- Function GetLastValidGraphicalElement() As Element
- Function GetMasterToReferenceTransform() As Transform3d
- Function GetNamedGroup(GroupName As String) As
  NamedGroupElement
- Function GetReferenceToMasterTransform() As Transform3d
- Function GetSelectedElements() As ElementEnumerator

- Function GetSheetDefinition() As SheetDefinition
- Function GetUserAttributeData(AttributeID As Long) As DataBlock()
- Function GetXData(ApplicationName As String) As XDatum()
- Function GetXDataApplicationNames() As String()()
- Property GlobalLineStyleScale As MsdGlobalLineStyleScale
- Property GlobalOrigin As Point3d {read-only}
- Property GraphicalElementCache As ElementCache {read-only}
- Function HasAnyXData() As Boolean
- Function HasXData(ApplicationName As String) As Boolean
- Property Is3D As Boolean {read-only}
- Property IsActive As Boolean {read-only}
- Property IsAttachment As Boolean {read-only}
- Property IsElementSelected As Boolean {readonly}
- Property IsLocked As Boolean
- Property IsMissingFile As Boolean {read-only}
- Property IsMissingModel As Boolean {readonly}
- Property IsReadOnly As Boolean {read-only}
- Property IsTrueScale As Boolean {read-only}
- Property Level As Level
- Property Levels As Levels {read-only}
- Property LineStylesScaled As Boolean
- Property LogicalDescription As String
- Property LogicalName As String
- Property ManipulateAsElement As Boolean
- Property MasterOrigin As Point3d {read-only}
- Property MasterUnit As MeasurementUnit
- Function MdlModelRefP() As Long
- Sub Move(Offset As Point3d, ApplyToClipElement As Boolean)
- Property Name As String
- Property NamedGroup As String

- Property NestLevel As Long
- Property NestOverrides As MsdNestOverrides
- Property NewLevelDisplay As MsdNewLevelDisplay
- Property ParentModelReference As ModelReference {read-only}
- Property Plot3d As Boolean
- Property Presentation As MsdRenderingMode
- Property PrintColorAdjustment As Boolean
- Sub PropagateAnnotationScale()
- Function Range(IncludeAttachments As Boolean) As Range3d
- Function Reattach(FileName As String,
  ModelName As String) As Attachment
- Sub Redraw([DrawMode As MsdDrawingMode =
  msdDrawingModeNormal])
- Sub RemoveElement(Element As Element)
- Sub ReplaceElement(OldElement As Element,
  NewElement As Element)
- Property RevisionNumber As String
- Sub Rewrite()
- Sub Rotate(Pivot As Point3d, AboutX As Double,
  AboutY As Double, AboutZ As Double, ViewSpecifier As Variant)
- Property Rotation As Matrix3d {read-only}
- Property ScaleFactor As Double
- Property ScaleMasterUnits As Double {readonly}
- Sub ScaleUniform(Origin As Point3d,
  ScaleFactor As Double, ApplyToClipElement As Boolean)
- Function Scan([ScanCriteria As
  ElementScanCriteria]) As ElementEnumerator
- Sub SelectElement(Element As Element,
  [DisplayAsSelected As Boolean = True])
- Sub SetAttachNameDeferred(FileSpecification As String)

China BDN

- Sub SetSheetDefinition(NewDefinition As SheetDefinition)
- Sub SetXData(ApplicationName As String, NewXData() As XDatum)
- Property StorageUnit As MeasurementUnit
- Property SubUnit As MeasurementUnit
- Property SubUnitsPerMasterUnit As Double {read-only}
- Property Transparency As Double
- Property Type As MsdModelType
- Sub UnselectAllElements()
- Sub UnselectElement(Element As Element)
- Property UORsPerMasterUnit As Double {readonly}
- Property UORsPerStorageUnit As Double
- Property UORsPerSubUnit As Double {read-only}
- Property UpdateOrder As Long
- Property UsesLights As Boolean
- Function WorkingUnitsToDouble(Value As String) As Double

Attachments

- Function Add(FileSpecification As String, ModelName As String, LogicalName As String, Description As String, MasterOrigin As Point3d, ReferenceOrigin As Point3d, [TrueScale As Boolean = True], [DisplayImmediately As Boolean = True]) As Attachment
- Function AddCoincident(FileSpecification As String, ModelName As String, LogicalName As String, Description As String, [DisplayImmediately As Boolean = True]) As Attachment
- Function AddCoincident1(FileSpecification As String, ModelName As String, LogicalName As String, Description As String, Flags As

China BDN

MsdAddAttachmentFlags) As Attachment

- Function AddUsingNamedView(FileSpecification As String, LogicalName As String, Description As String, ViewName As String, CenterPoint As Point3d, [DisplayImmediately As Boolean = True]) As Attachment
- Function AddUsingNamedView1(FileSpecification As String, ModelName As String, LogicalName As String, Description As String, ViewName As String, CenterPoint As Point3d, Flags As MsdAddAttachmentFlags) As Attachment
- Property Count As Long {read-only}
- Function FindByLogicalName(LogicalName As String) As Attachment
- Property Item As Attachment {read-only}
- Sub Remove(AttachmentSpecifier As Variant)

CadInputMessage

- Property CommandKeyin As String {read-only}
- Property CursorButton As Long {read-only}
- Property InputType As MsdCadInputType {readonly}
- Property Keyin As String {read-only}
- Property Point As Point3d {read-only}
- Property ScreenPoint As Point3d {read-only}
- Property View As View {read-only}
- CadInputQueue
- Function GetInput([Type1 As MsdCadInputType = msdCadInputTypeAny], [Type2 As MsdCadInputType], [Type3 As MsdCadInputType], [Type4 As MsdCadInputType]) As CadInputMessage
- Sub SendCommand(Command As String,

China BDN

[TreatLikeKeyboardInput As Boolean])

- Sub SendDataPoint(DataPoint As Point3d, [ViewSpecifier As Variant], [Qualifier As Long])
- Sub SendDataPointForLocate(ElementToLocate As Element, DataPoint As Point3d, [ViewSpecifier As Variant], [Qualifier As Long])
- Sub SendDragPoints(DownPoint As Point3d, UpPoint As Point3d, [ViewSpecifier As Variant], [Qualifier As Long])
- Sub SendKeyin(Keyin As String)
- Sub SendLastInput()
- Sub SendMessageToApplication(MdlApplication As String, Message As String)
- Sub SendReset()
- Sub SendTentativePoint(DataPoint As Point3d, ViewSpecifier As Variant)

DesignFile

- Function AddNewLevel(LevelName As String) As Level
- Sub AttachColorTable(ColorTable As ColorTable)
- Property Author As String
- Property Client As String
- Sub Close()
- Property Comments As String
- Property Company As String
- Function CustomPropertyExists(Name As String) As Boolean
- Property DateCreated As Date {read-only}
- Property DateLastPlotted As Date {read-only}
- Property DateLastSaved As Date {read-only}
- Property DefaultModelReference As ModelReference {read-only}
- Sub DeleteLevel(Level As Level)
- Property DimensionStyles As DimensionStyles {read-only}

China BDN

- Property Editor As String {read-only}
- Function ExtractColorTable() As ColorTable
- Property Fence As Fence {read-only}
- Function FindSavedView(NamePattern As String, [PreviouslyFoundSavedView As SavedViewElement], [Namespace As String]) As SavedViewElement
- Property Fonts As Fonts {read-only}
- Property Format As MsdDesignFileFormat {readonly}
- Property FormatMajorVersion As Long {readonly}
- Property FormatMinorVersion As Long {readonly}
- Property FullName As String {read-only}
- Function GetCustomProperty(Name As String) As Variant
- Function GetElementByID(ElementID As DLong) As Element
- Function GetElementByID64(ElementID64 As Empty) As Element
- Function GetLargestElementID() As DLong
- Sub GetLargestElementID64()
- Property IsActive As Boolean {read-only}
- Property Keywords As String
- Property LastSavedBy As String
- Property Levels As Levels {read-only}
- Property LineStyles As LineStyles {read-only}
- Property Manager As String
- Function MdlFileObjP() As Long
- Function MdlModelRefP() As Long
- Property Models As ModelReferences {read-only}
- Property Name As String {read-only}
- Property NonModelElementCache As ElementCache {read-only}
- Property Path As String {read-only}

- Property RevisionNumber As String {read-only}
- Sub RewriteLevels()
- Sub Save()
- Sub SaveAs(NewFileName As String, [Overwrite As Boolean = False], [NewFormat As MsdDesignFileFormat = msdDesignFileFormatCurrent])
- Sub SetCustomProperty(Name As String, Value As Variant)
- Property Subject As String
- Property TagSets As TagSets {read-only}
- Property TextStyles As TextStyles {read-only}
- Property Title As String
- Property TotalEditingTime As Long {read-only}
- Property ViewGroups As ViewGroups {read-only}
- Property Views As Views {read-only}

Element

- Sub AddDatabaseLink(LinkToAdd As DatabaseLink)
- Function AddTag(TagDefinition As TagDefinition) As TagElement
- Function AddTags(TagSet As TagSet) As TagElement()
- Sub AddUserAttributeData(AttributeID As Long, AttributeData As DataBlock)
- Function ApparentColor(View As View) As Long
- Function ApparentLineStyle(View As View) As LineStyle
- Function ApparentLineWeight(View As View) As Long
- Property AsApplicationElement As ApplicationElement {read-only}
- Property AsArcElement As ArcElement {readonly}
- Property AsAuxiliaryCoordinateSystemElement As AuxiliaryCoordinateSystemElement {read-only}
- Property AsBsplineCurveElement As BsplineCurveElement {read-only}

China BDN

- Property AsBsplineSurfaceElement As BsplineSurfaceElement {read-only}
- Property AsCellElement As CellElement {readonly}
- Property AsChainableElement As ChainableElement {read-only}
- Property AsClosedElement As ClosedElement {read-only}
- Property AsComplexElement As ComplexElement {read-only}
- Property AsComplexShapeElement As ComplexShapeElement {read-only}
- Property AsComplexStringElement As ComplexStringElement {read-only}
- Property AsConeElement As ConeElement {readonly}
- Property AsCurveElement As CurveElement {read-only}
- Property AsDimensionElement As DimensionElement {read-only}
- Property AsDroppableElement As DroppableElement {read-only}
- Property AsEllipseElement As EllipseElement {read-only}
- Property AsEllipticalElement As EllipticalElement {read-only}
- Property AsIntersectableElement As IntersectableElement {read-only}
- Property AsLineElement As LineElement {readonly}
- Property AsMultiLineElement As MultiLineElement {read-only}
- Property AsNamedGroupElement As NamedGroupElement {read-only}
- Property AsPlanarElement As PlanarElement {read-only}
- Property AsPointStringElement As PointStringElement {read-only}
- Property AsPossiblyPlanarElement As PossiblyPlanarElement {read-only}
- Property AsSavedViewElement As SavedViewElement {read-only}
- Property AsShapeElement As ShapeElement {read-only}
- Property AsSharedCellDefinitionElement As SharedCellDefinitionElement {read-only}
- Property AsSharedCellElement As SharedCellElement {read-only}
- Property AsTagElement As TagElement {readonly}

- Property AsTextElement As TextElement {readonly}
- Property AsTextNodeElement As TextNodeElement {read-only}
- Property AsTraversableElement As TraversableElement {read-only}
- Property AsVertexList As VertexList {readonly}
- Property Cache As ElementCache {read-only}
- Property CacheIndex As Long {read-only}
- Property Class As MsdElementClass
- Function Clone() As Element
- Property Color As Long
- Function ConstructVertexList(Tolerance As Double) As Point3d()
- Property DateLastModified As Date {read-only}
- Sub DeleteAllTags()
- Sub DeleteAllXData()
- Sub DeleteTag(Tag As TagElement)
- Sub DeleteTagSet(TagSet As TagSet)
- Function DeleteUserAttributeData(AttributeID As Long, Index As Integer) As Integer
- Sub DeleteXData(ApplicationName As String)
- Sub DrawToFile(FileName As String, Width As Long, Height As Long, [DrawBackGround As Boolean = False])
- Property FilePosition As Long {read-only}
- Function GetContainingNamedGroups() As NamedGroupElement()
- Function GetDatabaseLinks([DatabaseType As MsdDatabaseLinkage], [EntityNumber As Long]) As DatabaseLink()
- Function GetPicture(Width As Long, Height As Long, [DrawBackGround As Boolean = False]) As Unknown
- Function GetRelatedElements(Locked As Boolean,

Cuiha BDN

[TraverseType As MsdMemberTraverseType = msdMemberTraverseCopy], [NewTraversal As Boolean = True]) As ElementEnumerator

- Function GetTag(TagSet As TagSet, TagName As String) As TagElement
- Function GetTags() As TagElement()
- Function GetUserAttributeData(AttributeID As Long) As DataBlock()
- Function GetXData(ApplicationName As String) As XDatum()
- Function GetXDataApplicationNames() As String()()
- Property GraphicGroup As Long
- Function HasAnyDatabaseLinks([DatabaseType As MsdDatabaseLinkage], [EntityNumber As Long]) As Boolean
- Property HasAnyTags As Boolean {read-only}
- Function HasAnyXData() As Boolean
- Function HasXData(ApplicationName As String) As Boolean
- Property ID As DLong {read-only}
- Property ID64 As Empty {read-only}
- Property InDisplaySet As Boolean
- Property IsApplicationElement As Boolean {read-only}
- Property IsArcElement As Boolean {read-only}
- Property IsAuxiliaryCoordinateSystemElement As Boolean {read-only}
- Property IsBsplineCurveElement As Boolean {read-only}
- Property IsBsplineSurfaceElement As Boolean {read-only}
- Property IsCellElement As Boolean {read-only}
- Property IsChainableElement As Boolean {readonly}
- Property IsClosedElement As Boolean {readonly}
- Property IsComplexElement As Boolean {readonly}
- Property IsComplexShapeElement As Boolean {read-only}

China BDN

- Property IsComplexStringElement As Boolean {read-only}
- Property IsComponentElement As Boolean {readonly}
- Property IsConeElement As Boolean {read-only}
- Property IsCurveElement As Boolean {read-only}
- Property IsDimensionElement As Boolean {readonly}
- Property IsDroppableElement As Boolean {readonly}
- Property IsEllipseElement As Boolean {readonly}
- Property IsEllipticalElement As Boolean {read-only}
- Property IsGraphical As Boolean {read-only}
- Property IsHidden As Boolean
- Property IsIntersectableElement As Boolean {read-only}
- Property IsLineElement As Boolean {read-only}
- Property IsLocked As Boolean
- Property IsModified As Boolean {read-only}
- Property IsMultiLineElement As Boolean {readonly}
- Property IsNamedGroupElement As Boolean {read-only}
- Property IsNew As Boolean {read-only}
- Property IsPlanarElement As Boolean {readonly}
- Property IsPointStringElement As Boolean {read-only}
- Property IsPossiblyPlanarElement As Boolean {read-only}
- Property IsSavedViewElement As Boolean {readonly}
- Property IsShapeElement As Boolean {read-only}
- Property IsSharedCellDefinitionElement As Boolean {read-only}
- Property IsSharedCellElement As Boolean {read-only}
- Property IsSnappable As Boolean
- Property IsTagElement As Boolean {read-only}
- Property IsTextElement As Boolean {read-only}
- Property IsTextNodeElement As Boolean {readonly}
- Property IsTraversableElement As Boolean {read-only}
- Property IsValid As Boolean {read-only}

- Property IsVertexList As Boolean {read-only}
- Property Level As Level
- Property LineStyle As LineStyle
- Property LineWeight As Long
- Function MdlElementDescrP([Detach As Boolean]) As Long
- Sub Mirror(Point1 As Point3d, Point2 As Point3d)
- Sub Mirror3d(PlanePoint1 As Point3d, PlanePoint2 As Point3d, PlanePoint3 As Point3d)
- Property ModelReference As ModelReference {read-only}
- Sub Move(Offset As Point3d)
- Sub PartialDelete(Partial1 As Element, Partial2 As Element, Point1 As Point3d, Point2 As Point3d, Selector As Point3d, ViewSpecifier As Variant)
- Property Range As Range3d {read-only}
- Sub Redraw([DrawMode As MsdDrawingMode = msdDrawingModeNormal])
- Sub RemoveAllDatabaseLinks()
- Function RemoveDatabaseLink(DatabaseType As MsdDatabaseLinkage, [MSLinkNumber As Long], [EntityNumber As Long]) As Long
- Sub Rewrite()
- Sub Rotate(Pivot As Point3d, AboutX As Double, AboutY As Double, AboutZ As Double)
- Sub RotateAboutZ(Pivot As Point3d, Angle As Double)
- Sub ScaleAll(Origin As Point3d, XFactor As Double, YFactor As Double, ZFactor As Double)
- Sub ScaleUniform(Origin As Point3d, ScaleFactor As Double)
- Sub SetXData(ApplicationName As String, NewXData() As XDatum)

China BDN

- Property Subtype As MsdElementSubtype {readonly}
- Sub Transform(Transform3d As Transform3d)
- Property Type As MsdElementType {read-only}
- Property URL As String {read-only}
- Property URLTitle As String {read-only}

ElementEnumerator

- Function BuildArrayFromContents() As Element()
- Function Clone() As ElementEnumerator
- Property Current As Element {read-only}
- Function MoveNext() As Boolean
- Sub Reset()

ElementScanCriteria

- Sub ExcludeAllClasses()
- Sub ExcludeAllColors()
- Sub ExcludeAllLevels()
- Sub ExcludeAllLineStyles()
- Sub ExcludeAllLineWeights()
- Sub ExcludeAllSubtypes()
- Sub ExcludeAllTypes()
- Sub ExcludeGraphical()
- Sub ExcludeNonGraphical()
- Sub IncludeClass(ElemClass As MsdElementClass)
- Sub IncludeColor(ColorIndex As Long)
- Sub IncludeLevel(Level As Level)
- Sub IncludeLineStyle(LineStyle As LineStyle)
- Sub IncludeLineWeight(LineWeight As Long)
- Sub IncludeOnlyCell(CellName As String)
- Sub IncludeOnlyFilePositionRange(Min As Long, Max As Long)
- Sub IncludeOnlyGraphicGroup(GraphicGroupNumber As Long)
- Sub IncludeOnlyHole()

Sinia BDN

- Sub IncludeOnlyInvisible()
- Sub IncludeOnlyLocked()
- Sub IncludeOnlyModified()
- Sub IncludeOnlyModifiedRange(Min As Date, [Max As Date])
- Sub IncludeOnlyNew()
- Sub IncludeOnlyNonPlanar()
- Sub IncludeOnlyNonSnappable()
- Sub IncludeOnlyOld()
- Sub IncludeOnlyPlanar()
- Sub IncludeOnlySnappable()
- Sub IncludeOnlySolid()
- Sub IncludeOnlyUnlocked()
- Sub IncludeOnlyUnmodified()
- Sub IncludeOnlyUserAttribute(UserAttributeID As Long)
- Sub IncludeOnlyVisible()
- Sub IncludeOnlyWithinRange(Range As Range3d)
- Sub IncludeSubtype(Long As Long)
- Sub IncludeType(Type As MsdElementType)
- Sub Reset()

Level

- Sub AddUserAttributeData(AttributeID As Long, AttributeData As DataBlock)
- Function DeleteUserAttributeData(AttributeID As Long, Index As Integer) As Integer
- Property Description As String
- Property ElementAccess As MsdLevelElementAccess
- Property ElementColor As Long
- Property ElementLineStyle As LineStyle
- Property ElementLineWeight As Long

China BDN

- Function GetUserAttributeData(AttributeID As Long) As DataBlock()
- Property ID As Long {read-only}
- Property IsActive As Boolean
- Property IsDisplayed As Boolean
- Property IsDisplayedInView As Boolean
- Property IsEffectivelyDisplayedInView As Boolean {read-only}
- Property IsFromLevelLibrary As Boolean {readonly}
- Property IsFrozen As Boolean
- Property IsInUse As Boolean {read-only}
- Function IsInUseWithinModel(Model As ModelReference) As Boolean
- Property IsLocked As Boolean
- Property Name As String
- Property Number As Long
- Property OverrideColor As Long
- Property OverrideLineStyle As LineStyle
- Property OverrideLineWeight As Long
- Property ParentLevel As Level
- Property Plot As Boolean
- Property UsingOverrideColor As Boolean
- Property UsingOverrideLineStyle As Boolean
- Property UsingOverrideLineWeight As Boolean

ModelReference

- Sub Activate()
- Sub AddElement(Element As Element)
- Sub AddElements(Elements() As _Element)
- Function AddNewNamedGroup([Name As String], [Description As String]) As NamedGroupElement
- Sub AddUserAttributeData(AttributeID As Long, AttributeData As DataBlock)
- Property AnyElementsSelected As Boolean {read-only}

China BDN

- Property AsAttachment As Attachment {readonly}
- Property Attachments As Attachments {readonly}
- Property CanBePlacedAsCell As Boolean
- Property CellType As MsdCellType
- Property ControlElementCache As ElementCache {read-only}
- Function CopyElement(Element As Element,
  [CopyContext As CopyContext]) As Element
- Sub DeleteAllXData()
- Function DeleteUserAttributeData(AttributeID
  As Long, Index As Integer) As Integer
- Sub DeleteXData(ApplicationName As String)
- Property Description As String
- Property DesignFile As DesignFile {read-only}
- Function DoubleToWorkingUnits(Value As Double) As String
- Function
  ElementCacheContainingFilePosition(FilePosition
  As Long, [CacheIndex As Long]) As ElementCache
- Function GetElementByID(ElementID As DLong) As Element
- Function GetElementByID64(ElementID64 As Empty) As Element
- Function GetLastValidGraphicalElement() As Element
- Function GetNamedGroup(GroupName As String) As
  NamedGroupElement
- Function GetSelectedElements() As ElementEnumerator
- Function GetSheetDefinition() As SheetDefinition
- Function GetUserAttributeData(AttributeID As
  Long) As DataBlock()
- Function GetXData(ApplicationName As String) As XDatum()
- Function GetXDataApplicationNames() As String()()
- Property GlobalOrigin As Point3d {read-only}
- Property GraphicalElementCache As ElementCache {read-only}

China BDN

- Function HasAnyXData() As Boolean
- Function HasXData(ApplicationName As String) As Boolean
- Property Is3D As Boolean {read-only}
- Property IsActive As Boolean {read-only}
- Property IsAttachment As Boolean {read-only}
- Property IsElementSelected As Boolean {readonly}
- Property IsLocked As Boolean
- Property IsReadOnly As Boolean {read-only}
- Property Levels As Levels {read-only}
- Property MasterUnit As MeasurementUnit
- Function MdlModelRefP() As Long
- Property Name As String
- Property ParentModelReference As ModelReference {read-only}
- Sub PropagateAnnotationScale()
- Function Range(IncludeAttachments As Boolean) As Range3d
- Sub RemoveElement(Element As Element)
- Sub ReplaceElement(OldElement As Element, NewElement As Element)
- Function Scan([ScanCriteria As ElementScanCriteria]) As ElementEnumerator
- Sub SelectElement(Element As Element, [DisplayAsSelected As Boolean = True])
- Sub SetSheetDefinition(NewDefinition As SheetDefinition)
- Sub SetXData(ApplicationName As String, NewXData() As XDatum)
- Property StorageUnit As MeasurementUnit
- Property SubUnit As MeasurementUnit
- Property SubUnitsPerMasterUnit As Double {read-only}
- Property Type As MsdModelType
- Sub UnselectAllElements()
- Sub UnselectElement(Element As Element)

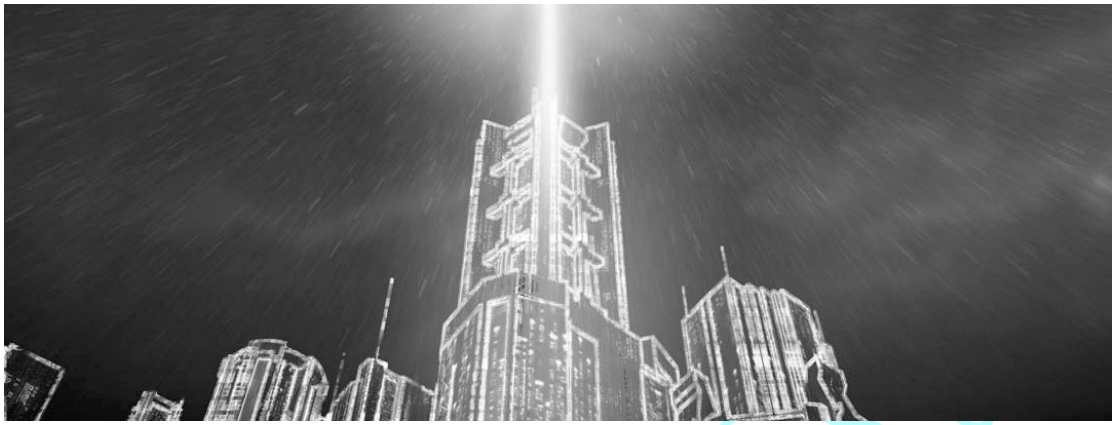
- Property UORsPerMasterUnit As Double {readonly}
- Property UORsPerStorageUnit As Double
- Property UORsPerSubUnit As Double {read-only}
- Function WorkingUnitsToDouble(Value As String) As Double

本章回顾

我们已经展示了在 MicroStation 中可用的一小部分对象。时不时地看一看本章中的列表(甚至是列表的一部分)很有帮助。

要想深入了解任何库的对象模型，VBA 中的对象浏览器最有用。VBA 还包括其它有助于开发的工具，如添加监控和自动列表功能。

China BDN



12. MicroStation 对象模型—枚举

什么是枚举(Enum)？枚举的全称是 Enumeration，它是能在代码中使用的常数集合。

让我们考察如下枚举：

MsdDesignFileFormat

msdDesignFileFormatCurrent = 0

msdDesignFileFormatDWG = 3

msdDesignFileFormatDXF = 4

msdDesignFileFormatUnknown = -1

msdDesignFileFormatV7 = 1

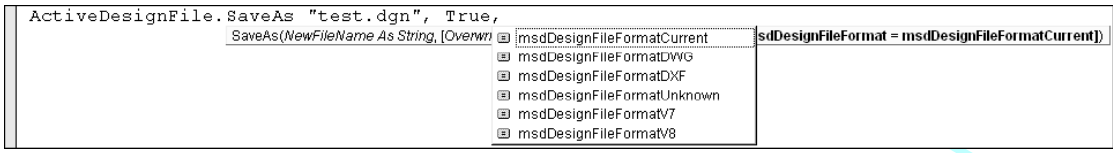
msdDesignFileFormatV8 = 2

该枚举名是“MsdDesignFileFormat”，它有六个成员，其值从-1 到 4。枚举中的每个成员都有名字和值，枚举成员“msdDesignFileFormatCurrent”的值是 0。正如前一章看到的那样，一些属性和方法使用这些枚举。例如，

```
Sub SaveAs(NewFileName As String, [Overwrite As _
Boolean = False], [NewFormat As MsdDesignFileFormat _
= msdDesignFileFormatCurrent])
```

SaveAs 方法能在 DesignFile 对象下找到，使用该方法时，指定一个文件名、是否覆盖现有的文件、所用的文件格式。“NewFileName”的数据类型是字符串，“Overwrite”的值类型是布尔型，“NewFormat”的值类型是

MsdDesignFileFormat。“NewFormat”参数利用了“MsdDesignFileFormat”枚举。使用 SaveAs 方法时会看到如下样子：



在 VBA 中，当遇到利用枚举的参数时，看到的是枚举成员的列表而不是每个成员的值。

枚举有几个好处，最大的一个就是在分析代码时能更容易地看到所需参数的结果。换句话说，在 NewFormat 中看到“msdDesignFileFormatDWG”比看到数字 3 要清楚。

```
ActiveDesignFile.SaveAs "test.dgn", True, _
                        MsdDesignFileFormat.msdDesignFileFormatDWG
```

使用枚举的另一种方式是不带枚举名，直接使用成员名：

```
ActiveDesignFile.SaveAs "test.dgn", True, msdDesignFileFormatDWG
```

枚举成员名经常以枚举名或枚举名的缩略版本开头，上例中的成员名“msdDesignFileFormatDWG”就是以枚举名“MsdDesignFileFormat”开头的。注意枚举名是如何在成员名前使用的。有时候成员名会使用枚举名的一个缩写，如“MsdCoordinateAccuracy”枚举中的成员是以“msdAccuracy”开头而不是以全枚举名“msdCoordinateAccuracy”开头。还要注意的是，MicroStation 中所有的枚举都以指定的三个字符“Msd”开头，所有的成员名都以“msd”开头。

枚举列表

MsdACSType

msdACSTypeCylindrical = 2

msdACSTypeNone = 0

msdACSTypeRectangular = 1

msdACSTypeSpherical = 3

MsdAddAttachmentFlags

msdAddAttachmentElementsVisible = 4

msdAddAttachmentFlagCoincidentWorld = 2

msdAddAttachmentFlagNone = 0

msdAddAttachmentFlagTrueScale = 1

MsdAngleAccuracy

msdAngleAccuracy0 = 0

msdAngleAccuracy1 = 1

msdAngleAccuracy2 = 2

msdAngleAccuracy3 = 3

msdAngleAccuracy4 = 4

msdAngleAccuracy5 = 5

msdAngleAccuracy6 = 6

msdAngleAccuracy7 = 7

msdAngleAccuracy8 = 8

MsdAngleFormat

msdFormatDD_DDDD = 0

msdFormatDD_MM_SS = 1

msdFormatGradians = 2

msdFormatRadians = 3

MsdAngleMode

msdAngleModeAzimuth = 1

msdAngleModeBearing = 2

msdAngleModeConventional = 0

MsdAttachMode

msdAttachNone = 1

msdAttachReference = 3

MsdBsplineCurveOffsetCuspType

msdBsplineCurveOffsetCuspArc = 4

msdBsplineCurveOffsetCuspChamfer = 1

msdBsplineCurveOffsetCuspJump = 0

msdBsplineCurveOffsetCuspParabola = 3

msdBsplineCurveOffsetCuspPoint = 2

China BDN

MsdBsplineCurveType

msdBsplineCurveCircle = 3

msdBsplineCurveCircularArc = 2

msdBsplineCurveEllipse = 5

msdBsplineCurveEllipticalArc = 4

msdBsplineCurveGeneral = 0

msdBsplineCurveHyperbolicArc = 7

msdBsplineCurveLine = 1

msdBsplineCurveParabolicArc = 6

MsdBsplineParametrizationType

msdBsplineParametrizationCentripetal = 2

msdBsplineParametrizationChordLength = 1

msdBsplineParametrizationInherited = 3

msdBsplineParametrizationUniform = 0

MsdBsplineSurfaceDirection

msdBsplineSurfaceU = 0

msdBsplineSurfaceV = 1

MsdBsplineSurfaceType

msdBsplineSurfaceCone = 3

msdBsplineSurfaceGeneral = 0

msdBsplineSurfacePlane = 1

msdBsplineSurfaceRevolution = 6

msdBsplineSurfaceRightCylinder = 2

msdBsplineSurfaceRuledSurface = 8

msdBsplineSurfaceSphere = 4

msdBsplineSurfaceTabCylinder = 7

msdBsplineSurfaceTorus = 5

MsdCadInputType

msdCadInputTypeAny = 5

msdCadInputTypeCommand = 1

China BDN

msdCadInputTypeDataPoint = 3

msdCadInputTypeKeyin = 4

msdCadInputTypeReset = 2

msdCadInputTypeUnassignedCB = 6

MsdCellType

msdCellTypeGraphic = 0

msdCellTypeMenu = 1

msdCellTypePoint = 7

MsdChangePropagation

msdChangePropagationAlways = 2

msdChangePropagationGroupLock = 0

msdChangePropagationNever = 1

MsdChangeTrackAction

msdChangeTrackActionAdd = 2

msdChangeTrackActionAppData = 8

msdChangeTrackActionDelete = 1

msdChangeTrackActionDrop = 6

msdChangeTrackActionMark = 7

msdChangeTrackActionModelAdd = 9

msdChangeTrackActionModelDelete = 10

msdChangeTrackActionModify = 3

msdChangeTrackActionModifyFence = 5

msdChangeTrackActionNewFilePositionAndModify = 4

MsdCommandResult

msdCommandResult3dLibrary2dFile = 50

msdCommandResult3dOnly = 39

msdCommandResultAcceptQuery = 68

msdCommandResultBadCellName = 47

msdCommandResultCellDeleted = 59

msdCommandResultCellExists = 55

China BDN

msdCommandResultCellLibraryNotFound = 12

msdCommandResultCellNestingError = 43

msdCommandResultCellNotFound = 44

msdCommandResultElementNotFound = 21

msdCommandResultEmptyFence = 27

msdCommandResultFileReadOnly = 287

msdCommandResultIllegalDefinition = 23

msdCommandResultInvalidReferenceOperation = 481

msdCommandResultNeedCharacters = 27

msdCommandResultNoActiveCell = 19

msdCommandResultNoCellLibrary = 54

msdCommandResultNoFenceActive = 15

msdCommandResultNoOrigin = 56

msdCommandResultOffDesignPlane = 22

msdCommandResultReferenceNotFound = 7

msdCommandResultSuccess = 0

msdCommandResultUnknownCommand = 16

msdCommandResultViewNotFound = 18

MsdConversionMode

msdConversionModeAlways = 1

msdConversionModeNever = 0

msdConversionModePrompt = 2

MsdCoordinateAccuracy

msdAccuracy0 = 1

msdAccuracy1 = 2

msdAccuracy16th = 56

msdAccuracy2 = 3

msdAccuracy3 = 4

msdAccuracy32nd = 120

msdAccuracy4 = 5

China BDN

msdAccuracy4th = 8

msdAccuracy5 = 6

msdAccuracy6 = 7

msdAccuracy64th = 248

msdAccuracy8th = 24

msdAccuracyHalf = 0

MsdCoordinateFormat

msdMasterUnits = 1

msdSubUnits = 0

msdWorkingUnits = 2

MsdCopyContextLevelOption

msdCopyContextLevelAlreadyRemapped = 4

msdCopyContextLevelByUserPreference = 0

msdCopyContextLevelCopyAlways = 3

msdCopyContextLevelCopyIfDifferent = 2

msdCopyContextLevelCopyIfNotFound = 1

MsdCopyViewPort

msdCopyViewPortApplyAspectRatio = 2

msdCopyViewPortApplySize = 3

msdCopyViewPortApplySizeAndPosition = 4

msdCopyViewPortKeepCurrent = 0

MsdDatabaseLinkage

msdDatabaseLinkageInformix = 1

msdDatabaseLinkageIngres = 32

msdDatabaseLinkageOdbc = 128

msdDatabaseLinkageOleDb = 256

msdDatabaseLinkageOracle = 8

msdDatabaseLinkageXBase = 4

MsdDataEntryRegionJustification

msdDataEntryRegionJustificationCenter = 0

China BDN

msdDataEntryRegionJustificationLeft = -1

msdDataEntryRegionJustificationRight = 1

MsdDesignFileFormat

msdDesignFileFormatCurrent = 0

msdDesignFileFormatDWG = 3

msdDesignFileFormatDXF = 4

msdDesignFileFormatUnknown = -1

msdDesignFileFormatV7 = 1

msdDesignFileFormatV8 = 2

MsdDevelopableElementOutputType

msdDevelopableCones = 4

msdDevelopableConesPlanar = 5

msdDevelopableRuleLines = 0

msdDevelopableRuleLinesPlanar = 1

msdDevelopableShapes = 2

msdDevelopableShapesPlanar = 3

MsdDialogBoxResult

msdDialogBoxResultApply = 1

msdDialogBoxResultCancel = 4

msdDialogBoxResultDefault = 5

msdDialogBoxResultHelp = 10

msdDialogBoxResultNo = 7

msdDialogBoxResultOK = 3

msdDialogBoxResultReset = 2

msdDialogBoxResultRetry = 8

msdDialogBoxResultStop = 9

msdDialogBoxResultYes = 6

msdDialogBoxResultYesToAll = 11

MsdDimAccuracy

msdDimAccuracy0 = 0

China BDN

msdDimAccuracy1 = 129

msdDimAccuracy16th = 8

msdDimAccuracy2 = 130

msdDimAccuracy3 = 132

msdDimAccuracy32nd = 16

msdDimAccuracy4 = 136

msdDimAccuracy4th = 2

msdDimAccuracy5 = 144

msdDimAccuracy6 = 160

msdDimAccuracy64th = 32

msdDimAccuracy7 = 192

msdDimAccuracy8 = 128

msdDimAccuracy8th = 4

msdDimAccuracyHalf = 1

msdDimAccuracySci1 = 64

msdDimAccuracySci2 = 65

msdDimAccuracySci3 = 66

msdDimAccuracySci4 = 67

msdDimAccuracySci5 = 68

msdDimAccuracySci6 = 69

msdDimAccuracySci7 = 70

msdDimAccuracySci8 = 71

MsdDimAlignment

msdDimAlignmentArbitrary = 3

msdDimAlignmentDrawing = 1

msdDimAlignmentTrue = 2

msdDimAlignmentView = 0

MsdDimAlternateThresholdComparison

MsdDimAlternateThresholdComparisonGreater = 1

MsdDimAlternateThresholdComparisonGreaterOrEqual = 3

China BDN

MsdDimAlternateThresholdComparisonLess = 0

MsdDimAlternateThresholdComparisonLessOrEqual = 2

MsdDimAngleMeasure

MsdDimAngleMeasureAngle = 1

MsdDimAngleMeasureArcLength = 0

MsdDimBallAndChainAlignment

msdDimBallAndChainAlignmentAuto = 0

msdDimBallAndChainAlignmentLeft = 1

msdDimBallAndChainAlignmentRight = 2

MsdDimBallAndChainChainType

msdDimBallAndChainChainTypeArc = 2

msdDimBallAndChainChainTypeBSpline = 3

msdDimBallAndChainChainTypeLine = 1

msdDimBallAndChainChainTypeNone = 0

MsdDimCustomSymbol

msdDimCustomSymbolCharacter = 1

msdDimCustomSymbolDefault = 0

MsdDimDMSPrecisionMode

MsdDimDMSPrecisionModeFixed = 0

MsdDimDMSPrecisionModeFloating = 1

MsdDimLabelLineFormat

MsdDimLabelLineFormatAngleAbove = 3

MsdDimLabelLineFormatAngleBelow = 5

MsdDimLabelLineFormatAngleOverLength = 1

MsdDimLabelLineFormatLengthAbove = 2

MsdDimLabelLineFormatLengthAngleAbove = 6

MsdDimLabelLineFormatLengthAngleBelow = 7

MsdDimLabelLineFormatLengthBelow = 4

MsdDimLabelLineFormatStandard = 0

MsdDimMLNoteFrameType

China BDN

msdDimMLNoteFrameTypeBox = 2

msdDimMLNoteFrameTypeLine = 1

msdDimMLNoteFrameTypeNone = 0

MsdDimMLNoteJustification

msdDimMLNoteJustificationCenter = 3

msdDimMLNoteJustificationDynamic = 2

msdDimMLNoteJustificationLeft = 0

msdDimMLNoteJustificationRight = 1

MsdDimNoteHorizontalAttachment

msdDimNoteHorizontalAttachmentAuto = 0

msdDimNoteHorizontalAttachmentLeft = 1

msdDimNoteHorizontalAttachmentRight = 2

MsdDimNoteLeaderType

MsdDimNoteLeaderTypeCurve = 1

MsdDimNoteLeaderTypeLine = 0

MsdDimNoteTextRotation

msdDimNoteTextRotationHorizontal = 0

msdDimNoteTextRotationInline = 2

msdDimNoteTextRotationVertical = 1

MsdDimNoteVerticalAttachment

msdDimNoteVerticalAttachmentBottom = 4

msdDimNoteVerticalAttachmentBottomLine = 3

msdDimNoteVerticalAttachmentDynamicCorner = 6

msdDimNoteVerticalAttachmentDynamicLine = 5

msdDimNoteVerticalAttachmentMiddle = 2

msdDimNoteVerticalAttachmentTop = 0

msdDimNoteVerticalAttachmentTopLine = 1

msdDimNoteVerticalAttachmentUnderline = 7

MsdDimNoteVerticalJustification

msdDimNoteVerticalJustificationBottom = 2

China BDN

msdDimNoteVerticalJustificationCenter = 1

msdDimNoteVerticalJustificationDynamic = 3

msdDimNoteVerticalJustificationTop = 0

MsdDimPlacementTextPosition

msdDimPlacementTextPositionAuto = 2

msdDimPlacementTextPositionManual = 0

msdDimPlacementTextPositionSemiAuto = 1

MsdDimRadialMode

msdDimRadialModeCenterMark = 0

msdDimRadialModeDiameter = 3

msdDimRadialModeDiameterExtended = 4

msdDimRadialModeRadius = 1

msdDimRadialModeRadiusExtended = 2

MsdDimStackedFractionAlignment

MsdDimStackedFractionAlignmentBottom = 2

MsdDimStackedFractionAlignmentCenter = 1

MsdDimStackedFractionAlignmentTop = 0

MsdDimStackedFractionType

MsdDimStackedFractionTypeDiagonal = 2

MsdDimStackedFractionTypeFromFont = 0

MsdDimStackedFractionTypeHorizontal = 1

MsdDimStyleProp

msdDimStylePropBallAndChainAlignment = 101

msdDimStylePropBallAndChainChainTerminator = 102

msdDimStylePropBallAndChainChainType = 103

msdDimStylePropBallAndChainIsActive = 104

msdDimStylePropBallAndChainNoDockOnDimLine = 106

msdDimStylePropBallAndChainShowTextLeader = 105

msdDimStylePropExtensionLineAngleChordAlign = 213

msdDimStylePropExtensionLineColor = 201

msdDimStylePropExtensionLineExtend = 202

msdDimStylePropExtensionLineJoin = 203

msdDimStylePropExtensionLineLeft = 204

msdDimStylePropExtensionLineLineStyle = 205

msdDimStylePropExtensionLineOffset = 206

msdDimStylePropExtensionLineOverrideColor = 207

msdDimStylePropExtensionLineOverrideLineStyle = 208

msdDimStylePropExtensionLineOverrideWeight = 209

msdDimStylePropExtensionLineRight = 210

msdDimStylePropExtensionLineShowAny = 211

msdDimStylePropExtensionLineWeight = 212

msdDimStylePropGeneralAlignment = 301

msdDimStylePropGeneralCenterMarkSize = 302

msdDimStylePropGeneralColor = 303

msdDimStylePropGeneralDimensionScale = 304

msdDimStylePropGeneralDimStyleDescription = 305

msdDimStylePropGeneralDimStyleName = 306

msdDimStylePropGeneralFont = 307

msdDimStylePropGeneralIgnoreLevelSymbology = 308

msdDimStylePropGeneralLineStyle = 309

msdDimStylePropGeneralOverrideColor = 310

msdDimStylePropGeneralOverrideLineStyle = 311

msdDimStylePropGeneralOverrideWeight = 312

msdDimStylePropGeneralRadialMode = 313

msdDimStylePropGeneralRelativeDimLine = 314

msdDimStylePropGeneralShowCenterMark = 315

msdDimStylePropGeneralStacked = 316

msdDimStylePropGeneralStackOffset = 317

msdDimStylePropGeneralWeight = 318

msdDimStylePropInvalid = 0

msdDimStylePropMLNoteElbowLength = 108

msdDimStylePropMLNoteFrameType = 401

msdDimStylePropMLNoteHorAttachment = 407

msdDimStylePropMLNoteJustification = 402

msdDimStylePropMLNoteLeaderType = 405

msdDimStylePropMLNoteLeftMargin = 410

msdDimStylePropMLNoteLowerMargin = 411

msdDimStylePropMLNoteShowLeader = 403

msdDimStylePropMLNoteTextRotation = 406

msdDimStylePropMLNoteVerLeftAttachment = 408

msdDimStylePropMLNoteVerRightAttachment = 409

msdDimStylePropMLNoteVerticalJustification = 404

msdDimStylePropPlacementAnnotationScale = 507

msdDimStylePropPlacementCompatibleV3 = 501

msdDimStylePropPlacementLevel = 502

msdDimStylePropPlacementNotUseModelAnnotationScale = 506

msdDimStylePropPlacementOverrideLevel = 503

msdDimStylePropPlacementTextPosition = 504

msdDimStylePropPlacementUseReferenceScale = 505

msdDimStylePropSymbolDiameterChar = 601

msdDimStylePropSymbolDiameterFont = 602

msdDimStylePropSymbolDiameterType = 603

msdDimStylePropSymbolLowerPrefixChar = 604

msdDimStylePropSymbolLowerSuffixChar = 605

msdDimStylePropSymbolMainPrefixChar = 606

msdDimStylePropSymbolMainSuffixChar = 607

msdDimStylePropSymbolPlusMinusChar = 608

msdDimStylePropSymbolPlusMinusType = 609

msdDimStylePropSymbolPrefix = 610

msdDimStylePropSymbolPrefixCellName = 611

CSDN @Qing

msdDimStylePropSymbolPrefixChar = 612

msdDimStylePropSymbolPrefixFont = 613

msdDimStylePropSymbolPrefixType = 614

msdDimStylePropSymbolSuffix = 615

msdDimStylePropSymbolSuffixCellName = 616

msdDimStylePropSymbolSuffixChar = 617

msdDimStylePropSymbolSuffixFont = 618

msdDimStylePropSymbolSuffixType = 619

msdDimStylePropSymbolTolPrefixChar = 620

msdDimStylePropSymbolTolSuffixChar = 621

msdDimStylePropSymbolUpperPrefixChar = 622

msdDimStylePropSymbolUpperSuffixChar = 623

msdDimStylePropTerminatorArrowCellName = 701

msdDimStylePropTerminatorArrowChar = 702

msdDimStylePropTerminatorArrowFont = 703

msdDimStylePropTerminatorArrowhead = 729

msdDimStylePropTerminatorArrowType = 704

msdDimStylePropTerminatorColor = 705

msdDimStylePropTerminatorDotCellName = 706

msdDimStylePropTerminatorDotChar = 707

msdDimStylePropTerminatorDotFont = 708

msdDimStylePropTerminatorDotType = 709

msdDimStylePropTerminatorFirst = 710

msdDimStylePropTerminatorHeight = 711

msdDimStylePropTerminatorJoint = 712

msdDimStylePropTerminatorLeft = 713

msdDimStylePropTerminatorLineStyle = 714

msdDimStylePropTerminatorMinLeader = 715

msdDimStylePropTerminatorMode = 716

msdDimStylePropTerminatorNoLineThruArrow = 717

msdDimStylePropTerminatorNoLineThruDot = 718

msdDimStylePropTerminatorNoLineThruOrigin = 719

msdDimStylePropTerminatorNoLineThruStroke = 720

msdDimStylePropTerminatorNote = 736

msdDimStylePropTerminatorNoteCellName = 738

msdDimStylePropTerminatorNoteChar = 739

msdDimStylePropTerminatorNoteFont = 740

msdDimStylePropTerminatorNoteType = 737

msdDimStylePropTerminatorOriginCellName = 721

msdDimStylePropTerminatorOriginChar = 722

msdDimStylePropTerminatorOriginFont = 723

msdDimStylePropTerminatorOriginType = 724

msdDimStylePropTerminatorOverrideColor = 725

msdDimStylePropTerminatorOverrideLineStyle = 726

msdDimStylePropTerminatorOverrideWeight = 727

msdDimStylePropTerminatorRight = 728

msdDimStylePropTerminatorStrokeCellName = 730

msdDimStylePropTerminatorStrokeChar = 731

msdDimStylePropTerminatorStrokeFont = 732

msdDimStylePropTerminatorStrokeType = 733

msdDimStylePropTerminatorWeight = 734

msdDimStylePropTerminatorWidth = 735

msdDimStylePropTextArcLengthSymbol = 801

msdDimStylePropTextAutoLift = 802

msdDimStylePropTextCapsule = 804

msdDimStylePropTextColor = 805

msdDimStylePropTextDecimalComma = 806

msdDimStylePropTextFont = 808

msdDimStylePropTextFrameType = 837

msdDimStylePropTextHeight = 809

msdDimStylePropTextHorizontal = 810

msdDimStylePropTextHorizontalMargin = 811

msdDimStylePropTextInlineTextLift = 838

msdDimStylePropTextJustification = 812

msdDimStylePropTextLeadingZero = 813

msdDimStylePropTextLocation = 835

msdDimStylePropTextOmitLeadingDelimiter = 815

msdDimStylePropTextOverrideColor = 816

msdDimStylePropTextOverrideHeight = 817

msdDimStylePropTextOverrideStackedFractions = 833

msdDimStylePropTextOverrideUnderline = 834

msdDimStylePropTextOverrideWeight = 818

msdDimStylePropTextOverrideWidth = 819

msdDimStylePropTextSecLeadingZero = 820

msdDimStylePropTextShowSecondary = 821

msdDimStylePropTextStackedFractionAlignment = 829

msdDimStylePropTextStackedFractions = 830

msdDimStylePropTextStackedFractionScale = 832

msdDimStylePropTextStackedFractionType = 831

msdDimStylePropTextSuperscriptMode = 839

msdDimStylePropTextTextStyle = 827

msdDimStylePropTextTextStyleID = 828

msdDimStylePropTextUnderline = 822

msdDimStylePropTextVerticalMargin = 824

msdDimStylePropTextVerticalOpts = 836

msdDimStylePropTextWeight = 825

msdDimStylePropTextWidth = 826

msdDimStylePropToleranceAccuracy = 910

msdDimStylePropToleranceLowerValue = 901

msdDimStylePropToleranceMode = 902

Qinqa BDN

msdDimStylePropToleranceSecAccuracy = 911

msdDimStylePropToleranceShow = 903

msdDimStylePropToleranceStackEqual = 904

msdDimStylePropToleranceTextHorizontalMargin = 905

msdDimStylePropToleranceTextScale = 906

msdDimStylePropToleranceTextVerticalMargin = 907

msdDimStylePropToleranceTextVerticalSeparation = 908

msdDimStylePropToleranceUpperValue = 909

msdDimStylePropValueAccuracy = 1001

msdDimStylePropValueAltAccuracy = 1002

msdDimStylePropValueAltFormat = 1067

msdDimStylePropValueAltIsActive = 1003

msdDimStylePropValueAltSecAccuracy = 1004

msdDimStylePropValueAltSecFormat = 1069

msdDimStylePropValueAltSecIsActive = 1005

msdDimStylePropValueAltSecShowZeroMasterUnit = 1012

msdDimStylePropValueAltSecShowZeroSubUnit = 1081

msdDimStylePropValueAltSecThreshold = 1013

msdDimStylePropValueAltSecThresholdComparison = 1071

msdDimStylePropValueAltShowZeroMasterUnit = 1020

msdDimStylePropValueAltShowZeroSubUnit = 1079

msdDimStylePropValueAltThreshold = 1021

msdDimStylePropValueAltThresholdComparison = 1070

msdDimStylePropValueAngleFormat = 1023

msdDimStylePropValueAngleLeadingZero = 1024

msdDimStylePropValueAngleMeasure = 1025

msdDimStylePropValueAnglePrecision = 1026

msdDimStylePropValueAngleTrailingZeros = 1027

msdDimStylePropValueDMSPrecisionMode = 1082

msdDimStylePropValueFormat = 1066

msdDimStylePropValueLabelLineFormat = 1077

msdDimStylePropValueNoReduceAltFraction = 1043

msdDimStylePropValueNoReduceAltSecFraction = 1062

msdDimStylePropValueNoReduceFraction = 1042

msdDimStylePropValueNoReduceSecFraction = 1061

msdDimStylePropValueNoReduceTolFraction = 1044

msdDimStylePropValueNoReduceTolSecFraction = 1063

msdDimStylePropValueOrdDatumValue = 1057

msdDimStylePropValueOrdDecrementReverse = 1055

msdDimStylePropValueOrdFreeLocation = 1065

msdDimStylePropValueOrdUseDatumValue = 1056

msdDimStylePropValueRoundLSD = 1028

msdDimStylePropValueSecAccuracy = 1029

msdDimStylePropValueSecFormat = 1068

msdDimStylePropValueSecShowTrailingZeros = 1033

msdDimStylePropValueSecShowZeroMasterUnit = 1035

msdDimStylePropValueSecShowZeroSubUnit = 1080

msdDimStylePropValueSecUnitMaster = 1075

msdDimStylePropValueSecUnitSub = 1076

msdDimStylePropValueShowTrailingZeros = 1039

msdDimStylePropValueShowZeroMasterUnit = 1041

msdDimStylePropValueShowZeroSubUnit = 1078

msdDimStylePropValueSuperscriptLSD = 1045

msdDimStylePropValueThousandsOpts = 1072

msdDimStylePropValueUnit = 1048

msdDimStylePropValueUnitLabelMaster = 1049

msdDimStylePropValueUnitLabelSecMaster = 1050

msdDimStylePropValueUnitLabelSecSub = 1051

msdDimStylePropValueUnitLabelSub = 1052

msdDimStylePropValueUnitMaster = 1073

msdDimStylePropValueUnitSec = 1053

msdDimStylePropValueUnitSub = 1074

msdDimStylePropValueUseWorkingUnits = 1054

MsdDimSuperScriptMode

MsdDimSuperScriptModeFromFont = 0

MsdDimSuperScriptModeGenerated = 1

MsdDimSymbolType

msdDimSymbolTypeCell = 2

msdDimSymbolTypeCharacter = 1

msdDimSymbolTypeDefault = 0

MsdDimTerminatorArrowhead

msdDimTerminatorArrowheadClosed = 1

msdDimTerminatorArrowheadFilled = 2

msdDimTerminatorArrowheadOpen = 0

MsdDimTerminatorMode

msdDimTerminatorModeAuto = 0

msdDimTerminatorModeInside = 2

msdDimTerminatorModeOutside = 3

msdDimTerminatorModeReversed = 1

MsdDimTerminatorType

msdDimTerminatorTypeArrow = 1

msdDimTerminatorTypeCircle = 3

msdDimTerminatorTypeDot = 4

msdDimTerminatorTypeNone = 0

msdDimTerminatorTypeNote = 5

msdDimTerminatorTypeOrigin = 3

msdDimTerminatorTypeStroke = 2

MsdDimTextField

msdDimTextFieldLowerLimit = 1

msdDimTextFieldMain = 0

China BDN

msdDimTextFieldMinus = 2

msdDimTextFieldPlus = 1

msdDimTextFieldUpperLimit = 0

MsdDimTextFormat

MsdDimTextFormatMU = 0

MsdDimTextFormatMU_dash_SU = 4

MsdDimTextFormatMU_Label = 1

MsdDimTextFormatMU_Label_dash_SU_Label = 6

MsdDimTextFormatMU_Label_SU_Label = 5

MsdDimTextFormatSU = 2

MsdDimTextFormatSU_Label = 3

MsdDimTextFrameType

MsdDimTextFrameTypeBox = 1

MsdDimTextFrameTypeCapsule = 2

MsdDimTextFrameTypeNone = 0

MsdDimTextJustification

msdDimTextJustificationCenter = 2

msdDimTextJustificationLeft = 1

msdDimTextJustificationRight = 3

MsdDimTextLocation

MsdDimTextLocationAbove = 1

MsdDimTextLocationInline = 0

MsdDimTextLocationOutside = 2

MsdDimTextLocationTopLeft = 3

MsdDimTextOrientation

MsdDimTextOrientationAligned = 0

MsdDimTextOrientationHorizontal = 1

MsdDimThousandsOpts

MsdDimThousandsOptsComma = 2

MsdDimThousandsOptsNone = 0

Ophia BDN

MsdDimThousandsOptsSpace = 1

MsdDimToleranceType

MsdDimToleranceTypeLimit = 1

MsdDimToleranceTypePlusMinus = 0

MsdDimType

msdDimTypeAngleAxis = 10

msdDimTypeAngleAxisX = 50

msdDimTypeAngleAxisY = 51

msdDimTypeAngleLines = 9

msdDimTypeAngleLocation = 7

msdDimTypeAngleSize = 5

msdDimTypeArcLocation = 8

msdDimTypeArcSize = 6

msdDimTypeCenter = 19

msdDimTypeCustomLinear = 15

msdDimTypeDiameter = 12

msdDimTypeDiameterExtended = 18

msdDimTypeDiameterPara = 13

msdDimTypeDiameterPerp = 14

msdDimTypeLabelLine = 52

msdDimTypeLocateSingle = 3

msdDimTypeLocateStacked = 4

msdDimTypeNone = 0

msdDimTypeNote = 53

msdDimTypeOrdinate = 16

msdDimTypeRadius = 11

msdDimTypeRadiusExtended = 17

msdDimTypeSizeArrow = 1

msdDimTypeSizeStroke = 2

msdDimTypeUseActive = -1

China BDN

MsdDimValueAngleFormat

msdDimValueAngleFormatCentesimal = 2

msdDimValueAngleFormatDegMinSec = 1

msdDimValueAngleFormatDegrees = 0

msdDimValueAngleFormatRadians = 3

MsdDimValueAnglePrecision

msdDimValueAnglePrecision1Place = 1

msdDimValueAnglePrecision2Place = 2

msdDimValueAnglePrecision3Place = 3

msdDimValueAnglePrecision4Place = 4

msdDimValueAnglePrecision5Place = 5

msdDimValueAnglePrecision6Place = 6

msdDimValueAnglePrecisionWhole = 0

MsdDimVerticalTextOptions

MsdDimVerticalTextOptionsAlways = 1

MsdDimVerticalTextOptionsNever = 0

MsdDimVerticalTextOptionsNoFit = 2

MsdDrawingMode

msdDrawingModeErase = 1

msdDrawingModeHilite = 2

msdDrawingModeNormal = 0

msdDrawingModeTemporary = 3

msdDrawingModeTemporaryErase = 4

msdDrawingModeXor = 6

MsdElementCachePurpose

msdElementCachePurposeControl = 2

msdElementCachePurposeGraphical = 4

msdElementCachePurposeNonModel = 1

MsdElementClass

msdElementClassConstruction = 2

China BDN

msdElementClassContructionRule = 6

msdElementClassDimension = 3

msdElementClassLinearPatterned = 5

msdElementClassPatternComponent = 1

msdElementClassPrimary = 0

msdElementClassPrimaryRule = 4

MsdElementSubtype

msdElementSubtypeApplicationElement = 20

msdElementSubtypeAuxiliaryCoordinateSystem = 3

msdElementSubtypeNone = -1

msdElementSubtypeUpdateSequenceElement = 33

MsdElementType

msdElementType44 = 44

msdElementTypeArc = 16

msdElementTypeBsplineBoundary = 25

msdElementTypeBsplineCurve = 27

msdElementTypeBsplineKnot = 26

msdElementTypeBsplinePole = 21

msdElementTypeBsplineSurface = 24

msdElementTypeBsplineWeight = 28

msdElementTypeCellHeader = 2

msdElementTypeCellLibraryHeader = 1

msdElementTypeComplexShape = 14

msdElementTypeComplexString = 12

msdElementTypeCone = 23

msdElementTypeConic = 13

msdElementTypeCurve = 11

msdElementTypeDesignFileHeader = 9

msdElementTypeDgnStoreComponent = 38

msdElementTypeDgnStoreHeader = 39

China BDN

msdElementTypeDigSetData = 8

msdElementTypeDimension = 33

msdElementTypeEllipse = 15

msdElementTypeGroupData = 5

msdElementTypeLevelMask = 99

msdElementTypeLevelSymbology = 10

msdElementTypeLine = 3

msdElementTypeLineString = 4

msdElementTypeMatrixDoubleData = 103

msdElementTypeMatrixHeader = 101

msdElementTypeMatrixIntegerData = 102

msdElementTypeMeshHeader = 105

msdElementTypeMicroStation = 66

msdElementTypeMultiLine = 36

msdElementTypeNamedGroupComponent = 111

msdElementTypeNamedGroupHeader = 110

msdElementTypePointString = 22

msdElementTypeRasterComponent = 88

msdElementTypeRasterFrame = 94

msdElementTypeRasterHeader = 87

msdElementTypeRasterReference = 90

msdElementTypeRasterReferenceComponent = 91

msdElementTypeReferenceAttachment = 100

msdElementTypeReferenceOverride = 108

msdElementTypeShape = 6

msdElementTypeSharedCell = 35

msdElementTypeSharedCellDefinition = 34

msdElementTypeSolid = 19

msdElementTypeSurface = 18

msdElementTypeTable = 96

China BDN

msdElementTypeTableEntry = 95

msdElementTypeTag = 37

msdElementTypeText = 17

msdElementTypeTextNode = 7

msdElementTypeView = 98

msdElementTypeViewGroup = 97

MsdError

msdAccuDrawNotEnabled = -2147218287

msdError3dReference2dMaster = -2147220767

msdErrorAccessViolation = -2147218313

msdErrorAcsNotFound = -2147220744

msdErrorAcsReplaced = -2147220745

msdErrorAddressNotKnown = -2147220784

msdErrorAddressNotValid = -2147220779

msdErrorAlreadyExists = -2147218310

msdErrorAlreadyInUse = -2147220804

msdErrorAlreadyOpen = -2147218312

msdErrorBadBSplineElement = -2147217996

msdErrorBadCharacterConstant = -2147220794

msdErrorBadContinuity = -2147217986

msdErrorBadElement = -2147218399

msdErrorBadFile = -2147218304

msdErrorBadFloat = -2147220796

msdErrorBadFormat = -2147218309

msdErrorBadHexNumber = -2147220799

msdErrorBadIndex = -2147218370

msdErrorBadKnots = -2147217991

msdErrorBadLineWeights = -2147217990

msdErrorBadModelId = -2147218334

msdErrorBadModelReference = -2147218397

China BDN

msdErrorBadName = -2147218316

msdErrorBadNumber = -2147220800

msdErrorBadOctal = -2147220797

msdErrorBadOrder = -2147217994

msdErrorBadParameter = -2147217995

msdErrorBadPeriodicity = -2147217993

msdErrorBadPoles = -2147217992

msdErrorBadRasterFormat = -2147218350

msdErrorBadResourceType = -2147220772

msdErrorBadScanList = -2147218389

msdErrorBadSpiralDefinition = -2147217989

msdErrorBadString = -2147220795

msdErrorBadType = -2147220803

msdErrorBadVersion = -2147218308

msdErrorBadWordsToFollow = -2147218311

msdErrorCacheInUse = -2147218318

msdErrorCacheLoadError = -2147218291

msdErrorCacheNotEnabled = -2147218320

msdErrorCacheNotFilled = -2147218288

msdErrorCacheNotFound = -2147218317

msdErrorCannotCreateFile = -2147218329

msdErrorCannotDereference = -2147220787

msdErrorCannotImportSeed = -2147218292

msdErrorCannotOpenFile = -2147218391

msdErrorCannotOpenSeed = -2147218303

msdErrorCannotSaveFile = -2147218328

msdErrorCellExists = -2147218372

msdErrorCellLibraryIs2d = -2147218365

msdErrorCellNotFound = -2147218373

msdErrorCellTooLarge = -2147218369

China BDN

msdErrorCircularDependency = -2147219604

msdErrorCommandReceived = -2147220704

msdErrorComplexHeaderRequired = -2147218387

msdErrorCompressionError = -2147218296

msdErrorCopyError = -2147218289

msdErrorDiskFull = -2147218395

msdErrorDivideByZero = -2147220780

msdErrorDuplicateLogical = -2147220766

msdErrorDuplicateTaskId = -2147218353

msdErrorElementFilled = -2147220756

msdErrorElementFrozen = -2147218359

msdErrorElementNotFilled = -2147220755

msdErrorElementNotFound = -2147218323

msdErrorElementNotPlanar = -2147220753

msdErrorElementTooLarge = -2147220754

msdErrorEndOfFile = -2147218390

msdErrorException = -2147219504

msdErrorFileExists = -2147218326

msdErrorFileNotFound = -2147218338

msdErrorHasChanges = -2147218298

msdErrorIdExists = -2147218321

msdErrorIdNotFound = -2147218322

msdErrorIllegalCharacter = -2147220793

msdErrorInsufficientInformation = -2147218401

msdErrorInsufficientMemory = -2147218388

msdErrorIntegralNeeded = -2147220777

msdErrorInvalidACSType = -2147220770

msdErrorInvalidButton = -2147220769

msdErrorInvalidCell = -2147218371

msdErrorInvalidClip = -2147220761

China BDN

msdErrorInvalidForFloat = -2147220781

msdErrorInvalidForFunction = -2147220771

msdErrorInvalidForStructure = -2147220782

msdErrorInvalidForType = -2147220778

msdErrorInvalidLibrary = -2147218368

msdErrorInvalidMaterOrigin = -2147220763

msdErrorInvalidOperationForNested = -2147218302

msdErrorInvalidOperationForNonNested = -2147218301

msdErrorInvalidPatternSpace = -2147220760

msdErrorInvalidReference = -2147220762

msdErrorInvalidReferenceOrigin = -2147220764

msdErrorInvalidSymbol = -2147220789

msdErrorLinkageNotFound = -2147218344

msdErrorLoadingInterface = -2147218297

msdErrorModelerNotLoaded = -2147219703

msdErrorModelIdExists = -2147218332

msdErrorModelNameExists = -2147218333

msdErrorModelNotEmpty = -2147218342

msdErrorModifyComplex = -2147218392

msdErrorNameNotUnique = -2147218343

msdErrorNameTooLong = -2147218335

msdErrorNeedExponent = -2147220798

msdErrorNeedInteger = -2147220802

msdErrorNoAcsDefined = -2147220746

msdErrorNoBounds = -2147218001

msdErrorNoBSplineHeader = -2147217999

msdErrorNoCellLibrary = -2147218375

msdErrorNoClipVolume = -2147218336

msdErrorNoFence = -2147218337

msdErrorNoGraphicGroup = -2147220742

msdErrorNoKnots = -2147218003

msdErrorNoLevelMask = -2147220740

msdErrorNoLineWeights = -2147218002

msdErrorNoMatch = -2147218381

msdErrorNoModel = -2147218299

msdErrorNoModelInformation = -2147218331

msdErrorNonClosedElement = -2147220757

msdErrorNonClosedPatternElement = -2147220759

msdErrorNonCoplanarShapes = -2147220750

msdErrorNonSolidPatternElement = -2147220758

msdErrorNoNumberBounds = -2147218000

msdErrorNoOffsetIntersection = -2147217987

msdErrorNoParentModel = -2147218330

msdErrorNoPoles = -2147218004

msdErrorNoReferenceSlots = -2147220747

msdErrorNoselectionSet = -2147220748

msdErrorNoSuchModel = -2147218294

msdErrorNoSymbol = -2147220791

msdErrorNotDesignFIle = -2147220768

msdErrorNotDirectAttachment = -2147220739

msdErrorNotFunction = -2147220775

msdErrorNotLoaded = -2147218300

msdErrorNotLocked = -2147218293

msdErrorNotMember = -2147220785

msdErrorNotOpen = -2147218315

msdErrorNotSingleView = -2147220765

msdErrorNotStructure = -2147220786

msdErrorNotSupported = -2147218348

msdErrorNotValidExpression = -2147220776

msdErrorNullSolution = -2147220752

CSDN @BDN

msdErrorOldMaterialTable = -2147220749

msdErrorOperationCanceled = -2147218306

msdErrorParasolidError = -2147219703

msdErrorReadOnly = -2147218396

msdErrorRecurseLimit = -2147217985

msdErrorRenameError = -2147218290

msdErrorRequires3dFile = -2147218400

msdErrorResourceNotFound = -2147218376

msdErrorSharingViolation = -2147218314

msdErrorStructureNeeded = -2147220801

msdErrorSymbolNotResolved = -2147219704

msdErrorSyntaxError = -2147220790

msdErrorSystemError = -2147218363

msdErrorTagBadAssociation = -2147220096

msdErrorTagBadReportFile = -2147220098

msdErrorTagBadReportKeyword = -2147220097

msdErrorTagNameTooLong = -2147220092

msdErrorTagNoTarget = -2147220095

msdErrorTagNotFound = -2147220093

msdErrorTagNotInSet = -2147220101

msdErrorTagPreviouslyDefined = -2147220094

msdErrorTagSetNameLong = -2147220103

msdErrorTagSetNotFound = -2147220102

msdErrorTagSetPreviouslyDefined = -2147220100

msdErrorTagSetTooBig = -2147220099

msdErrorTagUndefinedType = -2147220104

msdErrorTimeout = -2147218362

msdErrorTooComplex = -2147220783

msdErrorTooFewArguments = -2147220773

msdErrorTooFewPoles = -2147217998

msdErrorTooManyArguments = -2147220774

msdErrorTooManyKnots = -2147217988

msdErrorTooManyOpenFiles = -2147218307

msdErrorTooManyPoles = -2147217997

msdErrorTooManySurfaceElements = -2147218346

msdErrorTypesIncompatible = -2147220788

msdErrorUnboundedSolution = -2147220751

msdErrorUnknownError = -2147218305

msdErrorUnknownFormat = -2147218295

msdErrorUnsupported = -2147220792

msdErrorUserCanceledAction = -2147218382

msdErrorV7CellLibrary = -2147218327

msdErrorViewGroupNotFound = -2147220741

msdErrorViewNotDisplayed = -2147218374

msdErrorViewNotFound = -2147220743

msdErrorWriteFailed = -2147218393

msdErrorWriteInhibited = -2147218394

msdErrorWrongElementID = -2147218319

MsdFileAccessMode

msdFileAccessModeRead = 1

msdFileAccessModeReadWrite = 3

MsdFillMode

msdFillModeFilled = 1

msdFillModeNotFilled = 0

msdFillModeOutlined = 2

msdFillModeUseActive = -1

MsdFontType

msdFontTypeMicroStation = 0

msdFontTypeSHX = 1

msdFontTypeUnknown = 3

msdFontTypeWindowsTrueType = 2

MsdGeoReferenceSisterFileType

msdGeoReferenceSisterFileTypeHgr = 1

msdGeoReferenceSisterFileTypeNone = 0

msdGeoReferenceSisterFileTypeTwf = 2

MsdGlobalLineStyleScale

msdGlobalLineStyleScaleBoth = 3

msdGlobalLineStyleScaleMaster = 0

msdGlobalLineStyleScaleNone = 1

msdGlobalLineStyleScaleReference = 2

MsdLevelChangeType

msdLevelChangeAfterChangeActive = 9

msdLevelChangeAfterCreate = 2

msdLevelChangeAfterDelete = 3

msdLevelChangeBeforeChangeActive = 17

msdLevelChangeBeforeDelete = 18

msdLevelChangeChangeAttribute = 8

msdLevelChangeChangeCode = 5

msdLevelChangeChangeDisplay = 7

msdLevelChangeChangeName = 4

msdLevelChangeChangeParent = 6

msdLevelChangeTableRedo = 15

msdLevelChangeTableUndo = 14

MsdLevelElementAccess

msdLevelElementAccessAll = 0

msdLevelElementAccessLocked = 1

msdLevelElementAccessReadOnly = 2

msdLevelElementAccessViewOnly = 3

MsdLimits

msdLimitsMaxVertices = 5000

China BDN

msdLimitsMaxViews = 8

msdMeasurementBase

msdMeasurementBaseDegree = 2

msdMeasurementBaseMeter = 1

msdMeasurementBaseNone = 0

MsdMeasurementSystem

msdMeasurementSystemEnglish = 1

msdMeasurementSystemMetric = 2

msdMeasurementSystemUndefined = 0

MsdMemberTraverseType

msdMemberTraverseCopy = 2

msdMemberTraverseDirectMembers = 4

msdMemberTraverseEnumerate = 3

msdMemberTraverseManipulate = 1

msdMemberTraverseSimple = 0

MsdMessageCenterPriority

msdMessageCenterPriorityDebug = 13

msdMessageCenterPriorityError = 10

msdMessageCenterPriorityInfo = 12

msdMessageCenterPriorityNone = 14

msdMessageCenterPriorityWarning = 11

MdlModelChangeType

mdlModelChangeActive = 5

mdlModelChangeBeforeActive = 11

mdlModelChangeBeforeCreate = 15

mdlModelChangeBeforeDelete = 6

mdlModelChangeBeforeName = 12

mdlModelChangeBeforeProperties = 14

mdlModelChangeBeforeSettings = 13

mdlModelChangeBeforeUnCreate = 9

China BDN

mdlModelChangeBeforeUnDelete = 16

mdlModelChangeCreate = 1

mdlModelChangeDelete = 2

mdlModelChangeName = 10

mdlModelChangePropagateAnnotationScale = 17

mdlModelChangeProperties = 3

mdlModelChangeSettings = 4

mdlModelChangeUnCreate = 7

mdlModelChangeUnDelete = 8

MsdModelType

msdModelTypeDefault = -1

msdModelTypeExtraction = 2

msdModelTypeNormal = 0

msdModelTypeSheet = 1

MsdNestOverrides

msdNestOverridesAlways = 1

msdNestOverridesAsRequired = 0

msdNestOverridesNever = 2

MsdNewLevelDisplay

msdNewLevelDisplayAlways = 1

msdNewLevelDisplayFromConfig = 0

msdNewLevelDisplayNever = 2

MsdRasterBlockType

msdRasterBlockTypeImage = 4

msdRasterBlockTypeLine = 1

msdRasterBlockTypeStrip = 3

msdRasterBlockTypeTile = 2

MsdRasterDisplayOrderCommand

msdRasterDisplayOrderCommandBackward = 3

msdRasterDisplayOrderCommandForward = 2

China BDN

msdRasterDisplayOrderCommandToBack = 1

msdRasterDisplayOrderCommandToFront = 0

MsdRasterDisplayPriorityPlane

msdRasterDisplayPriorityPlaneBack = 1

msdRasterDisplayPriorityPlaneFront = 3

msdRasterDisplayPriorityPlaneVector = 2

MsdRasterModificationType

msdRasterModificationType_ClipBoundary = 5

msdRasterModificationType_ClipMask = 4

msdRasterModificationType_ExtendedInformation = 0

msdRasterModificationType_GeoReferenceInformation = 1

msdRasterModificationType_RasterInformation = 3

msdRasterModificationType_Reload = 6

msdRasterModificationType_RenderingInformation = 2

MsdRasterWorldFile

msdRasterWorldFileHgr = 1

msdRasterWorldFileNone = 0

msdRasterWorldFileWorldFile = 2

MsdReferenceSystem

msdReferenceSystemDgn = 2

msdReferenceSystemRaster = 1

MsdRenderingMode

msdRenderingModeConstantShade = 5

msdRenderingModeCrossSection = 1

msdRenderingModeHiddenLine = 3

msdRenderingModeParticleTrace = 11

msdRenderingModePhong = 7

msdRenderingModeRadiosity = 10

msdRenderingModeRayTrace = 8

msdRenderingModeRenderWireFrame = 9

China BDN

msdRenderingModeSmoothShade = 6

msdRenderingModeSolidFill = 4

msdRenderingModeWireFrame = 0

msdRenderingModeWireMesh = 2

MsdStandardsCheckerReplace

msdStandardsCheckerReplaceChoiceAbort = 4

msdStandardsCheckerReplaceChoiceFix = 1

msdStandardsCheckerReplaceChoiceMarkIgnored = 2

msdStandardsCheckerReplaceChoiceMarkNotIgnored = 3

msdStandardsCheckerReplaceChoiceSkip = 0

MsdStandardsCheckerReplaceOptions

msdStandardsCheckerReplaceOptionCanFix = 2

msdStandardsCheckerReplaceOptionCanIgnore = 1

MsdStatusBarArea

msdStatusBarAreaLeft = 16

msdStatusBarAreaMiddle = 15

MsdTagType

msdTagTypeBinary = 5

msdTagTypeCharacter = 1

msdTagTypeDouble = 4

msdTagTypeLongInteger = 3

msdTagTypeShortInteger = 2

MsdTangentElementOutputType

msdTangentArcs = 1

msdTangentCircles = 0

msdTangentTriangles = 2

MsdTangentInterpolationType

msdTangentFromCircleFit = 1

msdTangentFromCubicFit = 2

msdTangentFromCurve = 0

China BDN

MsdTextDirection

msdTextDirectionHorizontal = 0

msdTextDirectionRightToLeft = 8

msdTextDirectionVertical = 4

msdTextDirectionVerticalMultiLineRightToLeft = 2

MsdTextJustification

msdTextJustificationCenterBottom = 8

msdTextJustificationCenterCenter = 7

msdTextJustificationCenterTop = 6

msdTextJustificationLeftBottom = 2

msdTextJustificationLeftCenter = 1

msdTextJustificationLeftTop = 0

msdTextJustificationRightBottom = 14

msdTextJustificationRightCenter = 13

msdTextJustificationRightTop = 12

MsdTextNodeLineSpacingType

msdTextNodeLineSpacingTypeAtLeast = 3

msdTextNodeLineSpacingTypeAutomatic = 1

msdTextNodeLineSpacingTypeExact = 0

msdTextNodeLineSpacingTypeExactFromLineTop = 2

MsdV7Action

msdV7ActionAskUser = 0

msdV7ActionUpgradeToV8 = 1

msdV7ActionWorkmode = 3

MsdView

msdView1 = 1

msdView2 = 2

msdView3 = 4

msdView4 = 8

msdView5 = 16

China BDN

msdView6 = 32

msdView7 = 64

msdView8 = 128

msdViewAll = 255

msdViewNone = 0

MsdXDatumType

msdXDatumTypeBinaryData = 1004

msdXDatumTypeControlString = 1002

msdXDatumTypeDatabaseHandle = 1005

msdXDatumTypeDistance = 1041

msdXDatumTypeInt16 = 1070

msdXDatumTypeInt32 = 1071

msdXDatumTypeLevel = 1003

msdXDatumTypePoint = 1010

msdXDatumTypeReal = 1040

msdXDatumTypeScaleFactor = 1042

msdXDatumTypeString = 1000

msdXDatumTypeUnsupported = 0

msdXDatumTypeWorldDirection = 1013

msdXDatumTypeWorldSpaceDisplacement = 1012

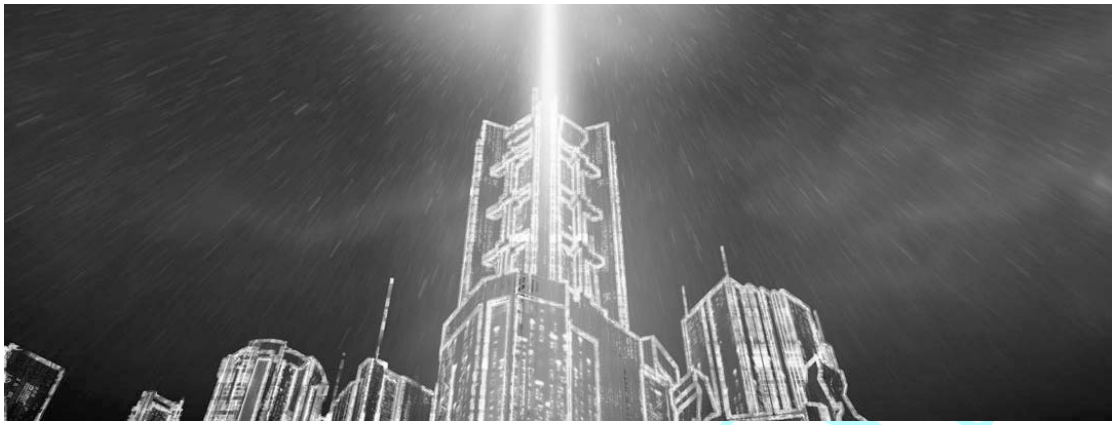
msdXDatumTypeWorldSpacePosition = 1011

本章回顾

随着继续学习本书，我们将会在代码样例中看到使用枚举的例子。

正如上一章指出的那样，对象浏览器在查找和确定如何使用枚举上也很有帮助。

China BDN



13. MicroStation 对象模型—类型

到目前为止，我们已经介绍并讨论了变量、对象、属性和方法这些概念，现在将要讨论类型。

类型用起来像变量但看起来像对象，因为它保存多个分量元素。演示类型的最佳途径就是看一个在 MicroStation VBA 编程中普遍使用的类型。

```
Type Point3d
    X As Double
    Y As Double
    Z As Double
End Type
```

Point3d 类型有三个成员：X(双精度)、Y(双精度)和 Z(双精度)。

```
Sub TestPoint3d()
    Dim StartPoint As Point3d
    Dim EndPoint As Point3d
    Dim MyLine As LineElement
    StartPoint.X = 1.5
    StartPoint.Y = 2.5
    StartPoint.Z = 3.5
    EndPoint.X = 4
    EndPoint.Y = 0
    EndPoint.Z = 0
```

```
    Set MyLine = CreateLineElement2(Nothing, StartPoint, EndPoint)
    ActiveModelReference.AddElement MyLine
End Sub
```

我们声明了两个具有类型“Point3d”的变量，给这些变量的 X、Y 和 Z 分量赋了坐标值，然后把它们用于 CreateLineElement2 方法。下面是“CreateLineElement2”的声明：

```
Sub CreateLineElement2(Template As Element, StartPoint As Point3d, _
EndPoint As Point3d) as LineElement
```

注意该方法是如何要求两个 Point3d 类型的：一个是起点，另一个是终点。

下面是在 MicroStation VBA 中可用的类型列表：

```
Type MsdACSType
    Justification As MsdDataEntryRegionJustification
    Length As Long
    StartPosition As Long
End Type
Type MsdACSType
    High As Long
    Low As Long
End Type
Type MsdAddAttachmentFlags
    Center As Point3d
    Start As Double
    Sweep As Double
    Vector0 As Point3d
    Vector90 As Point3d
End Type
Type MsdAngleAccuracy
    Du As Point3d
    Dv As Point3d
End Type
```


China BDN

```
Type MsdAngleAccuracy
    RowX As Point3d
    RowY As Point3d
    RowZ As Point3d
End Type
Type MsdAngleAccuracy
    Base As MsdMeasurementBase
    Label As String
    System As MsdMeasurementSystem
    UnitsPerBaseDenominator As Double
    UnitsPerBaseNumerator As Double
End Type
Type MsdAngleFormat
    Details As String
    Msg As String
    Priority As MsdMessageCenterPriority
End Type
Type MsdAngleMode
    Normal As Point3d
    Origin As Point3d
End Type
Type MsdAttachMode
    X As Double
    Y As Double
End Type
Type MsdBsplineCurveOffsetCuspType
    X As Double
    Y As Double
    Z As Double
End Type
```

China BDN

```
Type MsdBsplineCurveOffsetCuspType
    High As Point3d
    Low As Point3d
End Type
Type MsdBsplineCurveType
    Direction As Point3d
    Origin As Point3d
End Type
Type MsdBsplineCurveType
    Duu As Point3d
    Duv As Point3d
    Dvu As Point3d
    Dvv As Point3d
End Type
Type MsdBsplineCurveType
    EndPoint As Point3d
    StartPoint As Point3d
End Type
Type MsdBsplineParametrizationType
    RowX As Point3d
    RowY As Point3d
    RowZ As Point3d
    TranslationX As Double
    TranslationY As Double
    TranslationZ As Double
End Type
Type MsdBsplineSurfaceType
    X As Double
    Y As Double
    Z As Double
```

China BDN

```
End Type
Type MsdBsplineSurfaceType
    Type As MsdXDatumType
    Value As Variant
End Type
```

以上列出的每种类型都能用于 MicroStation VBA。“Type”和“End Type”声明是标准 VBA 的约定。事实上，我们在 VBA 中还能建立自己的“类型”。自定义类型在代码模块的通用声明区来声明。例如，如果想要一个叫做“Point4d”的新类型，可以使用代码：

```
Type Point4d
    X As Double
    Y As Double
    Z As Double
    A As Double
End Type
```

作出这个声明后就可如下所示声明变量了：

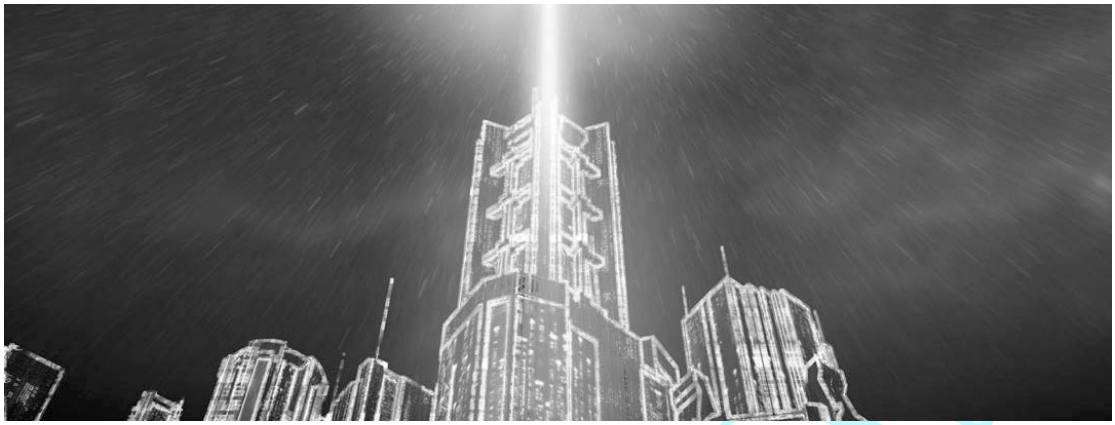
```
Dim MyPoint As Point4d
```

就像枚举一样，类型也将在我们继续学习 MicroStation VBA 中广泛使用。

本章回顾

类型类似于对象。对象具有属性，类型具有类似于属性的成员。在 MicroStation 中用的最普遍的类型是 Point3d 类型，它有 X、Y 和 Z 成员，每个成员都被声明为双精度。

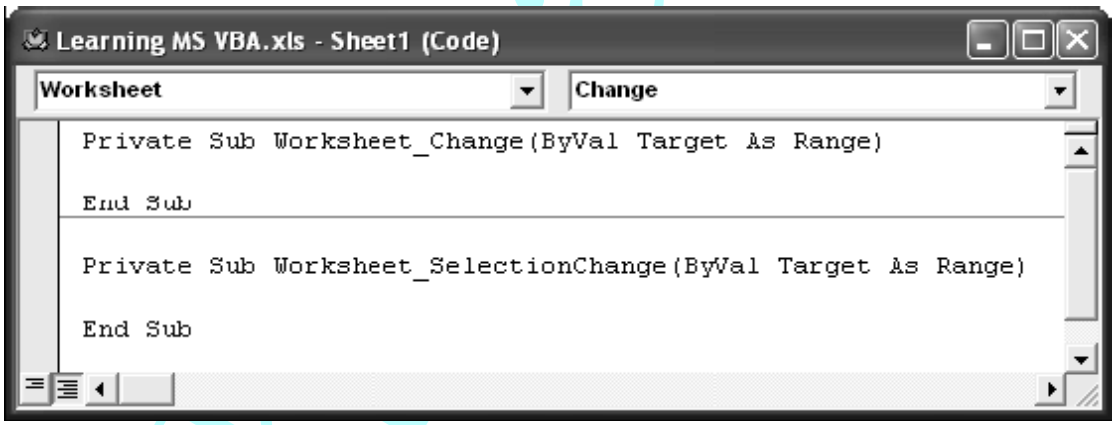
China BDN



14. MicroStation 对象模型—事件

我们已经讨论过，对象具有属性、方法和事件。在讨论创建可视化界面时介绍过事件。当用户点击一个命令按钮时，该按钮的 Click(点击)事件被触发。每当用户与 MicroStation 的各个方面交互时都会触发 MicroStation 事件。

当一家公司(如奔特力)把 VBA 嵌入其应用(目前情况下是指 MicroStation)时，如何处理事件的问题就被提了出来。下面是 Microsoft Excel 处理事件的方式：



Excel 工作簿中的每一个工作表都有自动公开的事件，其中两个是 Change 和 SelectionChange 事件。当工作表单元格的值改变时和当用户从一个单元移动到另一单元时触发这两个事件。

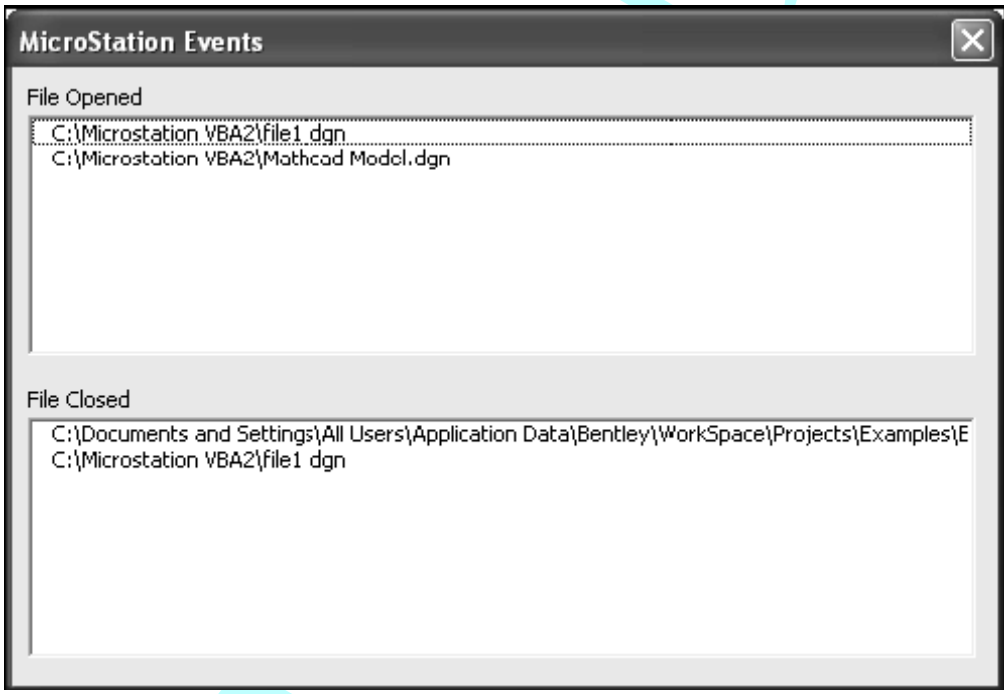
有两种方式来捕获和使用 MicroStation 事件。一种是在类模块或窗体中声明一个 Application 变量并使用“WithEvents”关键字，这种方式下有两个事件可用：OnDesignFileOpened 和 OnDesignFileClosed。MicroStation 中的大多数事件都是通过使用接口来访问的。

OnDesignFileOpened 和 OnDesignFileClosed

这里有个小例子来说明 OnDesignFileOpened 和 OnDesignFileClosed 事件是如何工作的。

我们将使用一个非模态的用户窗体。非模态的意思是：即使窗体显示着，用户仍然能与 MicroStation 交互。窗体初始化时，我们把 MicroStation 应用对象设置给一个声明为“WithEvents”的变量，声明“WithEvents”变量要在用户窗体的通用声明区内完成。声明一个“WithEvents”变量后，属于我们所指定的对象的事件就能在代码中使用。

下面是几个文件被打开(当新文件打开时前一个文件关闭)后程序运行的情况：



每当打开或关闭一个文件时，就会触发相关的事件。我们先从用户窗体代码区中的代码看起。

```
Dim WithEvents MyApp As MicroStationDGN.Application
Private Sub UserForm_Initialize()
    Set MyApp = Application
End Sub
Private Sub MyApp_OnDesignFileOpened(ByVal _
            DesignFileName As String)
```

```
        lstOpened.AddItem DesignFileName
    End Sub
    Private Sub MyApp_OnDesignFileClosed(ByVal _
                DesignFileName As String)
        lstClosed.AddItem DesignFileName
    End Sub
```

每当触发 OnDesignFileOpened 事件时，将 DesignFileName 参数添加到 **lstOpened** 列表框中；当关闭一个文件时，将 DesignFileName 添加到 **lstClosed** 列表框中。

我们想以非模态方式显示窗体，所以要运行下面的宏：

```
Sub ShowEvents()
    frmEvents.Show vbModeless
End Sub
```

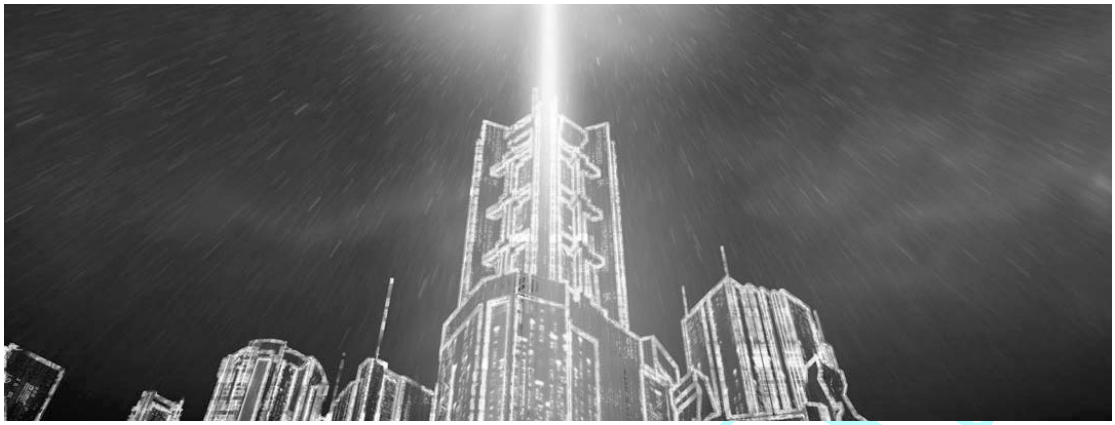
ShowEvents 过程放在代码模块中。

能用 OnDesignFileOpened 和 OnDesignFileClosed 事件来记录哪些文件被打开过，事件中以文件名为参数。还能对这一基本功能进行扩展，使其包含捕获当前日期/时间(用 Now 函数)和当前用户(用 Application.UserName 属性)的功能。

本章回顾

用户与软件交互时会触发事件。MicroStation 事件基本上都是通过使用接口(后续章节会讨论到)来公开的。OnDesignFileOpened 和 OnDesignFileClosed 事件通过在没模块或用户窗体中声明 MicroStation.Application 对象“WithEvents”来公开。关于使用“WithEvents”的更多信息能在标准的 VBA 帮助文件中找到。

China BDN



15. 向文档中添加内容

在介绍编程主题时已经建立过线、圆、弧和文本。现在让我们分析向设计文件中添加元素和其它对象的细节内容。先以图形元素开始，然后再到非图形元素，如：层。

本章会讲到：

- 图形元素
- 建立新文档
- 建立数据的安全性问题

图形元素

把元素添加到设计文件中需要两步：第一步是在内存中建立元素；第二步是把元素添加到设计文件中。正如您将要看到的那样，往往能有多种方式建立元素。对每种建立方法都将演示多个例子。

线

“两点之间最短距离就是一条直线”。果真如此的话，应该能够通过两点来建立一条线，对吗？对，这是建立线的一种方式，还可以通过一个顶点数组绘制多条线。

- Function CreateLineElement1(Template As Element,
  Vertices() As Point3d) As LineElement
- Function CreateLineElement2(Template As Element,
  StartPoint As Point3d, EndPoint As Point3d) As LineElement

```
Sub TestCreateLineA()
    Dim StPt As Point3d
    Dim EnPt As Point3d
    Dim myLine As LineElement
    EnPt.X = 4:    EnPt.Y = 6:    EnPt.Z = 8
    Set myLine = CreateLineElement2(Nothing, StPt, EnPt)
    ActiveModelReference.AddElement   myLine
End Sub
```

TestCreateLineA 用 CreateLineElement2 方法建立了一个新的线元素，是使用起点和终点方式建立的。

```
Sub TestCreateLineB()
    Dim StPt As Point3d
    Dim EnPt As Point3d
    Dim myLine As LineElement
    'Line 1
    StPt.X = 0: StPt.Y = 0: StPt.Z = 0
    EnPt.X = 4: EnPt.Y = 0: EnPt.Z = 0
    Set myLine = CreateLineElement2(Nothing, StPt, EnPt)
    ActiveModelReference.AddElement myLine
    'Line 2
    StPt.X = 4: StPt.Y = 0: StPt.Z = 0
    EnPt.X = 4: EnPt.Y = 4: EnPt.Z = 0
    Set myLine = CreateLineElement2(Nothing, StPt, EnPt)
    ActiveModelReference.AddElement myLine
    'Line 3
    StPt.X = 4: StPt.Y = 4: StPt.Z = 0
    EnPt.X = 0: EnPt.Y = 4: EnPt.Z = 0
    Set myLine = CreateLineElement2(Nothing, StPt, EnPt)
    ActiveModelReference.AddElement myLine
    'Line 4
```

```
    StPt.X = 0: StPt.Y = 4: StPt.Z = 0
    EnPt.X = 0: EnPt.Y = 0: EnPt.Z = 0
    Set myLine = CreateLineElement2(Nothing, StPt, EnPt)
    ActiveModelReference.AddElement myLine
End Sub
```

TestCreateLineB 使用 CreateLineElement2 函数建立并添加了四条线。当分析建立线所用的坐标时会发现，我们正在画一个正方形。让我们用 CreateLineElement1 方法来建立同样的正方形。

```
Sub TestCreateLineC()
    Dim LinePoints(0 To 4) As Point3d
    Dim myLine As LineElement
    LinePoints(0).X = 0: LinePoints(0).Y = 0
    LinePoints(1).X = 4: LinePoints(1).Y = 0
    LinePoints(2).X = 4: LinePoints(2).Y = 4
    LinePoints(3).X = 0: LinePoints(3).Y = 4
    LinePoints(4).X = 0: LinePoints(4).Y = 0
    Set myLine = CreateLineElement1(Nothing, LinePoints)
    ActiveModelReference.AddElement myLine
End Sub
```

正如在 TestCreateLineC 中看到的那样，可以提供一个 Point3d 类型的数组用单个 CreateLineElement1 方法就能建立四条线。

> **注意：**当声明一个数字变量时，一个零值(0)自动赋给该变量。知道这一点后就可以不去理睬每个点的.Z 分量，该分量缺省情况下将被赋予零值。也可以不去考虑要赋为零值的.X 和.Y 分量，但考虑它们使得代码更容易读懂。还要注意的是：使用冒号(:)把两行代码放到了一行上。把同一顶点的赋值放在一行上也增加了代码的易读性。

下个例子将建立一个过程，该过程允许用户指定建立三维线的 x、y、z 分量。我们想提供任意个 x、y、z 分量集，所以在参数声明中使用了 ParamArray 关键字。

```
Sub Create3dLines(ParamArray PointElems() As Variant)
    If (UBound(PointElems) + 1) Mod 3 <> 0 Then
        MsgBox "Invalid number of point elements.", vbCritical
        Exit Sub
    End If
    If UBound(PointElems) + 1 < 5 Then
        MsgBox "A minimum of 2 X, Y, Z points must be provided.", vbCritical
        Exit Sub
    End If
    Dim LinePoints() As Point3d
    ReDim LinePoints(0 To (UBound(PointElems) + 1) \ 3) As Point3d
    Dim I As Long
    Dim PointCounter As Long
    Dim myLine As LineElement
    For I = LBound(PointElems) To UBound(PointElems) Step 3
        LinePoints(PointCounter).X = PointElems(I)
        LinePoints(PointCounter).Y = PointElems(I + 1)
        LinePoints(PointCounter).Z = PointElems(I + 2)
        PointCounter = PointCounter + 1
    Next I
    Set myLine = CreateLineElement1(Nothing, LinePoints)
    ActiveModelReference.AddElement    myLine
End Sub
```

这个过程很直接，但比过去的那些过程稍微复杂些。让我们从头至尾分析一下吧。

1、过程名是 Create3dLines，声明了单个 ParamArray 参数；

> **注意：**在函数或过程中只能声明一个 ParamArray 参数且它必须是最后一个参数。

2、3D 点由 x、y、z 分量组成，缘于此，就需要保证 PointElems 数组以 3

个为组。假如参数的上界(upper-bound)值为 4，这就是说用户提供了 5 个分量元素，因为 5 个分量不能产生两个完整的 3D 点而会出现问题；

3、通过了“以 3 个为组”的测试后，还需要看是否至少提供了两个点，毕竟不能从一个点来建立线。能有多种不同的方式来实现这一点，一种方式就是查看 PointElems 数组的上界(用 Ubound 函数)。如果上界小于 5，就知道在数组中没有足够的分量元素；如果等于 5，就知道具有两个完整的 3D 点；

4、由于每次使用该过程时点的个数可能都不相同，所以需要建立一个存储点的动态数组。然后基于所提供的 PointElems 个数设置数组中点的个数；

5、现在需要基于 ParamArray 中提供的元素赋值点的 X、Y 和 Z 分量；

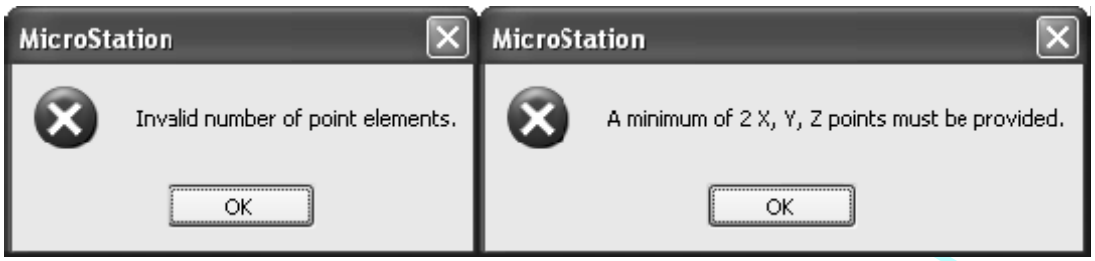
6、利用由 ParamArray 建立而来的点，用 CreateLineElement1 建立线；

7、把线添加到 ActiveModelReference 中。

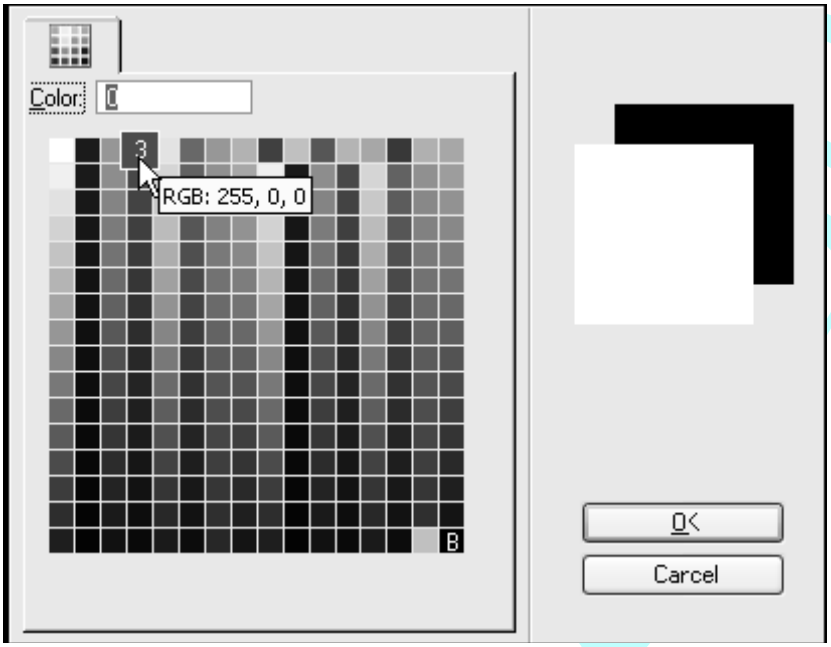
因为该过程用到了参数，所以不能直接运行。下面是运行“Create3dLines”过程的一个测试过程。

```
Sub TestCreate3dLines()
    Create3dLines 0, 0, 0, 4, 0, 0, 4, 4, 0, 0, 4, 0, 0, 0, 0
    Create3dLines 0, 0, 0, 4, 4, 0
    Create3dLines 0, 4, 0, 4, 0, 0
    Create3dLines 0, 4, 0, 4, 0
    Create3dLines 0, 4, 0
End Sub
```

测试过程 TestCreate3dLines 对新建过程 Create3dLines 调用了五次：第一次是绘制了一个正方形，从(0,0,0)到(4,0,0)到(4,4,0)到(0,4,0)再回到(0,0,0)；第二次绘制一条从(0,0,0)到(4,4,0)的线；第三次绘制一条从(0,4,0)到(4,0,0)的线；最后两次调用是用来测试 ParamArray 有效性的代码。我们不能从(0,4,0)到(4,0)画线，因为第二点只给了两个分量(x 和 y)而每个点需要三个分量。最后一次调用试图画一条从(0,4,0)到一个未知点的线，不能绘制仅有一个点的线。下面两个消息框是按出现顺序排列的。



一旦线被建立后，就能对其属性进行改变，如颜色、层或线型属性。



在 MicroStation 颜色表对话框中，当光标在表中的颜色上滚过时能看到每个颜色的颜色号和 RGB 值。从所示的右图中能看到颜色号 3 具有 RGB 值(255,0,0)。

让我们绘制几条线并把它们的颜色改成红色(255,0,0)。

```
Sub TestCreateLineD()
    Dim LinePoints(0 To 1) As Point3d
    Dim myLine As LineElement
    LinePoints(0).X = 0: LinePoints(0).Y = 0
    LinePoints(1).X = 4: LinePoints(1).Y = 4
    Set myLine = CreateLineElement1(Nothing, LinePoints)
    myLine.Color = 3
    ActiveModelReference.AddElement myLine
    LinePoints(0).X = 0: LinePoints(0).Y = 4
    LinePoints(1).X = 4: LinePoints(1).Y = 0
    Set myLine = CreateLineElement1(Nothing, LinePoints)
    myLine.Color = 3
```

```
    ActiveModelReference.AddElement myLine
End Sub
```

两条颜色属性改为颜色号 3(红)的线被建立。

下面是完成相同任务的另一种方式：

```
Sub TestCreateLineE()
    Dim LinePoints(0 To 1) As Point3d
    Dim myLine As LineElement
    Dim myLine2 As LineElement
    LinePoints(0).X = 0: LinePoints(0).Y = 0
    LinePoints(1).X = 4: LinePoints(1).Y = 4
    Set myLine = CreateLineElement1(Nothing, LinePoints)
    myLine.Color = 3
    ActiveModelReference.AddElement myLine
    LinePoints(0).X = 0: LinePoints(0).Y = 4
    LinePoints(1).X = 4: LinePoints(1).Y = 0
    Set myLine2 = CreateLineElement1(myLine, LinePoints)
    ActiveModelReference.AddElement myLine2
End Sub
```

本例中添加了一行代码，删去了一行代码，对另一行作了小改动。下面是改动的行：

```
Set myLine2 = CreateLineElement1(myLine, LinePoints)
```

以前用 CreateLineElement1 时在模板参数处给的是“Nothing”值，本例中此处提供了刚建立的第一条线作为变量。这样会使得新建立的线具有与“模板”元素相同的非几何属性。

建立形

形是由一系列线连接而成的一个元素。

下面是 CreateShapeElement1 的声明：

- Function CreateShapeElement1(Template As Element,
  Vertices() As Point3d, [FillMode As MsdFillMode =

```
    msdFillModeUseActive]) As ShapeElement
```

下面是一个建立三角形的过程。

```
Sub TestCreateShapeA()
    Dim myShape As ShapeElement
    Dim ShapePoints(0 To 2) As Point3d
    ShapePoints(0).X = 0: ShapePoints(0).Y = 0
    ShapePoints(1).X = 2: ShapePoints(1).Y = 0
    ShapePoints(2).X = 1: ShapePoints(2).Y = 1
    Set myShape = CreateShapeElement1(Nothing, ShapePoints)
    ActiveModelReference.AddElement myShape
End Sub
```

该代码运行后，建立了一个三角形并加入到 ActiveModelReference 中。注意，不需要提供第四个点(0,0,0)来封闭三角形。多边形总是封闭的。

比较一下 CreateShapeElement1 的声明和使用不难发现，程序中没有使用可选的 FillMode(填充方式)参数。缺省情况下，FillMode 参数使用 MicroStation 中的当前设置。让我们复制并粘贴 TestCreateShapeA，把新过程改名为 TestCreateShapeB 并给以 FillMode 参数：

```
Sub TestCreateShapeB()
    Dim myShape As ShapeElement
    Dim ShapePoints(0 To 2) As Point3d
    ShapePoints(0).X = 0: ShapePoints(0).Y = 0
    ShapePoints(1).X = 2: ShapePoints(1).Y = 0
    ShapePoints(2).X = 1: ShapePoints(2).Y = 1
    Set myShape = CreateShapeElement1(Nothing, ShapePoints, _
                                      msdFillModeFilled)
    ActiveModelReference.AddElement myShape
End Sub
```

TestCreateShapeB 建立一个填充的三角形。如果产生的三角形看起来没有填充，那可能是因为没有选择视图属性中的填充设置(**Settings > View**

Attributes)。

让我们进一步学习建立形的知识。现在要建立一个函数，它基于中心点、边数和半径建立一个正多边形，该正多边形与我们所给半径构成的圆内接。

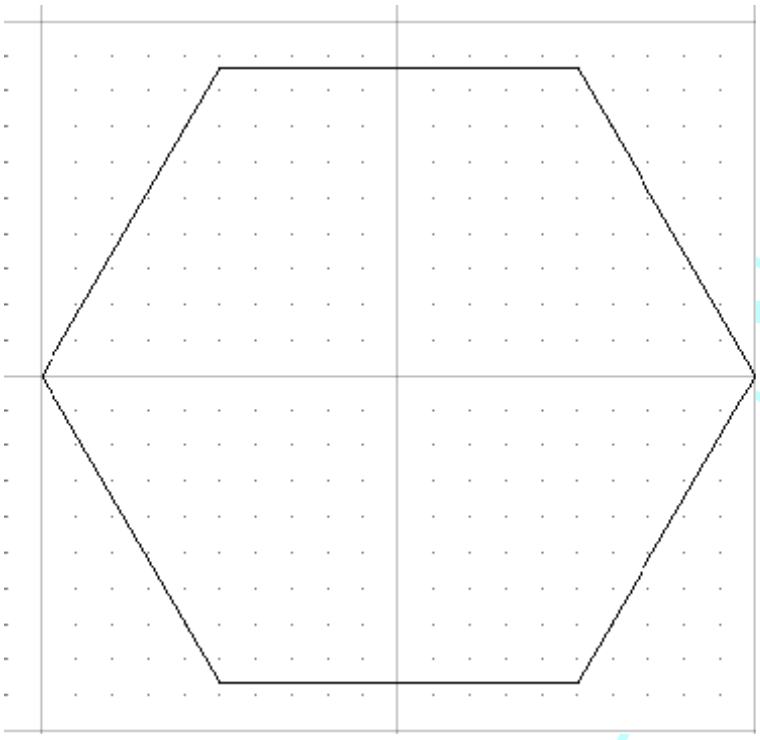
```
Function CreatePolygon(CenterPoint As Point3d, _
                NumOfSides As Long, Radius As Double) As ShapeElement
    Dim myShape As ShapeElement
    Dim ShapePoints() As Point3d
    ReDim ShapePoints(0 To NumOfSides - 1) As Point3d
    Dim PointIndex As Long
    Dim IncAngle As Double
    IncAngle = 360 / NumOfSides
    For PointIndex = LBound(ShapePoints) To UBound(ShapePoints)
        ShapePoints(PointIndex) = _
            Point3dAddAngleDistance(CenterPoint, _
                Radians(IncAngle * PointIndex), Radius, 0)
    Next
    Set CreatePolygon = CreateShapeElement1(Nothing, ShapePoints)
End Function
```

这就是我们的函数，它返回一个 ShapeElement。因为使用了参数，所以需要建立一个测试过程来运行它。

```
Sub TestCreatePolygon()
    Dim CPoint As Point3d
    Dim myShape As ShapeElement
    Set myShape = CreatePolygon(CPoint, 6, 1)
    ActiveModelReference.AddElement    myShape
End Sub
```

TestCreatePolygon 过程声明了一个 Point3d 变量，因为没有对该点的 X、Y 或 Z 分量作任何修改，所以正多边形以(0,0,0)点为中心建立。

China BDN



建立圆

圆由一个中心点和一个半径或直径来定义。在 MicroStation VBA 中用 CreateEllipseElement1 和 CreateEllipseElement2 方法来建立圆。

- Function CreateEllipseElement1(Template As Element, PerimeterPoint1 As Point3d, PerimeterPoint2 As Point3d, PerimeterPoint3 As Point3d, [FillMode As MsdFillMode = msdFillModeUseActive]) As EllipseElement
- Function CreateEllipseElement2(Template As Element, Origin As Point3d, PrimaryRadius As Double, SecondaryRadius As Double, Rotation As Matrix3d, [FillMode As MsdFillMode = msdFillModeUseActive]) As EllipseElement

我们先从 CreateEllipseElement2 开始。

```
Sub TestCreateCircleA()
    Dim CPoint As Point3d
    Dim myEllipse As EllipseElement
    Dim rotMatrix As Matrix3d
    CPoint.X = 2.5: CPoint.Y = 2.5
```

```
        Set myEllipse = CreateEllipseElement2(Nothing, CPoint, 0.5, 0.5, _
                                                        rotMatrix)
        ActiveModelReference.AddElement   myEllipse
    End Sub
```

中心点设在(2.5,2.5,0)，半径为 0.5。给 PrimaryRadius(长轴)参数和 SecondaryRadius(短轴)参数相同的值，其结果是一个圆。如果长轴半径和短轴半径不相等，将建立一个椭圆。

```
    Sub TestCreateCircleB()
        Dim CPoint As Point3d
        Dim myEllipse As EllipseElement
        Dim rotMatrix As Matrix3d
        Dim CirRad As Double
        CPoint.X = 2.5: CPoint.Y = 2.5
        For CirRad = 0.5 To 2 Step 0.125
            Set myEllipse = CreateEllipseElement2(Nothing, CPoint, _
                                                    CirRad, CirRad, rotMatrix)
            ActiveModelReference.AddElement myEllipse
        Next CirRad
    End Sub
```

TestCreateCircleB 建立了一系列的同心圆，它们的半径从 0.5 到 2，增量为 0.125 单位。

下一个过程让用户来选择所绘圆的中心点，半径是 0.5。

```
    Sub TestCreateCircleC()
        Dim CPoint As Point3d
        Dim myEllipse As EllipseElement
        Dim rotMatrix As Matrix3d
        Dim inputQueue As CadInputQueue
        Dim inputMessage As CadInputMessage
        Set inputQueue = CadInputQueue
        Set inputMessage = _
```


China BDN

```
            inputQueue.GetInput(msdCadInputTypeDataPoint, _
              msdCadInputTypeAny)
        Do
            Select Case inputMessage.InputType
                Case msdCadInputTypeDataPoint
                    CPoint = inputMessage.point
                    Set myEllipse = CreateEllipseElement2(Nothing, _
                                CPoint, 0.5, 0.5, rotMatrix)
                    ActiveModelReference.AddElement   myEllipse
                    Exit Do
                Case msdCadInputTypeReset
                    Exit Do
            End Select
        Loop
    End Sub
```

最后一个要写的建立圆的过程让用户选择两个点，然后过这两个选择的点绘制一个圆。

```
Sub TestCreateCircleD()
    Dim CPoint As Point3d
    Dim StPoint As Point3d
    Dim EnPoint As Point3d
    Dim myEllipse As EllipseElement
    Dim rotMatrix As Matrix3d
    Dim inputQueue As CadInputQueue
    Dim inputMessage As CadInputMessage
    Dim CirRad As Double
    Set inputQueue = CadInputQueue
    Set inputMessage = _
        inputQueue.GetInput(msdCadInputTypeDataPoint, _
                    msdCadInputTypeAny)
```

```
    Do
        Select Case inputMessage.InputType
            Case msdCadInputTypeDataPoint
                StPoint = inputMessage.point
                Exit Do
            Case msdCadInputTypeReset
                Exit Sub
        End Select
    Loop
    Set inputMessage = inputQueue.GetInput(msdCadInputTypeDataPoint, _
                                           msdCadInputTypeAny)
    Do
        Select Case inputMessage.InputType
            Case msdCadInputTypeDataPoint
                EnPoint = inputMessage.point
                Exit Do
            Case msdCadInputTypeReset
                Exit Sub
        End Select
    Loop
    CPoint.X = StPoint.X + (EnPoint.X - StPoint.X) / 2
    CPoint.Y = StPoint.Y + (EnPoint.Y - StPoint.Y) / 2
    CPoint.Z = StPoint.Z + (EnPoint.Z - StPoint.Z) / 2
    CirRad = Point3dDistance(StPoint, EnPoint) / 2
    Set myEllipse = CreateEllipseElement2(Nothing, CPoint, _
                                          CirRad, CirRad, rotMatrix)
    ActiveModelReference.AddElement myEllipse
End Sub
```

我们用选择的点计算圆的中心点，还用了 MicroStation VBA 的

Point3dDistance 函数算出了两点间的距离。

建立椭圆

我们已经使用过建立椭圆的代码，但当时建立的是圆，因为长轴和短轴相等。现在让我们看三个建立椭圆的例子。

```
Sub TestCreateEllipseA()
    Dim CPoint As Point3d
    Dim myEllipse As EllipseElement
    Dim rotMatrix As Matrix3d
    CPoint.X = 2.5: CPoint.Y = 2.5
    Set myEllipse = CreateEllipseElement2(Nothing, CPoint, 1, 0.5, _
                                          rotMatrix)
    ActiveModelReference.AddElement    myEllipse
End Sub

Sub TestCreateEllipseB()
    Dim MajorA1 As Point3d
    Dim MajorA2 As Point3d
    Dim MinorA1 As Point3d
    Dim myEllipse As EllipseElement
    MajorA1.X = 1: MajorA1.Y = 1
    MajorA2.X = 5: MajorA2.Y = 5
    MinorA1.X = 3: MinorA1.Y = 2
    Set myEllipse = CreateEllipseElement1(Nothing, MajorA1, MajorA2, _
                                          MinorA1)
    ActiveModelReference.AddElement    myEllipse
End Sub

Sub TestCreateEllipseC()
    Dim CPoint As Point3d
```

China BDN

```
    Dim myEllipse As EllipseElement
    Dim rotMatrix As Matrix3d
    CPoint.X = 2.5: CPoint.Y = 2.5
    rotMatrix.RowX.X = 2
    rotMatrix.RowY.X = 4: rotMatrix.RowY.Y = 5
    Set myEllipse = CreateEllipseElement2(Nothing, CPoint, 1, 0.5, _
                                          rotMatrix)
    ActiveModelReference.AddElement   myEllipse
End Sub
```

运行完以上过程后发现了什么？三个过程中的两个建立了椭圆，然而，过程 TestCreateEllipseB 建立了一个圆。方法 CreateEllipseElement1 通过三个点总是建立一个圆。

建立弧

MicroStation VBA 中有五种不同的建立弧的方法。

1、Function CreateArcElement1(Template As Element, StartPoint As Point3d, CenterPoint As Point3d, EndPoint As Point3d) As ArcElement

2、Function CreateArcElement2(Template As Element, CenterPoint As Point3d, PrimaryRadius As Double, SecondaryRadius As Double, Rotation As Matrix3d, StartAngle As Double, SweepAngle As Double) As ArcElement

3、Function CreateArcElement3(Template As Element, StartPoint As Point3d, PointOnCurve As Point3d, EndPoint As Point3d) As ArcElement

4、Function CreateArcElement4(Template As Element, StartTangent As Ray3d, EndPoint As Point3d) As ArcElement

5、Function CreateArcElement5(Template As Element, Chord As Segment3d, ArcLength As Double, PlanePoint As Point3d) As ArcElement

让我们看几个使用这些方法的过程。

```
Sub TestCreateArcA()
    Dim CPoint As Point3d
    Dim StPoint As Point3d
    Dim EnPoint As Point3d
    Dim myArc As ArcElement
    CPoint.X = 1: CPoint.Y = 1
    StPoint.X = 4: StPoint.Y = 1
    EnPoint.X = 1: EnPoint.Y = 4
    Set myArc = CreateArcElement1(Nothing, StPoint, CPoint, EnPoint)
    ActiveModelReference.AddElement    myArc
End Sub

Sub TestCreateArcB()
    Dim CPoint As Point3d
    Dim rotMatrix As Matrix3d
    Dim myArc As ArcElement
    CPoint.X = 1: CPoint.Y = 1
    Set myArc = CreateArcElement2(Nothing, CPoint, 0.5, 0.5, _
            rotMatrix, 0, Pi)
    ActiveModelReference.AddElement    myArc
End Sub

Sub TestCreateArcC()
    Dim PointA As Point3d
    Dim PointB As Point3d
    Dim PointC As Point3d
    Dim myArc As ArcElement
    PointA.X = 1: PointA.Y = 1
    PointB.X = 2: PointB.Y = 2
```

China BDN

```
    PointC.X = 1: PointC.Y = 3
    Set myArc = CreateArcElement3(Nothing, PointA, PointB, PointC)
    ActiveModelReference.AddElement    myArc
End Sub

Sub TestCreateArcD()
    Dim myArc As ArcElement
    Dim myRay As Ray3d
    Dim EndPoint As Point3d
    myRay.Origin.X = 1
    myRay.Origin.Y = 1
    myRay.Direction.X = 1
    myRay.Direction.Y = 4
    EndPoint.X = 0: EndPoint.Y = 2
    Set myArc = CreateArcElement4(Nothing, myRay, EndPoint)
    ActiveModelReference.AddElement    myArc
End Sub

Sub TestCreateArcE()
    Dim myArc As ArcElement
    Dim mySeg As Segment3d
    Dim myPoint As Point3d
    mySeg.startPoint.X = 1: mySeg.startPoint.Y = 1
    mySeg.EndPoint.X = 4: mySeg.EndPoint.Y = 4
    myPoint.X = 3.5: myPoint.Y = 3: myPoint.Z = 0
    Set myArc = CreateArcElement5(Nothing, mySeg, 8.5, myPoint)
    ActiveModelReference.AddElement    myArc
End Sub
```

China BDN

建立文本

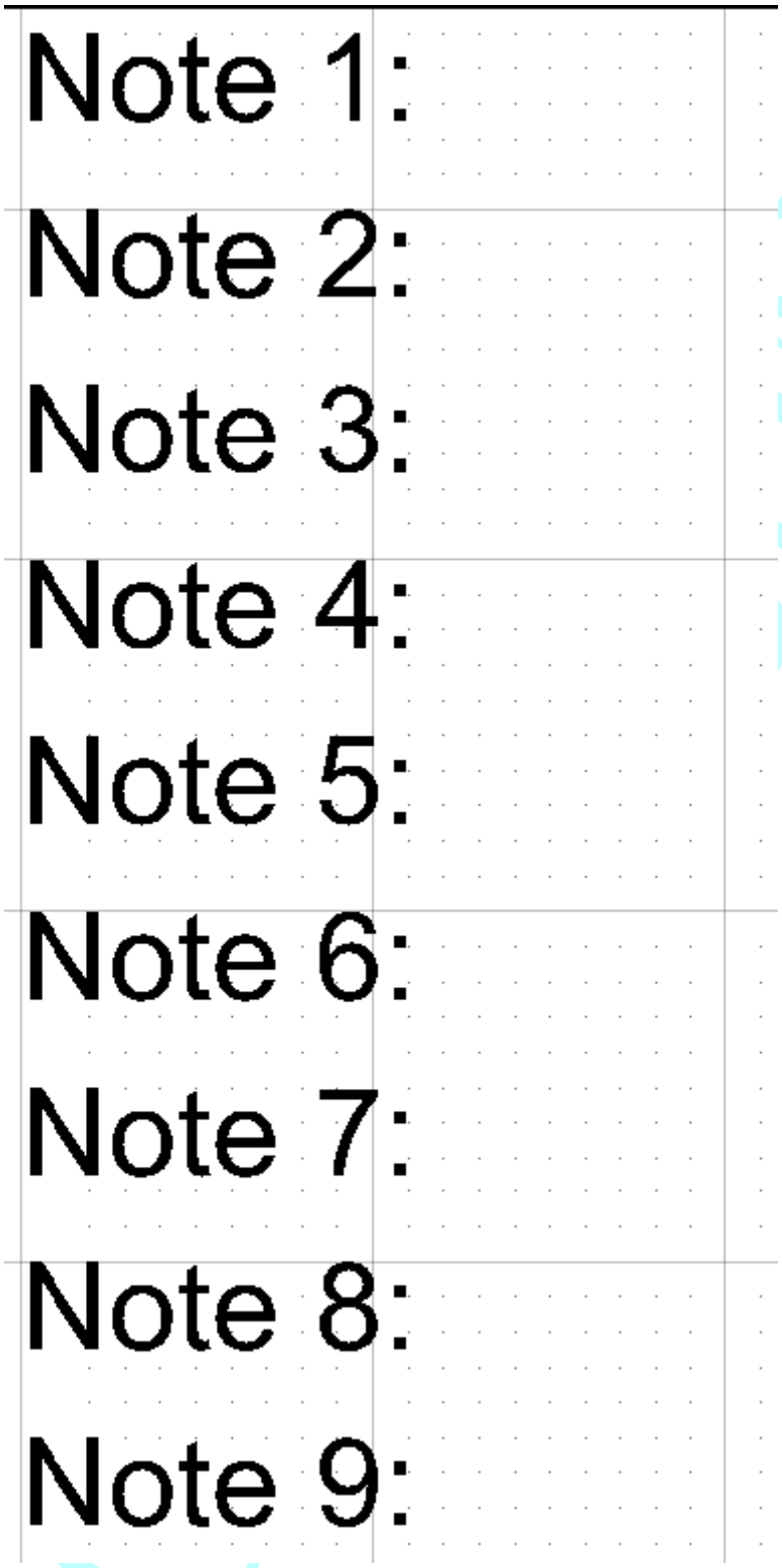
用 CreateTextElement1 方法很容易建立文本。

- Function CreateTextElement1(Template As Element, Text As String, Origin As Point3d, Rotation As Matrix3d) As TextElement

这里是一个建立九个文本元素的例子，这些文本元素相互间距 0.5 个单位。

```
Sub TestCreateTextA()
    Dim myText As TextElement
    Dim TextPt As Point3d
    Dim rotMatrix As Matrix3d
    Dim I As Double
    For I = 1 To 9
        TextPt.Y = TextPt.Y - 0.5
        Set myText = CreateTextElement1(Nothing, "Note " & I & _
                ":", TextPt, rotMatrix)
        ActiveModelReference.AddElement myText
    Next I
End Sub
```

China BDN



建立单元

到目前为止，我们建立的所有元素都以单独的元素添加到设计文件中。当操作单元时，我们把多个元素作为一个单元来处理。我们在建立单元时和建立普通元素的工作方式是一样的，普通元素添加到了模型中，但单元中的元素添加到了

单元中。建立单元有三种方式。

1、Function CreateCellElement1(Name As String, Elements() As _Element, Origin As Point3d, [IsPointCell As Boolean]) As CellElement

2、Function CreateCellElement2(CellName As String, Origin As Point3d, Scale As Point3d, TrueScale As Boolean, Rotation As Matrix3d) As CellElement

3、Function CreateCellElement3(CellName As String, Origin As Point3d, TrueScale As Boolean) As CellElement

第一个例子建立一个叫做“Box”的单元。四条线被加入到一个元素数组中，该数组用于建立单元。

```
Sub TestCreateCellA()
    Dim myCell As CellElement
    Dim BoxLines(0 To 3) As Element
    Dim OriginPoint As Point3d
    Set BoxLines(0) = CreateLineElement2(Nothing, _
                    Point3dFromXYZ(0, 0, 0), Point3dFromXYZ(4, 0, 0))
    Set BoxLines(1) = CreateLineElement2(Nothing, _
                    Point3dFromXYZ(4, 0, 0), Point3dFromXYZ(4, 4, 0))
    Set BoxLines(2) = CreateLineElement2(Nothing, _
                    Point3dFromXYZ(4, 4, 0), Point3dFromXYZ(0, 4, 0))
    Set BoxLines(3) = CreateLineElement2(Nothing, _
                    Point3dFromXYZ(0, 4, 0), Point3dFromXYZ(0, 0, 0))
    OriginPoint.X = 2: OriginPoint.Y = 2
    Set myCell = CreateCellElement1("Box", BoxLines, OriginPoint)
    ActiveModelReference.AddElement    myCell
    myCell.Redraw
End Sub
```

TestCreateCellA 建立了一个原点为(2,2,0)的边长为 4 个单位的正方形。

```
Sub TestCreateCellB()
    Dim myCell As CellElement
    Dim CellElements(0 To 6) As Element
    Dim OriginPoint As Point3d
    Dim rotMatrix As Matrix3d
    Set CellElements(0) = CreateLineElement2(Nothing, _
                Point3dFromXYZ(0, 0, 0), Point3dFromXYZ(4, 0, 0))
    Set CellElements(1) = CreateLineElement2(Nothing, _
                Point3dFromXYZ(4, 0, 0), Point3dFromXYZ(4, 4, 0))
    Set CellElements(2) = CreateLineElement2(Nothing, _
                Point3dFromXYZ(4, 4, 0), Point3dFromXYZ(0, 4, 0))
    Set CellElements(3) = CreateLineElement2(Nothing, _
                Point3dFromXYZ(0, 4, 0), Point3dFromXYZ(0, 0, 0))
    Set CellElements(4) = CreateLineElement2(Nothing, _
                Point3dFromXYZ(0, 0, 0), Point3dFromXYZ(4, 4, 0))
    Set CellElements(5) = CreateLineElement2(Nothing, _
                Point3dFromXYZ(4, 0, 0), Point3dFromXYZ(0, 4, 0))
    OriginPoint.X = 2: OriginPoint.Y = 2
    Set CellElements(6) = CreateEllipseElement2(Nothing, _
                OriginPoint, 1.25, 1.25, rotMatrix)
    Set myCell = CreateCellElement1("Box2", CellElements, OriginPoint)
    ActiveModelReference.AddElement    myCell
    myCell.Redraw
End Sub
```

随着加入单元的元素数量的增加，数组上界数也要增加。TestCreateCellB用六条线和一个圆建立了单元“Box2”。

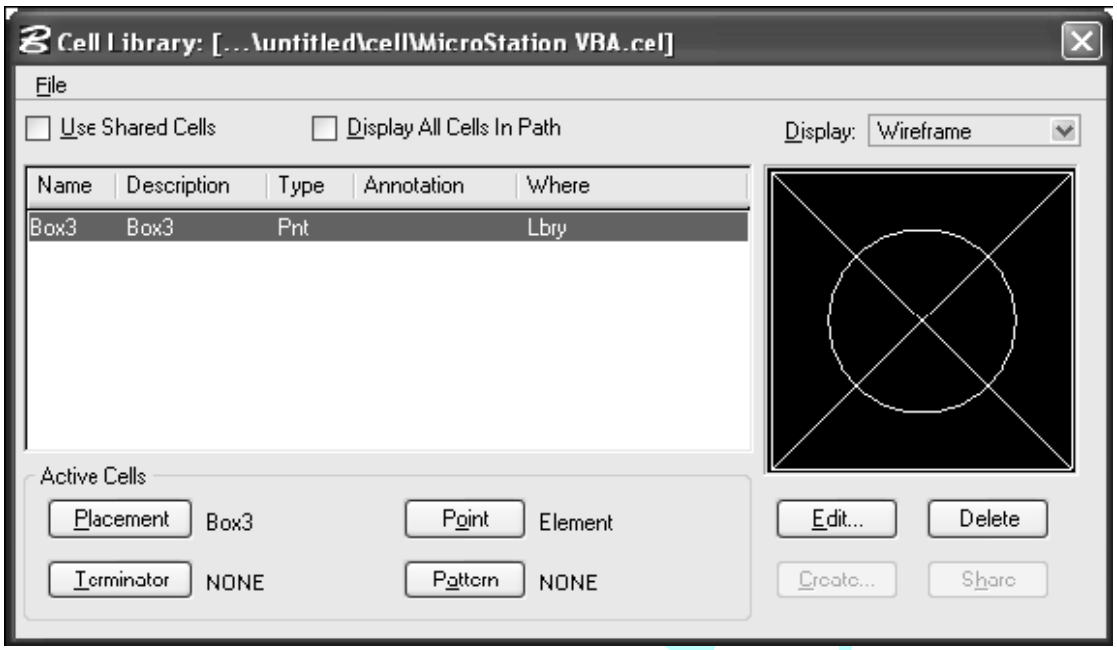
把单元加入一个单元库中有利于在别的文件中建立单元。

```
Sub TestCreateCellC()
    Dim myCell As CellElement
    Dim CellElements(0 To 6) As Element
```

```
    Dim OriginPoint As Point3d
    Dim rotMatrix As Matrix3d
    Set CellElements(0) = CreateLineElement2(Nothing, _
                Point3dFromXYZ(0, 0, 0), Point3dFromXYZ(4, 0, 0))
    Set CellElements(1) = CreateLineElement2(Nothing, _
                Point3dFromXYZ(4, 0, 0), Point3dFromXYZ(4, 4, 0))
    Set CellElements(2) = CreateLineElement2(Nothing, _
                Point3dFromXYZ(4, 4, 0), Point3dFromXYZ(0, 4, 0))
    Set CellElements(3) = CreateLineElement2(Nothing, _
                Point3dFromXYZ(0, 4, 0), Point3dFromXYZ(0, 0, 0))
    Set CellElements(4) = CreateLineElement2(Nothing, _
                Point3dFromXYZ(0, 0, 0), Point3dFromXYZ(4, 4, 0))
    Set CellElements(5) = CreateLineElement2(Nothing, _
                Point3dFromXYZ(4, 0, 0), Point3dFromXYZ(0, 4, 0))
    OriginPoint.X = 2: OriginPoint.Y = 2
    Set CellElements(6) = CreateEllipseElement2(Nothing, _
                    OriginPoint, 1.25, 1.25, rotMatrix)
    Set myCell = CreateCellElement1("Box3", CellElements, _
                OriginPoint)
    ActiveModelReference.AddElement    myCell
    myCell.Redraw
    Application.AttachCellLibrary "MicroStation VBA.cel"
    Application.AttachedCellLibrary.AddCell    myCell, "Box3", "Box3", False
End Sub
```

注意，我们指定了单元库的文件名，但没有指定全路径而仅仅是文件名。

China RDN



建立新文档

我们已经在当前设计文件中绘制了线、圆、椭圆、弧、文本和单元。这一切都假定已经有了一个要操作的文件，那么如何建立一个新的设计文件呢？

1、Function CreateDesignFile(SeedFileName As String,
NewDesignFileName As String, Open As Boolean) As DesignFile

2、Sub CopyDesignFile(ExistingDesignFileName As String,
NewDesignFileName As String, [Overwrite As Boolean])

以上是建立新设计文件的两种方法。CreateDesignFile 通过指定种子文档而让我们指定新文件是 2D 还是 3D。让我们看几个例子。

```
Sub TestCreateDesignFileA()
    Dim myFile As DesignFile
    Application.ActiveDesignFile.Close
    Set myFile = CreateDesignFile("seed2d", _
                "C:\MicroStation VBA\filea.dgn", True)
End Sub
```

TestCreateDesignFileA 建立一个新的 2D 设计文件，程序中指定了文件路径和文件名。CreateDesignFile 代码行之行后，新文件被建立并打开，这个新文件成为当前文档。如果指定的文件已存在，新文件被建立并覆盖原来的文件。由

于覆盖已存在的文件时不会收到任何警告，所以我们应当检查是否存在原文件。

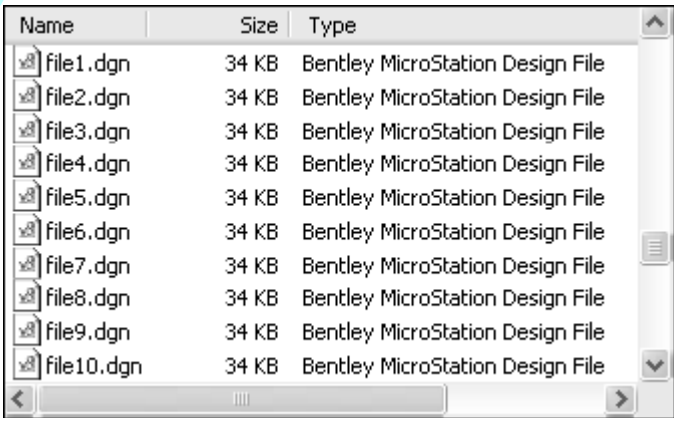
```
Sub TestCreateDesignFileB()
    Dim myFile As DesignFile
    Dim myFileName As String
    myFileName = "C:\MicroStation VBA\filea.dgn"
    If Dir(myFileName) = "" Then
        Set myFile = CreateDesignFile("seed3d", myFileName, True)
    Else
        MsgBox "The file " & myFileName & " already exists.", vbCritical
    End If
End Sub
```

如果想建立的文件已存在(通过使用 Dir 函数知道的这一点)，就通知用户该文件已存在；如果想建立的文件不存在，就建立一个新的 3D 文件。

让我们再看一个例子：

```
Sub TestCreateDesignFileC()
    Dim myFile As DesignFile
    Dim I As Long
    For I = 1 To 10
        Set myFile = CreateDesignFile("seed2d", _
                "C:\MicroStation VBA\file" & I & ".dgn", False)
    Next I
End Sub
```

TestCreateDesignFileC 建立了多少个文件？它建立了十(10)个文件，每个文件都是一个新的 2D 文件且这些文件没有在 MicroStation 中打开(False 参数起的作用)。



China BDN

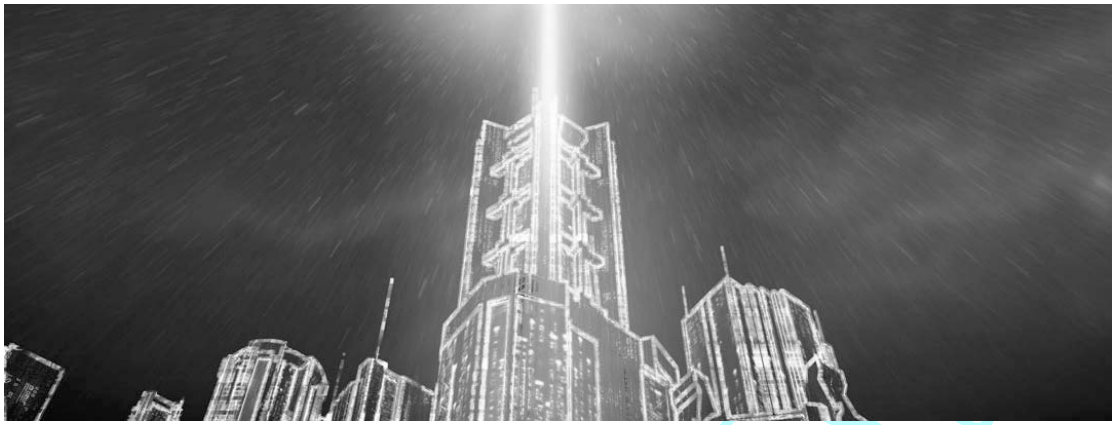
建立数据的安全性问题

能否用 VBA 在 MicroStation 中建立数据决定于安全设置，VBA 不会跳过这些安全设置。在一台机器上编写并试着运行本章中的过程可能能够生成想要的数据，但在另一台具有不同安全权限的机器可能就会使代码运行失败。如果出现这种问题，CAD 管理员应该能够为您提供合适的权限。

本章回顾

只要知道几个 MicroStation VBA 调用就能建立简单的几何图形。对象浏览器和 MicroStation VBA 帮助文件可用于查询建立数据的其它方法，同时还能提供如何使用它们的例子。

China BDN



16. 在文件中查找

设计文件的复杂度能从一或两个元素到数千个元素。元素个数、元素类型(线、圆、弧、文本)和颜色都能变化，层、线型和类别也因元素的不同而不同，线宽和透明度也能变化。在开始文件中的查找前，需要先学习如何在文件中发现这些属性。

本章包括：

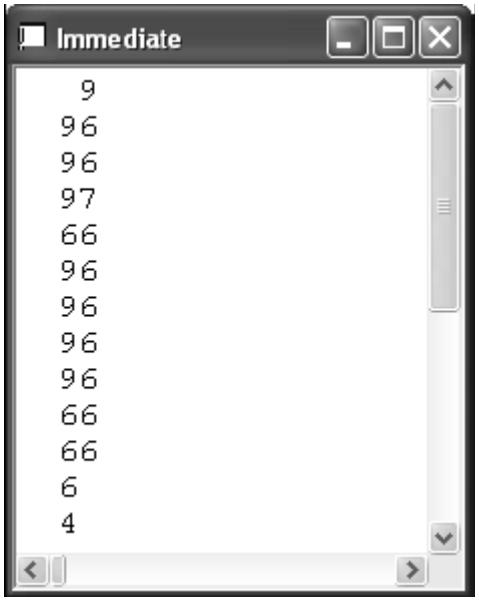
- 在文件中查找的基础知识
- 使用 ScanCriteria(扫描准则)
- 多个准则的组合
- 回顾三种集合方法
- 扫描准则的方法

在文件中查找的基础知识

先让我们看一看在文件中找到的每一个元素。

```
Sub TestScanAllA()
    Dim myElement As Element
    Dim myEnum As ElementEnumerator
    Set myEnum = ActiveModelReference.Scan()
    While myEnum.MoveNext
        Set myElement = myEnum.Current
        Debug.Print myElement.Type
```

```
    Wend
End Sub
```



该过程将当前模型中的每个元素的类型属性值都打印到了立即窗口。

运行过程 TestScallAllA 产生一系列数字，这些数字是找到的元素的类型，它们引用自 msdElementType 枚举类型。

下面是 msdElementType 枚举的成员列表：

msdElementType44 = 44

msdElementTypeArc = 16

msdElementTypeBsplineBoundary = 25

msdElementTypeBsplineCurve = 27

msdElementTypeBsplineKnot = 26

msdElementTypeBsplinePole = 21

msdElementTypeBsplineSurface = 24

msdElementTypeBsplineWeight = 28

msdElementTypeCellHeader = 2

msdElementTypeCellLibraryHeader = 1

msdElementTypeComplexShape = 14

msdElementTypeComplexString = 12

msdElementTypeCone = 23

msdElementTypeConic = 13

msdElementTypeCurve = 11

msdElementTypeDesignFileHeader = 9

msdElementTypeDgnStoreComponent = 38

msdElementTypeDgnStoreHeader = 39

msdElementTypeDigSetData = 8

msdElementTypeDimension = 33

msdElementTypeEllipse = 15

msdElementTypeGroupData = 5

msdElementTypeLevelMask = 99

msdElementTypeLevelSymbology = 10

msdElementTypeLine = 3

msdElementTypeLineString = 4

msdElementTypeMatrixDoubleData = 103

msdElementTypeMatrixHeader = 101

msdElementTypeMatrixIntegerData = 102

msdElementTypeMeshHeader = 105

msdElementTypeMicroStation = 66

msdElementTypeMultiLine = 36

msdElementTypeNamedGroupComponent = 111

msdElementTypeNamedGroupHeader = 110

msdElementTypePointString = 22

msdElementTypeRasterComponent = 88

msdElementTypeRasterFrame = 94

msdElementTypeRasterHeader = 87

msdElementTypeRasterReference = 90

msdElementTypeRasterReferenceComponent = 91

msdElementTypeReferenceAttachment = 100

msdElementTypeReferenceOverride = 108

msdElementTypeShape = 6

msdElementTypeSharedCell = 35

msdElementTypeSharedCellDefinition = 34

msdElementTypeSolid = 19

msdElementTypeSurface = 18

China BDN

msdElementTypeTable = 96

msdElementTypeTableEntry = 95

msdElementTypeTag = 37

msdElementTypeText = 17

msdElementTypeTextNode = 7

msdElementTypeView = 98

msdElementTypeViewGroup = 97

立即窗口中显示的前三个元素类型是 9、96 和 97。查阅以上列表可知前三个元素类型为：

msdElementTypeDesignFileHeader = 9

msdElementTypeTable = 96

msdElementTypeViewGroup = 97

这三个类型全然不是线、圆或弧，这对吗？其实 MicroStation 设计文件远不止由我们在屏幕上看到的那些元素组成。下面三个元素类型是什么？66、6 和 4。

msdElementTypeMicroStation = 66

msdElementTypeShape = 6

msdElementTypeLineString = 4

现在有一些进展了，我们能看到形和线串了。

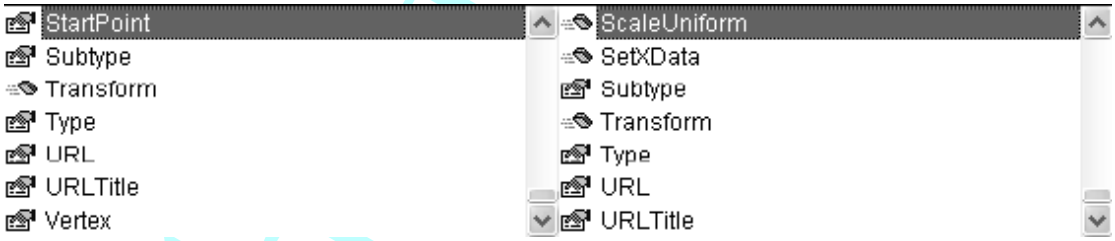
这一章里将会进行大量的复制和粘贴操作，让我们先复制 TestScallAllA 并粘贴成 TestScanAllB。

```
Sub TestScanAllB()
    Dim myElement As Element
    Dim myEnum As ElementEnumerator
    Set myEnum = ActiveModelReference.Scan()
    While myEnum.MoveNext
        Set myElement = myEnum.Current
        Select Case myElement.Type
            Case msdElementTypeArc
                Dim myArc As ArcElement
                Set myArc = myElement
```


China BDN

```
            Case msdElementTypeCurve
                Dim myCurve As CurveElement
                Set myCurve = myElement
            Case msdElementTypeLine
                Dim myLine As LineElement
                Set myLine = myElement
            Case msdElementTypeText
                Dim myText As TextElement
                Set myText = myElement
            Case Else
                Debug.Print myElement.Type
        End Select
    Wend
End Sub
```

我们利用 Select ... Case 语句按 Element.Type 属性来执行不同的动作。当在 ElementEnumerator 中遍历每个元素时，我们把每个元素设置给一个通用的元素对象。如果想操作一个 LineElement，可以通过这个通用的元素对象来完成，但声明一个 LineElement 会使编程工作更容易。让我们看一下为什么会是这样。



在编程操作一个线元素时，以上哪个列表对我们更有帮助呢？左边的列表给出了 StartPoint(起点)属性，线具有起点，而右边的列表没有 StartPoint。如果声明一个 LineElement 变量，除了标准元素属性外还能看到线所特有的属性。

让我们对上面的过程再多做点工作：复制并粘贴为 TestScanAllC，除“Case msdElementTypeText”段外删去 Select ... Case 语句中的所有内容。

```
Sub TestScanAllC()
    Dim myElement As Element
    Dim myEnum As ElementEnumerator
```

```
    Set myEnum = ActiveModelReference.Scan()
    While myEnum.MoveNext
        Set myElement = myEnum.Current
        Select Case myElement.Type
            Case msdElementTypeText
                Dim myText As TextElement
                Set myText = myElement
                myText.Text = UCase(myText.Text)
        End Select
    Wend
End Sub
```

现在，我们的过程仅对文本元素作出反应。那么我们对文本元素做了什么事情呢？用 UCase 将每个字符都转换为大写。该过程的结果应该是将所有文本元素都转换成了大写，对吗？

这段代码运行后，我们会发现什么也没有变化。这怎么可能？这段代码确实将文本转换成大写了呀！让我们仔细分析下一个过程，看能否发现本过程遗忘了什么。

```
Sub TestScanAllD()
    Dim myElement As Element
    Dim myEnum As ElementEnumerator
    Set myEnum = ActiveModelReference.Scan()
    While myEnum.MoveNext
        Set myElement = myEnum.Current
        Select Case myElement.Type
            Case msdElementTypeText
                Dim myText As TextElement
                Set myText = myElement
                myText.Text = UCase(myText.Text)
                myText.Rewrite
        End Select
```


China BDN

```
    Wend
End Sub
```

如果我们不把元素重写(Rewrite)回模型，文本元素只在内存中被改变，这种改变实际上没有反映到设计文件中。

使用 ScanCriteria

现在，假定我们正操作一个大文件，它由数千个元素组成但只有四个元素是文本元素(TextElement)。如果运行上面所示的代码，文本元素确实能被转换为大写。然而，运行这段代码可能会花上一定的时间，因为设计文件中的每个元素都需要被查看。如果仅操作文本元素就能使代码效率更高，通过使用 ElementScanCriteria 对象来实现这一点。

```
Sub TestScanFilterA()
    Dim myEnum As ElementEnumerator
    Dim myFilter As New ElementScanCriteria
    Dim ElementCounter As Long
    myFilter.IncludeType msdElementTypeText
    myFilter.IncludeType msdElementTypeTextNode
    Set myEnum = ActiveModelReference.Scan(myFilter)
    While myEnum.MoveNext
        ElementCounter = ElementCounter + 1
    Wend
    MsgBox ElementCounter & " elements found."
End Sub
```

当包含了文本和文本节点元素时，应该仅计数文本和文件节点元素的个数。然而，运行完这段代码后工作效果并不像我们所期望的那样。

经过仔细检查我们发现，缺省情况下 ScanCriteria 包含所有元素。所以，在指定要查找哪些元素前需要排除所有元素然后再包含要找的元素。

```
Sub TestScanFilterB()
    Dim myEnum As ElementEnumerator
    Dim myFilter As New ElementScanCriteria
```

```
    Dim ElementCounter As Long
    myFilter.ExcludeAllTypes
    myFilter.IncludeType msdElementTypeText
    myFilter.IncludeType msdElementTypeTextNode
    Set myEnum = ActiveModelReference.Scan(myFilter)
    While myEnum.MoveNext
        ElementCounter = ElementCounter + 1
    Wend
    MsgBox ElementCounter & " elements found."
End Sub
```

现在，**myEnum** 仅包含文本和文本节点元素。

下面在 TestScanFilterB 的基础上添加对指定层的过滤。在查找指定层前必须排除所有层，如果忘记了这一关键性步骤，将会检索出所有层。

```
Sub TestScanFilterC()
    Dim myEnum As ElementEnumerator
    Dim myFilter As New ElementScanCriteria
    Dim ElementCounter As Long
    myFilter.ExcludeAllTypes
    myFilter.ExcludeAllLevels
    myFilter.IncludeType msdElementTypeText
    myFilter.IncludeType msdElementTypeTextNode
    myFilter.IncludeLevel ActiveDesignFile.Levels("SIDEWALK")
    Set myEnum = ActiveModelReference.Scan(myFilter)
    While myEnum.MoveNext
        ElementCounter = ElementCounter + 1
    Wend
    MsgBox ElementCounter & " elements found."
End Sub
```

大致看一下宏 TestScanFilterC，在这个程序中什么被计数？“SIDEWALK”

China BDN

层上的文本元素和文本节点元素。

```
Sub TestScanFilterD()
    Dim myEnum As ElementEnumerator
    Dim myFilter As New ElementScanCriteria
    Dim ElementCounter As Long
    myFilter.ExcludeAllTypes
    myFilter.ExcludeAllLevels
    myFilter.ExcludeAllColors
    myFilter.IncludeType msdElementTypeText
    myFilter.IncludeType msdElementTypeTextNode
    myFilter.IncludeLevel ActiveDesignFile.Levels("SIDEWALK")
    myFilter.IncludeColor 4
    Set myEnum = ActiveModelReference.Scan(myFilter)
    While myEnum.MoveNext
        ElementCounter = ElementCounter + 1
    Wend
    MsgBox ElementCounter & " elements found."
End Sub
```

我们又多加了一条扫描准则。除考察元素类型和层以外，现在又加了颜色。

如果知道文档颜色表中的一个颜色索引，就能如上所示指定它。请看下一个例子，在那里我们指定了一个 RGB 颜色值来对指定的颜色进行过滤。还将再添加一项扫描准则，那是一个线型准则。

```
Sub TestScanFilterF()
    Dim myEnum As ElementEnumerator
    Dim myFilter As New ElementScanCriteria
    Dim ElementCounter As Long
    Dim myColorTable As ColorTable
    Dim myColor As Long
    Set myColorTable = ActiveDesignFile.ExtractColorTable
    myColor = myColorTable.FindClosestColor(RGB(192, 192, 192))
```


China BDN

```
    myFilter.ExcludeAllTypes
    myFilter.ExcludeAllLevels
    myFilter.ExcludeAllColors
    myFilter.ExcludeAllLineStyles
    myFilter.IncludeType msdElementTypeLineString
    myFilter.IncludeLineStyle ActiveDesignFile.LineStyles("( Hidden )")
    myFilter.IncludeLevel ActiveDesignFile.Levels("SIDEWALK")
    myFilter.IncludeColor myColor - 1
    Set myEnum = ActiveModelReference.Scan(myFilter)
    While myEnum.MoveNext
        ElementCounter = ElementCounter + 1
    Wend
    MsgBox ElementCounter & " elements found."
End Sub
```

下一个例子又加入了另一个扫描准则：

```
Sub TestScanFilterG()
    Dim myEnum As ElementEnumerator
    Dim myFilter As New ElementScanCriteria
    Dim ElementCounter As Long
    Dim myColorTable As ColorTable
    Dim myColor As Long
    Set myColorTable = ActiveDesignFile.ExtractColorTable
    myColor = myColorTable.FindClosestColor(RGB(192, 192, 192))
    myFilter.ExcludeAllTypes
    myFilter.ExcludeAllLevels
    myFilter.ExcludeAllColors
    myFilter.ExcludeAllLineStyles
    myFilter.ExcludeAllClasses
    myFilter.IncludeType msdElementTypeLineString
    myFilter.IncludeLineStyle ActiveDesignFile.LineStyles("( Hidden )")
```


China BDN

```
    myFilter.IncludeLevel ActiveDesignFile.Levels("SIDEWALK")
    myFilter.IncludeColor myColor - 1
    myFilter.IncludeClass msdElementClassPrimary
    Set myEnum = ActiveModelReference.Scan(myFilter)
    While myEnum.MoveNext
        ElementCounter = ElementCounter + 1
    Wend
    MsgBox ElementCounter & " elements found."
End Sub
```

我们在扫描准则中加入了“类别(Class)”。

迄今为止，我们已经从准则中排除了所有的东西，并且仅加入想要的准则。当用“ExcludeAllLevels”排除所有层时，实际所排除的层数随文件的不同而不同。

现在，让我们考察设计文件中的每个层。只有一个指定的层没有加入扫描准则，其它的都加入了。

```
Sub TestScanFilterH()
    Dim myEnum As ElementEnumerator
    Dim myFilter As New ElementScanCriteria
    Dim myLevel As Level
    Dim ElementCounter As Long
    myFilter.ExcludeAllLevels
    For Each myLevel In ActiveDesignFile.Levels
        Select Case UCase(myLevel.Name)
            Case "SIDEWALK"
            Case Else
                myFilter.IncludeLevel myLevel
        End Select
    Next
    Set myEnum = ActiveModelReference.Scan(myFilter)
    While myEnum.MoveNext
```


China BDN

```
        ElementCounter = ElementCounter + 1
    Wend
    MsgBox ElementCounter & " elements found."
End Sub
```

当处理 ElementScanCriteria 对象时，所有元素都在准则范围内。由于上例中没有选项来移除指定的元素类型或层等等，所以我们移除所有层，然后添加回满足准则的那些层。上例中除“SIDEWALK”层外添加了所有的层到准则中。

多个准则的组合

到目前为止，每个过程都是在处理符合特定准则的元素。如果想要搜索“Columns”层上的所有单元和“Marks”层上的所有文本又该如何做呢？下面是完成同样任务的三种方法。

```
Sub TestScanFilterJ()
    Dim myElem As Element
    Dim myEnum As ElementEnumerator
    Dim myEnum2 As ElementEnumerator
    Dim myFilter As New ElementScanCriteria
    Dim myFilter2 As New ElementScanCriteria
    Dim ElementCounter As Long
    myFilter.ExcludeAllTypes
    myFilter.ExcludeAllLevels
    myFilter.IncludeType msdElementTypeSharedCell
    myFilter.IncludeLevel ActiveDesignFile.Levels("COLUMNS")
    Set myEnum = ActiveModelReference.Scan(myFilter)
    While myEnum.MoveNext
        ElementCounter = ElementCounter + 1
    Wend
    myFilter2.ExcludeAllTypes
    myFilter2.ExcludeAllLevels
    myFilter2.IncludeType msdElementTypeText
```

```
    myFilter2.IncludeLevel ActiveDesignFile.Levels("MARKS")
    Set myEnum2 = ActiveModelReference.Scan(myFilter2)
    While myEnum2.MoveNext
        ElementCounter = ElementCounter + 1
    Wend
    MsgBox ElementCounter & " elements found."
End Sub
```

我们使用了带有两个扫描准则对象的两个不同的枚举对象。这是处理当前任务的一种方法。还有别的方法能实现同样的目的吗？

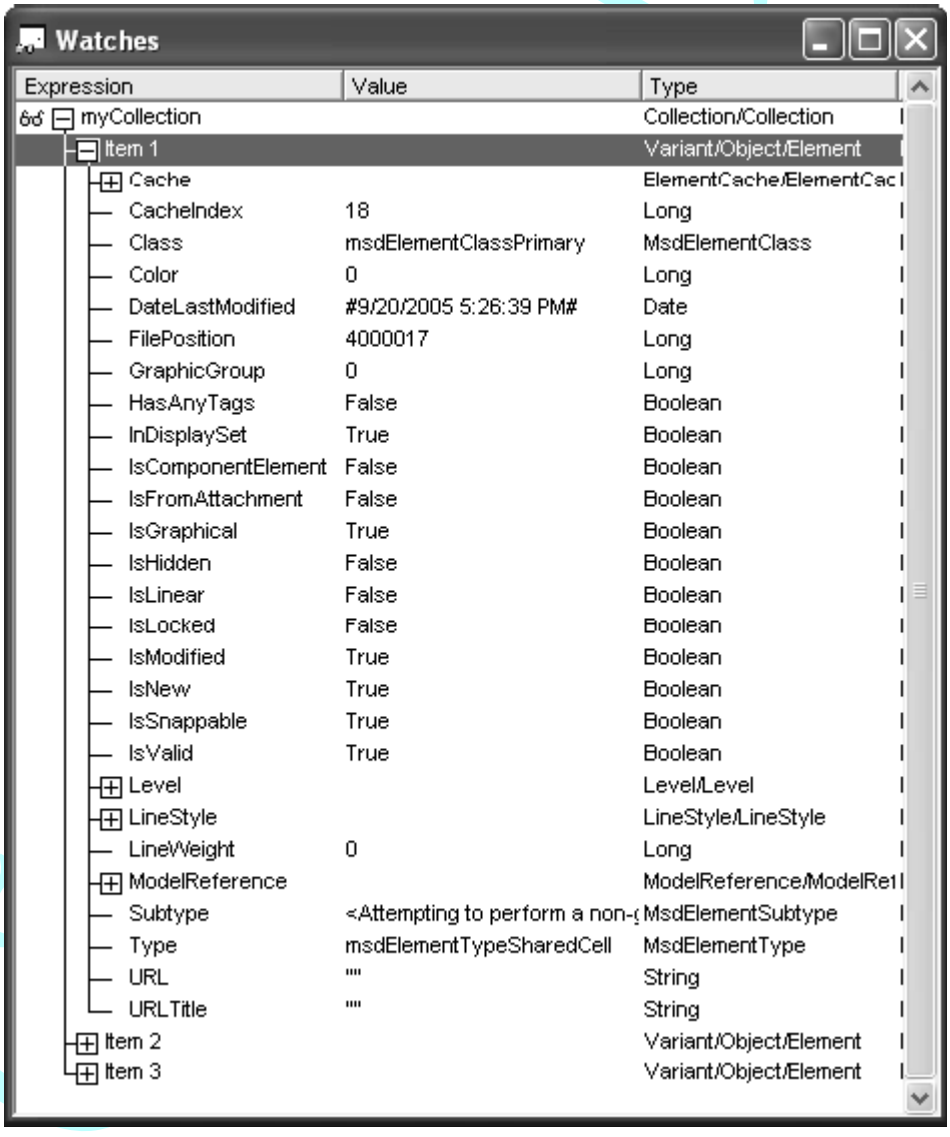
```
Sub TestScanFilterK()
    Dim myElem As Element
    Dim myEnum As ElementEnumerator
    Dim myFilter As New ElementScanCriteria
    Dim myCollection As New Collection
    myFilter.ExcludeAllTypes
    myFilter.ExcludeAllLevels
    myFilter.IncludeType msdElementTypeSharedCell
    myFilter.IncludeLevel ActiveDesignFile.Levels("COLUMNS")
    Set myEnum = ActiveModelReference.Scan(myFilter)
    While myEnum.MoveNext
        Set myElem = myEnum.Current
        myCollection.Add myElem
    Wend
    myFilter.Reset
    myFilter.ExcludeAllTypes
    myFilter.ExcludeAllLevels
    myFilter.IncludeType msdElementTypeText
    myFilter.IncludeLevel ActiveDesignFile.Levels("MARKS")
    Set myEnum = ActiveModelReference.Scan(myFilter)
    While myEnum.MoveNext
```


China BDN

```
        Set myElem = myEnum.Current
        myCollection.Add myElem
    Wend
    MsgBox myCollection.Count & " elements found."
End Sub
```

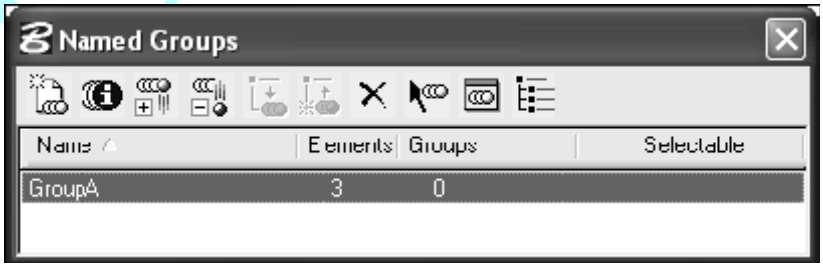
这是实现相同目标的另一种方法。我们给出了两个单独的准则。当在每个枚举器中遍历时，把枚举器中符合条件的元素添加到了一个自定义的集合中。当每个准则组合完成后，可操作单个的对象集合。

对变量**myCollection**添加监控后的显示如右图所示：



下面是实现相同任务的另一方法。我们将建立一个命名的组并把找到的对象加入到命名的组中。

```
Sub TestScanFilterM()
    Dim myEnum As ElementEnumerator
    Dim myFilter As New ElementScanCriteria
    Dim myGroup As NamedGroupElement
    Set myGroup = ActiveModelReference.AddNewNamedGroup("GroupA")
    myFilter.ExcludeAllTypes
    myFilter.ExcludeAllLevels
    myFilter.IncludeType msdElementTypeSharedCell
    myFilter.IncludeLevel ActiveDesignFile.Levels("COLUMNS")
    Set myEnum = ActiveModelReference.Scan(myFilter)
    While myEnum.MoveNext
        myGroup.AddMember myEnum.Current
    Wend
    myFilter.Reset
    myFilter.ExcludeAllTypes
    myFilter.ExcludeAllLevels
    myFilter.IncludeType msdElementTypeText
    myFilter.IncludeLevel ActiveDesignFile.Levels("MARKS")
    Set myEnum = ActiveModelReference.Scan(myFilter)
    While myEnum.MoveNext
        myGroup.AddMember myEnum.Current
    Wend
    myGroup.Rewrite
    MsgBox myGroup.MembersCount & " elements found."
End Sub
```



回顾三种集合方法

上述三种方法各有其优缺点。为了便于讨论，我们把这些方法分别叫做多准则、集合和组。

多准则方法直接而简单，在其独立的枚举器中取得符合准则的元素组。这样做的一个好处是，我们具有独立的准则组，区分很明显，在需要时可分别操作每个组。根本缺点是，这些独立的组使得我们很难把每个组中的元素作为一个整体来操作。

集合方法只用了一个 ScanCriteria 对象，并把找到的所有对象都放入了一个单独的自定义 VBA 集合中。这样做就能在整个集合上应用 For Each ... Next 语句、从集合中移除项等等。

组方法能提供最好的结果。每一项都放入到了一个容器中，这与使用集合具有相同的好处。使用组的实际好处是，当用“myGroup.Rewrite”将组添加到设计文件中后，这个组就能被其它标准的 MicroStation 命令和功能所利用。如果不“重写”组，即使组本身在设计文件中，添加到组中的元素也不会被添加到设计文件的组中去。所以，假如想使用组而不把它重写如设计文件，在完成程序任务后应该删除这个组。

扫描准则的方法

本章中所编写的代码已经使用了 ElementScanCriteria 对象的几个方法。下面是扫描准则对象方法的完整列表：

Sub ExcludeAllClasses()

Sub ExcludeAllColors()

Sub ExcludeAllLevels()

Sub ExcludeAllLineStyles()

Sub ExcludeAllLineWeights()

Sub ExcludeAllSubtypes()

Sub ExcludeAllTypes()

Sub ExcludeGraphical()

Sub ExcludeNonGraphical()

Sub IncludeClass(ElemClass As MsdElementClass)

Sub IncludeColor(ColorIndex As Long)

Sub IncludeLevel(Level As Level)

Sub IncludeLineStyle(LineStyle As LineStyle)

Sub IncludeLineWeight(LineWeight As Long)

Sub IncludeOnlyCell(CellName As String)

Sub IncludeOnlyFilePositionRange(Min As Long, Max As Long)

Sub IncludeOnlyGraphicGroup(GraphicGroupNumber As Long)

Sub IncludeOnlyHole()

Sub IncludeOnlyInvisible()

Sub IncludeOnlyLocked()

Sub IncludeOnlyModified()

Sub IncludeOnlyModifiedRange(Min As Date, [Max As Date])

Sub IncludeOnlyNew()

Sub IncludeOnlyNonPlanar()

Sub IncludeOnlyNonSnappable()

Sub IncludeOnlyOld()

Sub IncludeOnlyPlanar()

Sub IncludeOnlySnappable()

Sub IncludeOnlySolid()

Sub IncludeOnlyUnlocked()

Sub IncludeOnlyUnmodified()

Sub IncludeOnlyUserAttribute(UserAttributeID As Long)

Sub IncludeOnlyVisible()

Sub IncludeOnlyWithinRange(Range As Range3d)

Sub IncludeSubtype(Long As Long)

Sub IncludeType(Type As MsdElementType)

Sub Reset()

查看 MicroStation VBA 帮助文件能解释任何非自解释的方法。有一个方法

China BDN

值得注意：““IncludeOnlyWithinRange”方法。

```
Sub TestScanFilterN()
    Dim myEnum As ElementEnumerator
    Dim myFilter As New ElementScanCriteria
    Dim myGroup As NamedGroupElement
    Dim myRange As Range3d
    Set myGroup = ActiveModelReference.AddNewNamedGroup("GroupC")
    myRange.Low.X = 1: myRange.Low.Y = 1: myRange.Low.Z = 0
    myRange.High.X = 3: myRange.High.Y = 3: myRange.High.Z = 0
    myFilter.IncludeOnlyWithinRange myRange
    Set myEnum = ActiveModelReference.Scan(myFilter)
    While myEnum.MoveNext
        myGroup.AddMember myEnum.Current
    Wend
    myGroup.Rewrite
    MsgBox myGroup.MembersCount & " elements found."
End Sub
```

这种仅在特定的区域内扫描文件的功能非常强大。例如，可以查找用户选定的点周围的元素，或者可以扫描具有特定名字的单元周围的元素。指定的范围是三维的，所以如果工作在 3D 文件中，可以提供 Low.Z 和 High.Z 值。

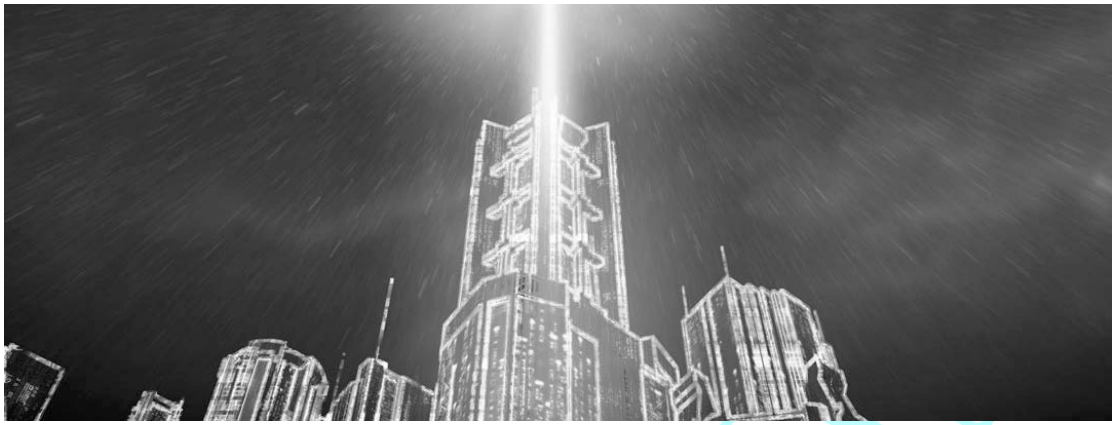
本章回顾

MicroStation 中的每个文件都有许多对象组成。有些对象是可见的，其它的则是不可见的。例如，层是非图形元素但仍然非常重要。

在扫描文件时应当小心。如果扫描的目的是要建立新的几何体，就很有可能会出现问题。例如，假设在文件中扫描线并在旧线上绘制新线，新绘制的线可能又被加入到扫描准则中，这样就陷入到了一个永无止境的死循环中。

本章讲述了用预定义的准则来扫描 MicroStation 文件。下一章中，用户在 MicroStation 中构造选择集，我们用代码操作这个选择集。

China BDN



17. 交互式修改

当用 VBA 修改文件和元素时，用户与 MicroStation 间的交互是很有用的。设计良好的程序，交互能力应该是强大而又灵活的。

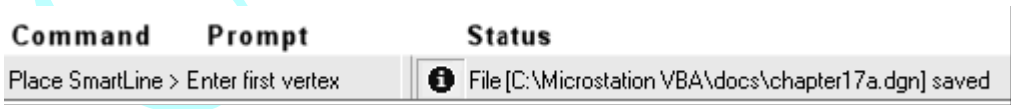
本章将讨论：

- 向用户提供反馈和信息
- 操作选择集
- 取得用户输入
- 使用 SendCommand
- 使用非模态对话框
- 一些实际应用程序
- 与 MDL 应用交互

向用户提供反馈和信息

无论您以什么身份来操作 MicroStation，MicroStation 窗口底部的三个区域都给我们以信息和反馈。

这些区域叫做命令区(Command)、提示区(Prompt)和状态区(Status)。

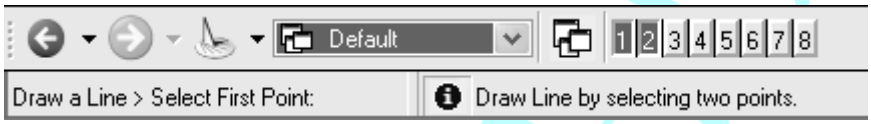


当启动“Place SmartLine”命令时，能看到该命令以及和该命令相关的第一个提示，这个提示是“Enter first vertex(输入第一个顶点)”。在点击了第一个顶点后，又被提示“Enter next vertex or reset to complete(输入下一个顶点或按

复位键完成)”。状态区给出选择或其他命令结果的一般性反馈。

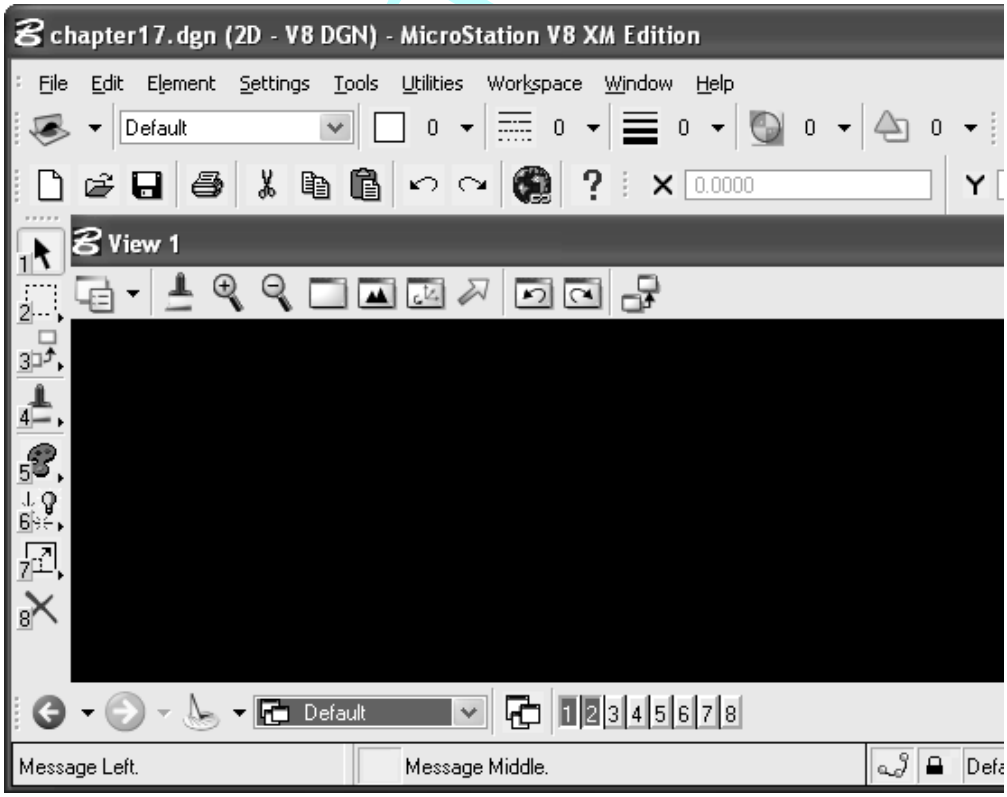
让我们看一下当用户使用我们的程序时，我们是如何在这些区域中给用户以相似的反馈和信息的。

```
Sub TestShowCommand()
    ShowCommand    "Draw a Line"
    ShowPrompt    "Select First Point:"
    ShowStatus    "Draw Line by selecting two points."
End Sub
```



三个方法用于在 MicroStation 的命令区、提示区和状态区显示文本。即使用户能改变命令/提示区的大小，也要保证不需用户把该区域拉得更大就能看到命令和提示。命令和提示不是用来提供完整的操作指南而是提供一般性的指导。

```
Sub TestShowTempMessage()
    ShowTempMessage msdStatusBarAreaLeft, “Message Left.”
    ShowTempMessage msdStatusBarAreaMiddle, “Message Middle.”
End Sub
```

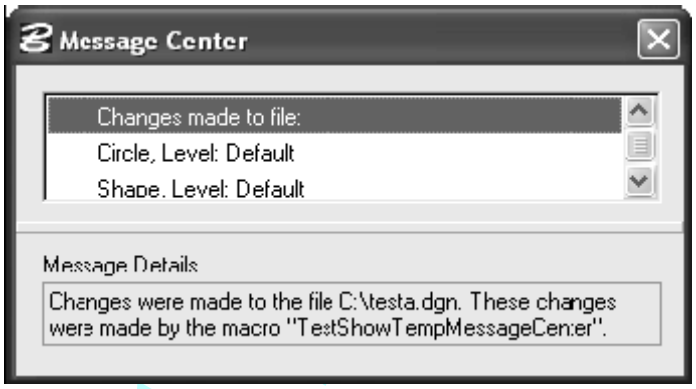


向用户提供反馈的另一种方式是发送“临时消息”。可以在“左侧区”或“中间区”放置消息，放在中间区的消息也会出现在消息中心(Message Center)。

```
Sub ShowTempMessage(Area As MsdStatusBarArea,
Message As String, [Details As String])
```

这是 ShowTempMessage 方法的声明，它有一个可选参数“Details(详细)”。当给这个参数提供一个值并且消息位置指定为“msdStatusBarAreaMiddle”时，所给的详细消息会显示在消息中心。这是向用户提供更长消息的一个最佳途径。

```
Sub TestShowTempMessageCenter()
    ShowTempMessage msdStatusBarAreaMiddle, "Changes made to file:", _
            "Changes were made to the file C:\testa.dgn. " & _
            "These changes were made by the macro " & _
            """TestShowTempMessageCenter""."
End Sub
```



下一个要考察的反馈信息的方法是 ShowError 方法。给该方法提供的文本也显示在命令/提示区。

```
Sub TestShowError()
    ShowError "Selection of Cell Failed."
End Sub
```



操作选择集

用户能通过不同的方法在其文件中选择元素。一旦元素被选定，我们能用 GetSelectedElements 方法对所选元素进行修改。

```
Sub TestSelectionSetA()
    Dim myElement As Element
    Dim myElemEnum As ElementEnumerator
    Set myElemEnum = ActiveModelReference.GetSelectedElements
    While myElemEnum.MoveNext
        Set myElement = myElemEnum.Current
        myElement.Level = ActiveModelReference.Levels("A-FURN-FREE")
    Wend
End Sub
```

前一章使用过 ElementEnumerator，本章用它取得选择的元素然后逐个修改每个元素的层。请仔细分析以上代码，遗漏了什么东西了吗？

```
Sub TestSelectionSetB()
    Dim myElement As Element
    Dim myElemEnum As ElementEnumerator
    Set myElemEnum = ActiveModelReference.GetSelectedElements
    While myElemEnum.MoveNext
        Set myElement = myElemEnum.Current
        myElement.Level = ActiveModelReference.Levels("A-FURN-FREE")
        myElement.Rewrite
    Wend
End Sub
```

如果不把元素重写入设计文件，对元素的修改不能保持住。这一点非常重要！你可能花了大量时间来调试代码，但只发现对元素作的修改没有反映到文件中。任何对文件中元素的修改必须重写回文件，否则这些元素将不能持久。MicroStation 有意这样做的。

```
Sub TestSelectionSetC()
    Dim mySettings As Settings
    Set mySettings = Application.ActiveSettings
    If MsgBox("Change Selection to Color " & mySettings.Color & "?", _
                                        vbYesNo) = vbYes Then
```

```
        Dim myElement As Element
        Dim myElemEnum As ElementEnumerator
        Set myElemEnum = ActiveModelReference.GetSelectedElements
        While myElemEnum.MoveNext
            Set myElement = myElemEnum.Current
            myElement.Color = mySettings.Color
            myElement.Rewrite
        Wend
    End If
End Sub
```



如果用户在右图所示消息框中点击 Yes 按钮，TestSelectionSetC 将把所有选择的元素改为 MicroStation 中的当前颜色。我们使用相同的方法学来遍历 ElementEnumerator 中的元素。

取得用户输入

到目前为止，我们已经讨论过向用户提示信息和操作已选元素。在过程执行时让用户输入能使交互修改的功能更加强大。

CADInputQueue(CAD 输入队列)能捕获某些用户与 MicroStation 间的交互。让我们看几个使用 CADInputQueue 的例子。先从演示如何使用 CADInputQueue 的一个非常简单的例子开始，然后再转到一些现实世界中的例子。

```
Sub TestCadInputA()
    Dim myCIQ As CadInputQueue
    Dim myCIM As CadInputMessage
    Dim I As Long
    Set myCIQ = CadInputQueue
    For I = 1 To 10
        Set myCIM = myCIQ.GetInput
        Debug.Print myCIM.InputType
    Next I
```

China BDN

```
End Sub
```

上例捕获十个用户交互并把其输入类型(InputType)打印到立即窗口中。该例主要想看的是 CadInputQueue 和 CadInputMessage 的工作机制。

我们对上例作几处改动来只捕获对点的选择。

```
Sub TestCadInputB()
    Dim myCIQ As CadInputQueue
    Dim myCIM As CadInputMessage
    Dim I As Long
    Dim pt3Selection As Point3d
    Set myCIQ = CadInputQueue
    For I = 1 To 10
        Set myCIM = myCIQ.GetInput(msdCadInputTypeDataPoint)
        pt3Selection = myCIM.Point
        Debug.Print pt3Selection.X & ", " & pt3Selection.Y
    Next I
End Sub
```

CADInputQueue 捕获几种不同的输入类型，在使用 GetInput 方法时可以指定要捕获的输入类型。我们把上例限制到了对数据点的捕获。由于知道将要取得一个点，所以就能使用 CADInputMessage 对象的 Point(点)属性并把其 X、Y 和 Z 分量打印到调试窗口(也就是立即窗口)。

让我们继续构建“TestCadInput”宏，下个例子将捕获点和复位键。

```
Sub TestCadInputC()
    Dim myCIQ As CadInputQueue
    Dim myCIM As CadInputMessage
    Dim I As Long
    Dim pt3Selection As Point3d
    Set myCIQ = CadInputQueue
    For I = 1 To 10
        Set myCIM = myCIQ.GetInput(msdCadInputTypeDataPoint, _
                                   msdCadInputTypeReset)
```


China BDN

```
        Select Case myCIM.InputType
            Case msdCadInputTypeDataPoint
                pt3Selection = myCIM.Point
                Debug.Print pt3Selection.X & ", " & pt3Selection.Y
            Case msdCadInputTypeReset
                Exit For
        End Select
    Next I
End Sub
```

现在，这个宏捕获最多十个输入点或直到用户按了复位键。当程序监测到一个复位键时，用 Exit For 退出循环。

已经介绍了数据点和复位输入类型，那么还有其它哪些可用类型呢？

msdCadInputTypeCommand = 1

msdCadInputTypeReset = 2

msdCadInputTypeDataPoint = 3

msdCadInputTypeKeyin = 4

msdCadInputTypeAny = 5

msdCadInputTypeUnassignedCB = 6

当用 CADInputQueue 开始捕获输入时，我们的程序监听每一个输入，所有的输入结果都由我们的程序掌控。例如，如果开始捕获输入，从工具栏中选择一个命令将把命令信息发送到我们的队列中，MicroStation 并不立即开始处理该命令。

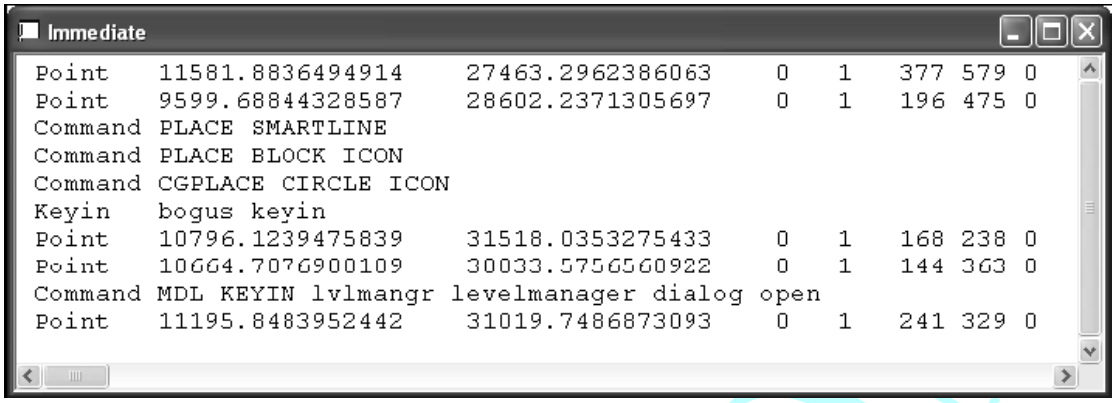
```
Sub TestCadInputD()
    Dim myCIQ As CadInputQueue
    Dim myCIM As CadInputMessage
    Dim I As Long
    Dim pt3Selection As Point3d
    Set myCIQ = CadInputQueue
    For I = 1 To 10
        Set myCIM = myCIQ.GetInput
```


China BDN

```
        Select Case myCIM.InputType
            Case msdCadInputTypeCommand
                Debug.Print "Command" & vbTab & myCIM.CommandKeyin
            Case msdCadInputTypeReset
                Exit For
            Case msdCadInputTypeDataPoint
                pt3Selection = myCIM.Point
                Debug.Print "Point" & vbTab & pt3Selection.X & vbTab & _
                    pt3Selection.Y & vbTab & _
                    pt3Selection.Z & vbTab & _
                    myCIM.View.Index & vbTab & _
                    myCIM.ScreenPoint.X & vbTab & _
                    myCIM.ScreenPoint.Y & vbTab & _
                    myCIM.ScreenPoint.Z
            Case msdCadInputTypeKeyin
                Debug.Print "Keyin" & vbTab & myCIM.Keyin
            Case msdCadInputTypeAny
                Debug.Print "Any"
            Case msdCadInputTypeUnassignedCB
                Debug.Print "UnassignedCB" & vbTab & _
                    myCIM.CursorButton
        End Select
    Next I
End Sub
```

该过程捕获十个输入或监测到一个复位键后结束。

China BDN



上图是运行该过程的结果，带有各种不同的输入。

点

在 MicroStation 中选择的点不仅仅只具有 X、Y 和 Z 位置信息，还能看到该点在哪个视图被选取以及其屏幕坐标的 X、Y 和 Z 值。屏幕坐标 X、Y 和 Z 对更进一步的工作有帮助，如，用 Windows API 在 MicroStation 中显示图形信息。

命令

无论是从菜单、工具栏或是键入窗口发出命令，只要是合法的 MicroStation 命令而我们又正在用 CAD 输入队列监听，则这些输入都会以命令的形式发送到我们的程序中。

键入

如果键入窗口用于输入一个合法的命令，则输入就被注册为命令而不是键入。当在键入窗口中输入的内容不产生一个合法的命令时，该内容就被注册为一个键入。上例中对键入的演示就是在键入窗口中输入“bogus keyin”。

未赋值的光标按钮

一个未赋值的光标按钮生成一个 UnassignedCB 输入。用消息的 CursorButton 属性来检索所使用的是哪个光标按钮。

复位

当用户发出一个复位时触发复位输入。例如，当请求选择一个点时用户点击了鼠标右键就会发出一个复位。

一些实际应用程序

既然已经明白了这些输入是如何工作的，就让我们在实际的例子中应用它们吧。

```
Sub TestCadInputE()
    Dim myCIQ As CadInputQueue
    Dim myCIM As CadInputMessage
    Dim pt3Start As Point3d
    Dim pt3End As Point3d
    Dim myLine As LineElement
    Set myCIQ = CadInputQueue
    Set myCIM = myCIQ.GetInput(msdCadInputTypeDataPoint, _
                               msdCadInputTypeReset)
    Select Case myCIM.InputType
        Case msdCadInputTypeReset
            Exit Sub
        Case msdCadInputTypeDataPoint
            pt3Start = myCIM.Point
    End Select
    Set myCIM = myCIQ.GetInput(msdCadInputTypeDataPoint, _
                               msdCadInputTypeReset)
    Select Case myCIM.InputType
        Case msdCadInputTypeReset
            Exit Sub
        Case msdCadInputTypeDataPoint
            pt3End = myCIM.Point
    End Select
    Set myLine = CreateLineElement2(Nothing, pt3Start, pt3End)
    ActiveModelReference.AddElement myLine
    myLine.Redraw
End Sub
```

TestCadInputE 让用户选择两个点，随后在这两个点间绘制一条线。仔细检查一下代码，还好；运行代码后发现，虽然用户选择两个点后在两点间画出了一

条线，但用户无法知道在做什么或动作的结果会是什么。让我们用 ShowCommand 和 ShowPrompt 知识使该宏更加用户友好一些。

```
Sub TestCadInputF()
    Dim myCIQ As CadInputQueue
    Dim myCIM As CadInputMessage
    Dim pt3Start As Point3d
    Dim pt3End As Point3d
    Dim myLine As LineElement
    Set myCIQ = CadInputQueue
    ShowCommand "Two-Point Line"
    ShowPrompt "Select First Point:"
    Set myCIM = myCIQ.GetInput(msdCadInputTypeDataPoint, _
                               msdCadInputTypeReset)
    Select Case myCIM.InputType
        Case msdCadInputTypeReset
            ShowPrompt ""
            ShowCommand ""
            ShowStatus "Two-Point Line Reset."
            Exit Sub
        Case msdCadInputTypeDataPoint
            pt3Start = myCIM.Point
    End Select
    ShowPrompt "Select Second Point:"
    Set myCIM = myCIQ.GetInput(msdCadInputTypeDataPoint, _
                               msdCadInputTypeReset)
    Select Case myCIM.InputType
        Case msdCadInputTypeReset
            ShowPrompt ""
            ShowCommand ""
            ShowStatus "Two-Point Line Reset."
```


China BDN

```
            Exit Sub
        Case msdCadInputTypeDataPoint
            pt3End = myCIM.Point
    End Select
    Set myLine = CreateLineElement2(Nothing, pt3Start, pt3End)
    ActiveModelReference.AddElement myLine
    myLine.Redraw
    ShowPrompt ""
    ShowCommand ""
    ShowStatus "Two-Point Line Drawn."
End Sub
```

现在，当宏运行时，用户在每一步都有提示。

CADInputQueue 不仅只用于捕获用户输入，它还能被用于执行命令。下面是一个例子：

```
Sub TestCadInputH()
    Dim myCIQ As CadInputQueue
    Dim myCIM As CadInputMessage
    Dim pt3Start As Point3d
    Dim pt3End As Point3d
    Dim myLine As LineElement
    Dim SelElems() As Element
    Set myCIQ = CadInputQueue
    Set myCIM = myCIQ.GetInput(msdCadInputTypeDataPoint, _
                               msdCadInputTypeReset)
    Select Case myCIM.InputType
        Case msdCadInputTypeReset
            Exit Sub
        Case msdCadInputTypeDataPoint
            pt3Start = myCIM.point
    End Select
```


China BDN

```
    Set myCIM = myCIQ.GetInput(msdCadInputTypeDataPoint, _
                                msdCadInputTypeReset)
    Select Case myCIM.InputType
        Case msdCadInputTypeReset
            Exit Sub
        Case msdCadInputTypeDataPoint
            pt3End = myCIM.point
    End Select
    CadInputQueue.SendDragPoints pt3Start, pt3End
    SelElems = _
      ActiveModelReference.GetSelectedElements.BuildArrayFromContents
    If MsgBox("Are you sure you want to delete " & _
        UBound(SelElems) + 1 & " Elements?", vbYesNo) _
        = vbYes Then
        CadInputQueue.SendCommand "DELETE"
    End If
End Sub
```

在这个例子中，用户选择的两个点能构成一个矩形窗口，用 CADInputQueue 对象的 SendDragPoints 方法选取该窗口内的元素。取得选取的元素个数后，通过带有 Yese 和 No 按钮的消息框向用户提问是否删除这些元素。如果用户选择“Yes”，就发送一个“DELETE”命令来删除这些元素。

这个例子让用户选取两点并删除两点间窗口内的元素。但我们必须自问：这个程序工作得好吗？在选取了第一点后，我们不能看到选取点的位置。如果能看到第一个选择点就像画线时那样会更好些。

下一个函数让用户选择两个点，第一点选择后，能看到和 MicroStation 标准画线命令一样的图形界面，直到选择了第二点。该函数最后返回这两个点。

```
Function PointsByLine() As Point3d()
    Dim myCIQ As CadInputQueue
    Dim myCIM As CadInputMessage
    Dim pt3Start As Point3d
```

```
    Dim pt3End As Point3d
    Dim selPts(0 To 1) As Point3d
    Set myCIQ = CadInputQueue
    Set myCIM = myCIQ.GetInput(msdCadInputTypeDataPoint, _
                               msdCadInputTypeReset)
    Select Case myCIM.InputType
        Case msdCadInputTypeReset
            Err.Raise -12345
            Exit Function
        Case msdCadInputTypeDataPoint
            pt3Start = myCIM.point
    End Select
    CadInputQueue.SendCommand "PLACE LINE"
    CadInputQueue.SendDataPoint pt3Start
    Set myCIM = myCIQ.GetInput(msdCadInputTypeDataPoint, _
                               msdCadInputTypeReset)
    Select Case myCIM.InputType
        Case msdCadInputTypeReset
            Err.Raise -12346
            Exit Function
        Case msdCadInputTypeDataPoint
            pt3End = myCIM.point
    End Select
    selPts(0) = pt3Start
    selPts(1) = pt3End
    PointsByLine = selPts
End Function
```

在用户选择了第一点后，开始“PLACE LINE”命令并将用户选择的点提供给该命令。这样做能建立橡皮筋效果使我们能看到选择的第一点，在等待选择第

China BDN

二点时也能显示光标的坐标。第二点选择后，把选择的点放入一个数组中作为该函数的返回值。如果用户在输入第一点或第二点时发出了一个复位，本函数将激发一个错误，这样调用“PointsByLine”的函数或过程就知道发生了什么事情。我们需要记住，当退出该函数时“PLACE LINE”命令仍在处理中，应该在调用的过程或函数中如下所示处理它：

```
Sub TestCadInputJ()
    On Error GoTo errhnd
    Dim selPts() As Point3d
    selPts = PointsByLine
    CadInputQueue.SendReset
    CommandState.StartDefaultCommand
    Debug.Print selPts(0).X & ", " & selPts(0).Y & ", " & selPts(0).Z
    Debug.Print selPts(1).X & ", " & selPts(1).Y & ", " & selPts(1).Z
    Exit Sub
errhnd:
    CadInputQueue.SendReset
    CommandState.StartDefaultCommand
    Select Case Err.Number
        Case -12345
            '未选择起点
            MsgBox "Start Point not selected.", vbCritical
        Case -12346
            '未选择终点
            MsgBox "End Point not selected.", vbCritical
    End Select
End Sub
```

我们使用 PointsByLine 取得两个点。注意对 SendReset 和 StartDefaultCommand 的调用，当我们调用“PointsByLine”时这两个调用复位 PLACE LINE 命令。如果用户按要求选择了两个点，将在立即窗口中显示点的坐

标；如果用户没有选择其中的一个点，基于 PointsByLIne 函数激发的错误，我们能知道用户放弃对哪个点的选择。

下面是一个新的调用 PointsByLine 函数的更实用的应用程序：

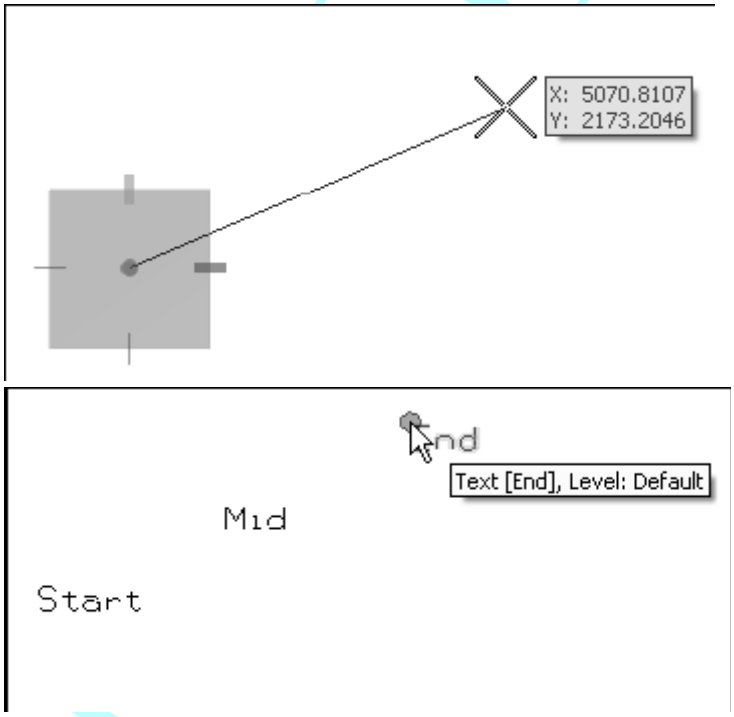
```
Sub TestCadInputK()
    On Error GoTo errhnd
    Dim selPts() As Point3d
    Dim pt3TextPt As Point3d
    Dim myText As TextElement
    Dim rotMatrix As Matrix3d
    selPts = PointsByLine
    CadInputQueue.SendReset
    CommandState.StartDefaultCommand
    Set myText = CreateTextElement1(Nothing, “Start”, selPts(0), rotMatrix)
    ActiveModelReference.AddElement myText
    Set myText = CreateTextElement1(Nothing, “End”, selPts(1), rotMatrix)
    ActiveModelReference.AddElement myText
    pt3TextPt.X = selPts(0).X + (selPts(1).X – selPts(0).X) / 2
    pt3TextPt.Y = selPts(0).Y + (selPts(1).Y – selPts(0).Y) / 2
    pt3TextPt.Z = selPts(0).Z + (selPts(1).Z – selPts(0).Z) / 2
    Set myText = CreateTextElement1(Nothing, “Mid”, pt3TextPt, rotMatrix)
    ActiveModelReference.AddElement myText
    Exit Sub
errhnd:
    CadInputQueue.SendReset
    CommandState.StartDefaultCommand
    Select Case Err.Number
        Case -12345
            ‘Start Point not selected
            MsgBox “Start Point not selected.”, vbCritical
        Case -12346
```


China BDN

```
        'End Point not selected
        MsgBox "End Point not selected.", vbCritical
    End Select
End Sub
```

该过程的架构与前一个例子相同。在模拟 PLACE LINE 命令时，使用新的 PointsByLine 函数取得两个点。尔后，用这两个点在文件中放置三个新的文本元素，“Start”、“End”和“Mid”分别放置在起点、终点和计算出的中点上。

右图是在 MicroStation 中看到的样子：



下面是另一个函数，它模拟“PLACE BLOCK”命令，能让用户拉伸一个矩形。

```
Function PointsByRectangle() As Point3d()
    Dim myCIQ As CadInputQueue
    Dim myCIM As CadInputMessage
    Dim pt3Start As Point3d
    Dim pt3End As Point3d
    Dim selPts(0 To 1) As Point3d
    Set myCIQ = CadInputQueue
    Set myCIM = myCIQ.GetInput(msdCadInputTypeDataPoint, _
                               msdCadInputTypeReset)
```

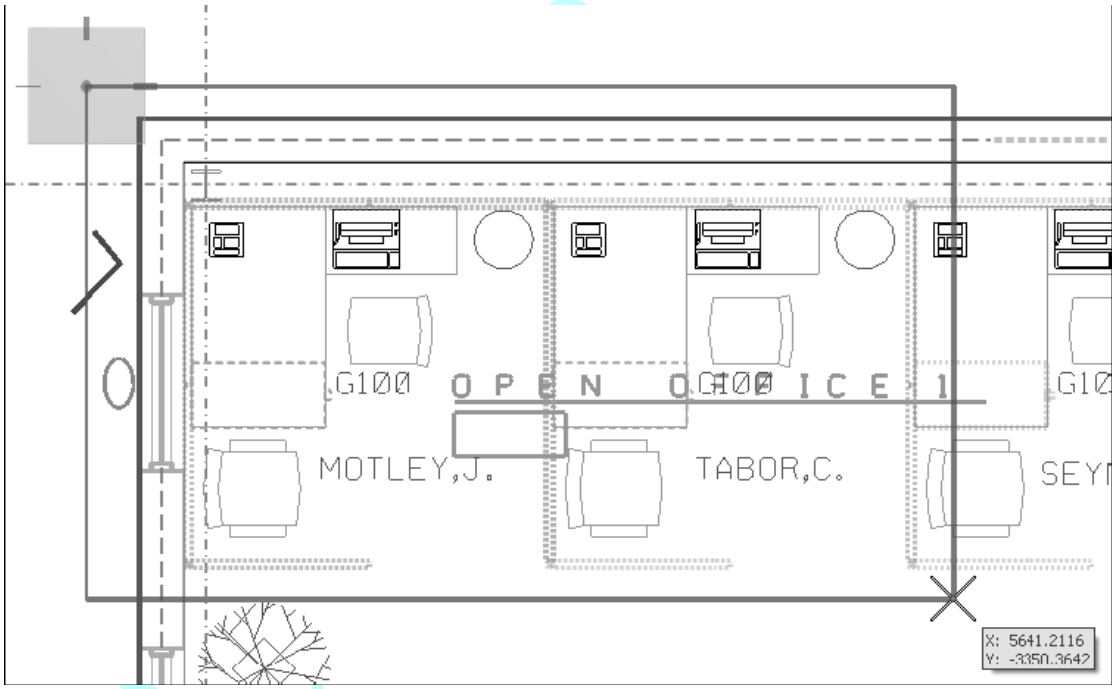
```
    Select Case myCIM.InputType
        Case msdCadInputTypeReset
            Err.Raise -12345
            Exit Function
        Case msdCadInputTypeDataPoint
            pt3Start = myCIM.point
    End Select
    CadInputQueue.SendCommand "PLACE BLOCK"
    CadInputQueue.SendDataPoint pt3Start
    Set myCIM = myCIQ.GetInput(msdCadInputTypeDataPoint, _
                               msdCadInputTypeReset)
    Select Case myCIM.InputType
        Case msdCadInputTypeReset
    Err.Raise -12346
    Exit Function
    Case msdCadInputTypeDataPoint
    pt3End = myCIM.point
    End Select
    selPts(0) = pt3Start
    selPts(1) = pt3End
    PointsByRectangle = selPts
End Function
```

下面是使用 PointsByRectangle 的过程：

```
Sub TestCadInputL()
    On Error GoTo errhnd
    Dim selPts() As Point3d
    selPts = PointsByRectangle
    CadInputQueue.SendReset
    CommandState.StartDefaultCommand
    Debug.Print selPts(0).X & ", " & selPts(0).Y & ", " & selPts(0).Z
```


China BDN

```
        Debug.Print selPts(1).X & ", " & selPts(1).Y & ", " & selPts(1).Z
        Exit Sub
errhnd:
        CadInputQueue.SendReset
        CommandState.StartDefaultCommand
        Select Case Err.Number
            Case -12345
                'Start Point not selected
                MsgBox "Start Point not selected.", vbCritical
            Case -12346
                'End Point not selected
                MsgBox "End Point not selected.", vbCritical
        End Select
End Sub
```



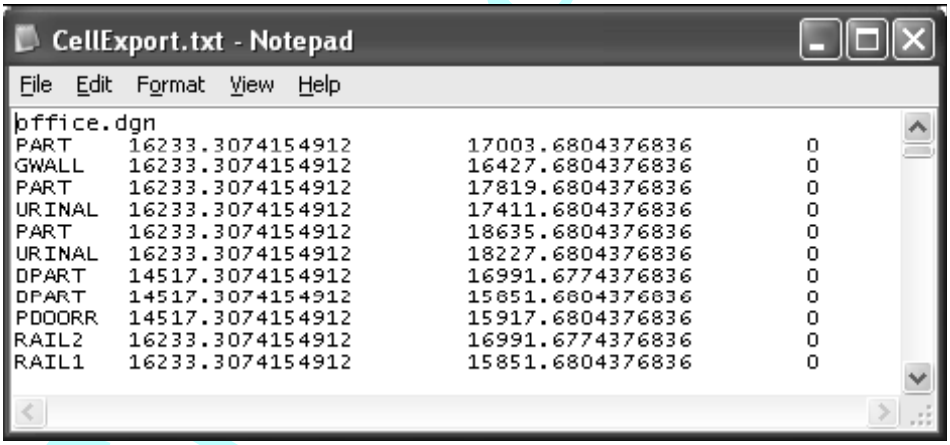
TestCadInputL 没有做任何有趣的事情，它仅在立即窗口中显示选择的点。让我们更好地使用 PointsByRectangle，下面的例子中用选择的点构成的矩形作为在文件中选择单元的扫描准则。

```
Sub TestCadInputM()
    On Error GoTo errhnd
    Dim selPts() As Point3d
    Dim LinePts(0 To 1) As Point3d
    Dim LineElem As LineElement
    Dim myESC As New ElementScanCriteria
    Dim myRange As Range3d
    Dim myElemEnum As ElementEnumerator
    Dim myElem As Element
    Dim FFile As Long
    Dim myCellHeader As CellElement
    selPts = PointsByRectangle
    CadInputQueue.SendReset
    CommandState.StartDefaultCommand
    myRange = Range3dFromPoint3dPoint3d(selPts(0), selPts(1))
    myESC.ExcludeAllTypes
    myESC.IncludeType msdElementTypeCellHeader
    myESC.IncludeOnlyWithinRange myRange
    Set myElemEnum = ActiveModelReference.Scan(myESC)
    FFile = FreeFile
    Open "C:\MicroStation VBA\CellExport.txt" For Output As #FFile
    Print #FFile, ActiveDesignFile.Name
    While myElemEnum.MoveNext
        Set myElem = myElemEnum.Current
        Set myCellHeader = myElem
        Print #FFile, myCellHeader.Name & vbTab & _
                    myCellHeader.Origin.X & vbTab & _
                    myCellHeader.Origin.Y & vbTab & _
                    myCellHeader.Origin.Z
    Wend
```

```
    Close #FFile
    Exit Sub
errhnd:
    CadInputQueue.SendReset
    CommandState.StartDefaultCommand
    Select Case Err.Number
        Case -12345
            'Start Point not selected
            MsgBox "Start Point not selected.", vbCritical
        Case -12346
            'End Point not selected
            MsgBox "End Point not selected.", vbCritical
    End Select
End Sub
```

该宏写出了在选择矩形内的当前模型参考中单元的名字和位置。

下图是随 MicroStation 安装的 office.dgn 文件的输出。



该宏的结果会随文件的不同而不同，也会随选择的不同而不同。如果矩形框选中的单元较少，输出到文本文件中的单元也就较少。

使用 SendCommand

到目前为止，我们已经用 SendCoomand 发出了命令“DELETE”、“PLACE LINE”和“PLACE BLOCK”。虽然这些命令对有些读者来说可能很熟悉，但对其它读者来说却较生疏。每当选取菜单条目或点击工具栏按钮时，都会有一个命

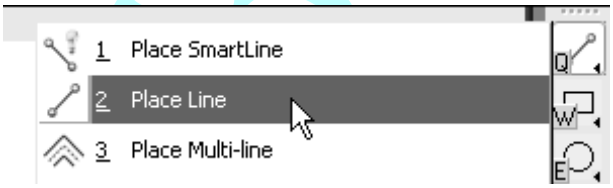
令发送给 MicroStation。我们如何能知道发送了什么命令呢？好问题！

MicroStation VBA 宏记录器能帮助我们发现命令名以及如何使用它们。让我们试着记录几个宏来演示这个过程。

1、从 VBA 工程管理器中选择我们当前使用的 VBA 工程，然后点击记录按钮；

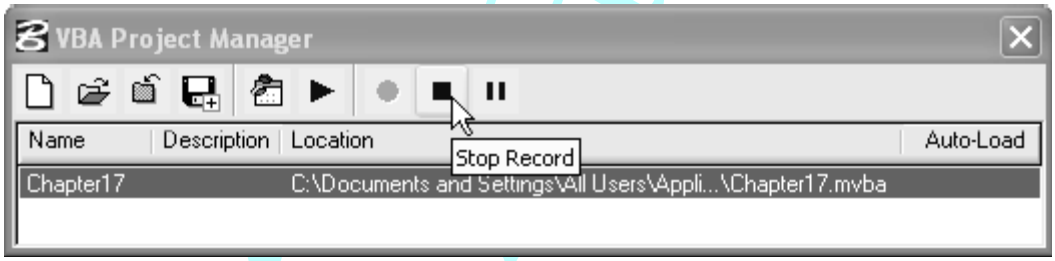


2、现在，从工具栏中选择线命令，如右图所示：



3、“Place Line”开始提问绘制线的点。在 MicroStation 中选择两个点，然后点击鼠标右键发出一个复位；

4、接下来点击“Stop Record(停止记录)”按钮来停止记录宏；



5、宏记录停止后，返回到 VBA 环境来看一看新的宏。宏是被自动命名的，其名字可能随计算机的不同而不同。

这个记录的宏看起来应该类似下面的这一段：

```
Sub Macro1()
    Dim startPoint As Point3d
    Dim point As Point3d, point2 As Point3d
    Dim lngTemp As Long
    'Start a command
    CadInputQueue.SendCommand "CGPLACE LINE CONSTRAINED "
    'Coordinates are in master units
```

```
    startPoint.X = 16735.231975
    startPoint.Y = 33020.733029
    startPoint.Z = 0#
    'Send a data point to the current command
    point.X = startPoint.X
    point.Y = startPoint.Y
    point.Z = startPoint.Z
    CadInputQueue.SendDataPoint point, 1
    point.X = startPoint.X + 1985.401024
    point.Y = startPoint.Y - 610.892623
    point.Z = startPoint.Z
    CadInputQueue.SendDataPoint point, 1
    'Send a reset to the current command
    CadInputQueue.SendReset
    CommandState.StartDefaultCommand
End Sub
```

该记录的宏揭示了一个命令“CGPLACE LINE CONSTRAINED”。所示的坐标是宏被记录时在 MicroStation 中选择的点坐标。让我们复制并粘贴这个记录的宏，再如下进行修改：

```
Sub Macro1_modifiedA()
    Dim point As Point3d
    CadInputQueue.SendCommand "CGPLACE LINE CONSTRAINED "
    point.X = 0: point.Y = 0: point.Z = 0
    CadInputQueue.SendDataPoint    point, 1
    point.X = 4: point.Y = 5: point.Z = 6
    CadInputQueue.SendDataPoint    point, 1
    CadInputQueue.SendReset
    CommandState.StartDefaultCommand
End Sub
```

China BDN

现在已经将该宏剥去杂项而只留下最本质的东西。线的坐标被替换为(0,0,0)和(4,5,6)。

让我们再记录另一个宏。这次将记录绘制一个方块(矩形)。

```
Sub Macro2()
    Dim startPoint As Point3d
    Dim point As Point3d, point2 As Point3d
    Dim lngTemp As Long
    ' Start a command
    CadInputQueue.SendCommand "PLACE BLOCK ICON "
    ' Coordinates are in master units
    startPoint.X = 3.196418
    startPoint.Y = 6.071205
    startPoint.Z = 0#
    ' Send a data point to the current command
    point.X = startPoint.X
    point.Y = startPoint.Y
    point.Z = startPoint.Z
    CadInputQueue.SendDataPoint point, 1
    point.X = startPoint.X + 2.537984
    point.Y = startPoint.Y - 0.882104
    point.Z = startPoint.Z
    CadInputQueue.SendDataPoint point, 1
    CommandState.StartDefaultCommand
End Sub
```

下面是去除了无用的内容的 Macro2 的修改版本。

```
Sub Macro2_modifiedA()
    Dim point As Point3d
    CadInputQueue.SendCommand "PLACE BLOCK ICON "
    point.X = 0
    point.Y = 0
```


China BDN

```
    point.Z = 0
    CadInputQueue.SendDataPoint point, 1
    point.X = point.X + 2.5
    point.Y = point.Y - 0.75
    CadInputQueue.SendDataPoint point, 1
    CommandState.StartDefaultCommand
End Sub
```

本例中的第二点是基于第一点产生的。第二点相对于第一点，而不是输入硬编码的坐标。然而，即使第二点的位置是相对于第一点的，但第一点仍是硬编码的。让我们再做些小改动。

```
Sub Macro2_modifiedB()
    Dim point As Point3d
    Dim myCIQ As CadInputQueue
    Dim myCIM As CadInputMessage
    Set myCIQ = CadInputQueue
    Set myCIM = myCIQ.GetInput(msdCadInputTypeDataPoint)
    point = myCIM.point
    CadInputQueue.SendCommand "PLACE BLOCK ICON "
    CadInputQueue.SendDataPoint point, 1
    point.X = point.X + 2.5
    point.Y = point.Y - 0.75
    CadInputQueue.SendDataPoint point, 1
    CommandState.StartDefaultCommand
End Sub
```

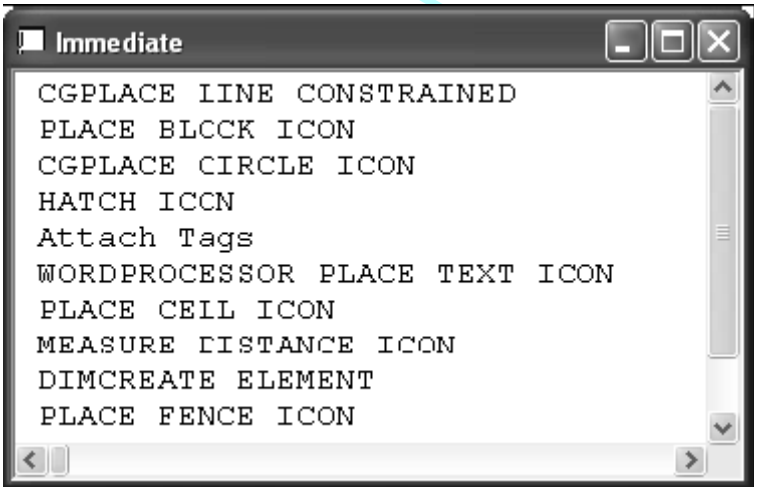
现在，用于生成方块的第一点完全基于用户输入，第二点仍然是相对于第一点。

记录宏是发现 MicroStation 命令名的一种方法。下面的宏是另一种方法。

```
    Sub TestCadInputN()
        Dim myCIQ As CadInputQueue
        Dim myCIM As CadInputMessage
```

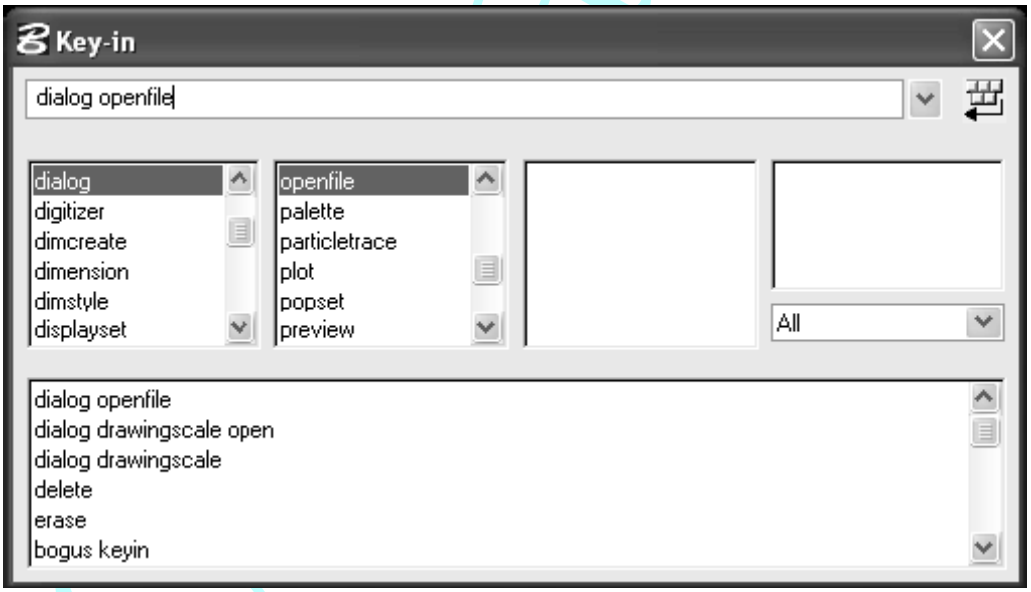

China BDN

```
    Dim I As Long
    Set myCIQ = CadInputQueue
    For I = 1 To 10
        Set myCIM = myCIQ.GetInput(msdCadInputTypeCommand)
        Debug.Print myCIM.CommandKeyin
    Next I
End Sub
```



TestCadInputN 捕获十个命令。该宏不同于记录宏，在这里我们没有取得所有其它的输入，如点的选择等。我们捕获的惟一东西是命令名。

另一个确定命令名的方法也应该在此提及。



在 MicroStation Utilities(实用程序)菜单中选 Key-in(键入)可打开键入对话框。从列表框中选择的条目能构成适当的键入。上图所示告诉我们能把“DIALOG OPENFILE”作为一个命令。让我们试一试。

```
Sub TestMessageA()
    CadInputQueue.SendCommand "DIALOG OPENFILE"
```

End Sub

运行 TestMessageA 宏能告知我们，“DIALOG OPENFILE”确实是一个合法的命名。该命令显示打开文件对话框，用户可选择一个文件打开。

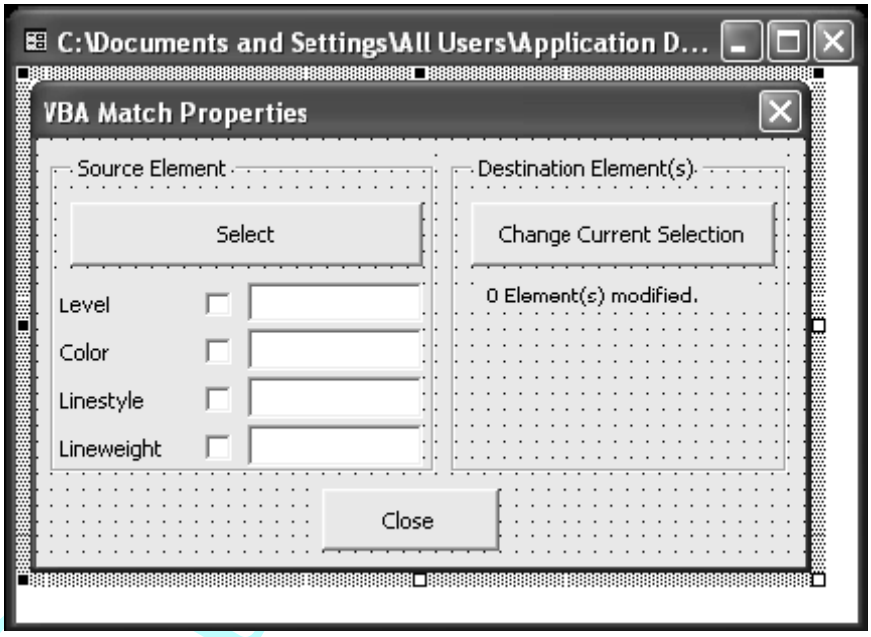
非模态对话框

输入框和消息框能让用户与代码交互，然而其功能有一定限制。当我们的目标能由一个消息框完成时就应该用它，但当需要更丰富的界面或与用户间更动态的交互时，就要使用窗体。

接下来的四个例子都在本书附带的 CD 上。每次导入一个新的窗体到当前的 VBA 工程中，方法是用 VBA 菜单 **File > Import File(文件 > 导入文件)**并从 CD 上选择适当的文件。

frmMatchProperties.frm

第一个窗体“frmMatchProperties.frm”看起来如下图所示：



该窗体看上去很简单。有几个命令按钮、两个框架、几个复选框、一个标签和四个文本框。在分析控件背后的代码前，先来讨论一下程序所期望的功能。

所期望的功能

1、用户能在 MicroStation 中选择一个源元素(Source Element)。选择元素后，Select 按钮被点击，四个属性(层、颜色、线型和线宽)从被选择的元素中提取出来；

2、用户能从源元素中选择哪些属性将在目标元素(Destination Element)中被改变；

3、用户能在 MicroStation 中选择任意数量的元素进行修改，这种修改是基于源元素中选择的属性而进行的。

这听起来很简单，让我们开始吧。尽管窗体能从 CD 上导入进来，但我们仍将讨论建立窗体的全过程。

第一件要做的事是放置控件。在制作界面时，通过适当缩放和移动控件使用户感觉界面很流畅。在添加完每个控件后可立即修改其标题(当它有标题时)。

下一步是命名控件，下面是我们将要与之交互的控件的名字：

- frmMatchProperties
- btnSelectSource
- chkLevel
- txtLevel
- chkColor
- txtColor
- chkLinestyle
- txtLinestyle
- chkLineweight
- txtLineweight
- btnChange
- lblCount
- btnClose
- fraSource
- fraDestination

在开发程序时就应该考虑程序的将来。例如，该窗体的代码未对框架作任何操作，我们可以把框架名保持为缺省的“Frame1”和“Frame2”。但是，我们可能决定在选取源元素前使目的元素框架不可见，只有在选取了源元素后才使目的元素可见。用“Frame2”作为框架名当然也能实现这一点，但在我们的代码中把它命名为了“fraDestination”。这样做能准确地知道我们正在使哪个控件可

见或不可见，而不用在窗体上浏览框架来找到想操作的框架。

控件属性

1、每个文本框的 Locked 属性应该为“True”。我们不想让用户随意地输入值，文本框的内容将由源元素的属性填充；

2、每个复选框的 Alignment 的属性应该为“fmAlignmentLeft”。这使得复选框的标题在复选框的左边；

3、Select 命令按钮的 ControlTipText 是“Click Here to make the current selection the source element”；

4、“Change Current Selection”命令按钮的 ControlTipText 是“Click Here to modify the current selection to match the selected properties from the Source element”；

5、“Close” 命令按钮的 ControlTipText 是“Click Here to Close the VBA Match Properties Program”；

下面我们将要添加代码使该窗体以非模态方式显示。非模态的意思是，即使窗体正显示着，用户也能与 MicroStation 交互。当我们分析控件背后的代码时，牢记这一点是很重要的。

通用声明区

在我们代码的通用声明区有两行代码。

```
Option Explicit
Dim elemSource As Element
```

Select(选择)按钮

源元素只能有一个。当用户点击 Select 按钮时，首先要做的事情是，考察有多少个元素被选取。如果仅选择了一个元素，就继续；否则，会显示两个消息框中的一个：其一是假如一个元素也没有选择，其二是选择了一个以上的元素。

如果仅选择了一个元素，作如下动作：

1、取得层(如果元素被赋予一个层的话)。层名放入适当的文本框中；

2、取得颜色并将颜色号显示在适当的文本框中，改变文本框的 BackColor(背景色)属性使其与源元素的颜色相匹配；

3、取得并放置线型属性；

4、取得并放置线宽属性；

现在，让我们看一看 btnSelectSource_Click 事件背后的代码：

```
Private Sub btnSelectSource_Click()
    Dim myElements() As Element
    Dim myElemEnum As ElementEnumerator
    Dim myColorTable As ColorTable
    Set myElemEnum = ActiveModelReference.GetSelectedElements
    myElements = _
    ActiveModelReference.GetSelectedElements.BuildArrayFromContents
    If UBound(myElements) = 0 Then
        Set elemSource = myElements(0)
        If Not myElements(0).Level Is Nothing Then
            txtLevel.Text = myElements(0).Level.Name
        End If
        Set myColorTable = ActiveDesignFile.ExtractColorTable
        Select Case myElements(0).Color
            Case -1
                txtColor.Text = ""
                txtColor.BackColor = RGB(255, 255, 255)
                txtLinestyle.Text = _
                    myElements(0).LineStyle.Name
                txtLineweight.Text = myElements(0).LineWeight
            Case Else
                txtColor.Text = myElements(0).Color
                txtColor.BackColor = _
                    myColorTable.GetColorAtIndex(myElements(0).Color)
                txtLinestyle.Text =
                        myElements(0).LineStyle.Name
                txtLineweight.Text = myElements(0).LineWeight
        End Select
```


China BDN

```
    Else
        Select Case UBound(myElements)
            Case -1
                MsgBox "No ""Source"" element selected.", _
                        vbCritical, Me.Caption
                Exit Sub
            Case Else
                MsgBox "Only one element can be the ""Source""" & _
                    “element.", vbCritical, Me.Caption
                Exit Sub
        End Select
    End If
End Sub
```

我们在以上代码中使用 ElementEnumerator 的方式与以前稍有不同。没有用“MoveNext”并取得当前元素，而是用 BuildArrayFromContents 取得了一个元素数组。如果该数组的上界为 0，则数组由一个元素组成，这正式我们所期望的；如果上界为-1，说明数组为空，即用户点击按钮前什么也没有选中。

> 在代码中放一个断点能让我们逐行运行代码。这有助于我们发现程序是如何运行的以及验证程序是否像设计的那样在运行。

Change Current Selection(改变当前选择集)按钮

当用户点击“Change Current Selection”按钮时，我们想把选择集中的元素的那些选择了的属性改变为源元素对应的属性。我们还改变标签控件的标题以反映有多少元素被修改。从枚举器中建立元素数组的方法和前一个例子相同。

```
Private Sub btnChange_Click()
    Dim myElements() As Element
    Dim myElemEnum As ElementEnumerator
    Dim I As Long
    Dim boolElemModified As Boolean
    Dim lngModCount As Long
```

```
lblCount.Caption = "0 Element(s) modified."
ShowStatus "0 Element(s) modified."
Set myElemEnum = ActiveModelReference.GetSelectedElements
myElements = myElemEnum.BuildArrayFromContents
lngModCount = 0
For I = LBound(myElements) To UBound(myElements)
    boolElemModified = False
    If chkLevel.Value = True Then
        myElements(I).Level = elemSource.Level
        boolElemModified = True
    End If
    If chkColor.Value = True Then
        myElements(I).Color = elemSource.Color
        boolElemModified = True
    End If
    If chkLinestyle.Value = True Then
        myElements(I).LineStyle = elemSource.LineStyle
        boolElemModified = True
    End If
    If chkLineweight.Value = True Then
        myElements(I).LineWeight = elemSource.LineWeight
        boolElemModified = True
    End If
    If boolElemModified = True Then
        myElements(I).Rewrite
        lngModCount = lngModCount + 1
    End If
Next I
lblCount.Caption = lngModCount & " Element(s) modified."
ShowStatus lngModCount & " Element(s) modified."
```

China BDN

```
End Sub
```

当在数组中查看每个元素时，我们只想按复选框的值来改变元素的属性。仅当确实对元素作出了改变后才增加元素修改计数。有这样的可能性：选择了一个源元素也选择了多个目标元素，但假如复选框都设置为假则一个元素也不会被修改。

Close(关闭)按钮

Close 按钮卸载窗体。

```
    Private Sub btnClose_Click()
        Unload Me
    End Sub
```

向用户提供反馈和信息

本章早些时候学过如何通过使用 MicroStation 的状态栏区向用户提供反馈和信息。如果仔细观察状态栏区就能看到，当在不同的工具栏按钮上移动光标时状态栏区在变化。让我们在 VBA 中模拟相同的功能，要用到几个控件的 MouseMove(鼠标移动)事件。

```
Private Sub UserForm_MouseMove(ByVal Button As Integer, _
                ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
    ShowPrompt ""
End Sub
Private Sub fraDestination_MouseMove(ByVal Button As Integer, _
                ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
    ShowPrompt ""
End Sub
Private Sub fraSource_MouseMove(ByVal Button As Integer, _
                ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
    ShowPrompt ""
End Sub
```

当用户在窗体和框架附近移动时，我们不想在提示区显示任何内容，因为在窗体或框架上点击什么事情也不会发生。所以，使用带空串的 ShowPrompt 让

China BDN

提示区什么也不显示。

```
Private Sub btnSelectSource_MouseMove(ByVal Button As Integer, _
                ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
    ShowPrompt "Select a single ""Source"" Element:"
End Sub
Private Sub btnChange_MouseMove(ByVal Button As Integer, _
                ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
    ShowPrompt "Select ""Destination"" Elements:"
End Sub
Private Sub btnClose_MouseMove(ByVal Button As Integer, _
                ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
    ShowPrompt "Close ""VBA Match Properties"""
End Sub
```

当用户将光标移动到命令按钮上时，我们想让用户知道按下按钮后将会发生什么。我们已经用每个按钮的 ControlTipText 属性实现了这个功能，但用提示区给用户提示更接近于 MicroStation 的标准功能。

用户窗体初始化(UserForm Initialize)

我们需要讨论另两个事件。第一个是用户窗体初始化事件，该事件在窗体将要显示时触发。

```
Private Sub UserForm_Initialize()
    ShowCommand "VBA Match Properties:"
End Sub
```

当窗体被首次初始化时，用 ShowCommand 将命令区设为“VBA Match Properties”。即使当光标在窗体的其它控件上移动引起提示的变化，该命令也一直显示。

用户窗体查询关闭(UserForm QueryClose)

QueryClose 事件仅在窗体终止前被触发。该事件能让我们执行清理操作，它也告知我们窗体是如何被要求关闭的。CloseMode(关闭方式)参数是下列四个值之一(它们都对应一个常数)。

- vbFormControlMenu = 0
- vbFormCode = 1
- vbAppWindows = 2
- vbAppTaskManager = 3

要了解每个值的含义的更多信息，请在 Microsoft VBA 帮助中查找“QueryCloseConstants(常数)”。

本程序中我们不关心窗体是如何被关闭的，仅仅知道其将要关闭就足够了。

```
Private Sub UserForm_QueryClose(Cancel As Integer, _
                                CloseMode As Integer)
    ShowPrompt ""
    ShowCommand ""
End Sub
```

以非模态形式显示窗体

窗体中的代码完全基于用户在 MicroStation 中对元素的选择。尽管在显示窗体前能选择元素，但实际上我们需要用户作出两次不同的选择：源和目标。源选择仅能是一个元素。为了让用户在窗体显示时仍能选择元素，需要以非模态形式显示窗体。请记住，非模态和模态是相反的，在模态情况下，当窗体聚焦时在 MicroStation 中不能作任何事情直到窗体关闭为止。非模态则意味着，即使窗体处于显示状态，MicroStation 内的交互仍能继续。为了能让窗体显示成非模态，必须在一个代码模块中的一个过程里这样显示它。

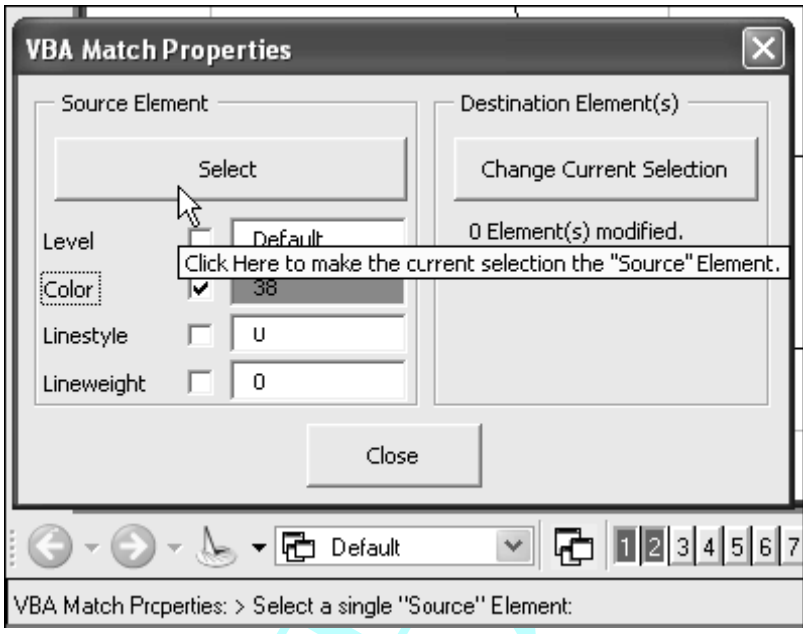
```
Sub TestMatchProperties()
    frmMatchProperties.Show vbModeless
End Sub
```

过程 TestMatchProperties 放入代码模块后用户就能够使用了，通过 VBA 工程管理器或选 MicroStation 菜单 **Utilities > Macro > Macros(实用程序 > 宏 > 宏)**或按下**<Alt>**键的同时按**<F8>(<Alt+F8>)**都能调用该过程。

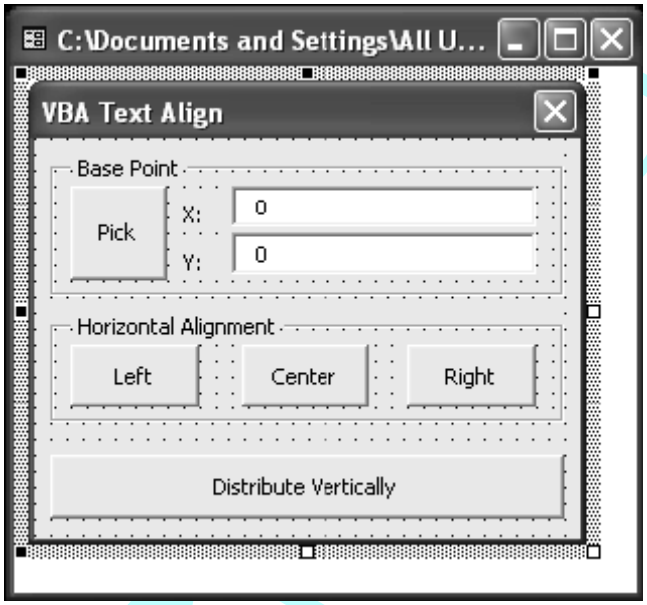
China BDN

这是使用中的匹配属性窗体。注意底部的命令和提示区。

匹配属性程序既简单又直接。它能让用户修改当前模型参考中的元素，方法是使用一个非模态对话框选择一个源元素，然后用该元素的属性改变选择的目标元素。



frmAlignText.frm



下一个将要导入到 VBA 工程中的窗体是 frmAlignText.frm 文件。该窗体能让用户对 MicroStation 中选择的文本执行对齐和分布操作。因为想让用户选择一个对齐点，所以该窗体需要以非模态显示。本程序包含几何计算并基于这些计算来移动文本元素。

所期望的功能

- 被选择的文本能在水平方向上与选取点的“X”或输入的“X”值对齐；
- 文本能左对齐、中心对齐或右对齐；
- 文本能在垂直方向上均匀分布，每个文本元素间空的距离相等；
- 只能使用文本元素，不能使用文本节点。

该工程中使用了框架、命令按钮、标签和文本框。再次说明一下，您可以从

本书附带的 CD 中导入该窗体，但我们将从头开始讨论如何建立该窗体。

放置控件

如图所示放置控件。Base Point(基准点)框架和 Horizontal Alignment(水平对齐)框架含有其各自的控件，Distribute Vertically(垂直分布)按钮是独立的。如果还存在一个“水平分布”按钮，我们将会把两个“分布”按钮放入其自己的框架内。放置完控件后，如上所示改变它们的标题和文本属性。

控件名称

- fraBasePoint
- fraHoriAlign
- btnPickBasePoint
- txtX
- txtY
- btnAlignLeft
- btnAlignCenter
- btnAlignRight
- btnDistributeVert

Pick(拾取)按钮

Pick 按钮 Click 事件中的代码让用户在 MicroStation 中选择一个点。选择的点的 X 和 Y 分量显示在文本框中。

```
Private Sub btnPickBasePoint_Click()
    Dim myCIQ As CadInputQueue
    Dim myCIM As CadInputMessage
    Set myCIQ = CadInputQueue
    Set myCIM = myCIQ.GetInput(msdCadInputTypeDataPoint, _
                    msdCadInputTypeReset)
    Do
        Select Case myCIM.InputType
            Case msdCadInputTypeDataPoint
                pt3BasePoint = myCIM.Point
                txtX.Text = pt3BasePoint.X
```


China BDN

```
                txtY.Text = pt3BasePoint.Y
                Exit Do
            Case msdCadInputTypeReset
                Exit Do
        End Select
    Loop
End Sub
```

我们使用 CadInputQueue 来捕获 MicroStation 中对点的选取。在初始化 CadInputMessage 时指定仅需要数据点和复位输入。Do ... Loop 被设计成死循环，这就意味着如果没有明确的退出语句，循环将永远继续下去。之所以使用 Do ... Loop 代码是因为我们能够随时用 Exit Do 命令退出循环。

X 和 Y 文本框

X 和 Y 文本框从用户选取的点中得到值，用户可通过刚刚讨论过的 Pick 按钮来选择点。除了能拾取点外还允许用户手工输入 X 和 Y 的值。通过拾取获得点最好，因为我们知道，用户在 MicroStation 中不可能选择一个无效的点。如果不小心，允许用户数据输入会出现问题。比如，用户在文本框中输入“somewhere around 4.5”将会发生什么？这种输入与我们预期的数字值相去甚远。限制用户在这些文本框中输入的一个方法是使用 KeyPress 事件。

```
Private Sub txtX_KeyPress(ByVal KeyAscii As MSForms.ReturnInteger)
    Select Case KeyAscii
        Case Asc("0") To Asc("9")
        Case Asc(".")
            If InStr(1, txtX.Text, ".") > 0 Then
                KeyAscii = 0
            End If
        Case Else
            KeyAscii = 0
    End Select
End Sub
```

```
Private Sub txtY_KeyPress(ByVal KeyAscii As MSForms.ReturnInteger)
    Select Case KeyAscii
        Case Asc("0") To Asc("9")
        Case Asc(".")
            If InStr(1, txtY.Text, ".") > 0 Then
                KeyAscii = 0
            End If
        Case Else
            KeyAscii = 0
    End Select
End Sub
```

KeyPress 事件能告诉我们用户按的键盘字符的 ASCII 码。如果将 KeyAscii 参数改为零值(0)，就像从未按键一样。所以，我们考察 KeyAscii 参数并问自己如下带有结果的问题：

- KeyAscii 是在 0 至 9 间的数字吗？如果是，什么也不做。总是允许输入数字 0 到 9；
- 是小数点键吗？是小数点的话就在文本框中查看是否已经存在小数点。如果文本框中已经存在小数点，设置 KeyAscii 为零，否则，什么也不做以允许小数点输入；
- Case Else(其它情况，即如果按了其它的键)，设置 KeyAscii 为零表示该键根本就没有按。

应该注意的是，这段代码仅保证从键盘只能输入数字值，它没有禁止用户从 Windows 剪切板粘贴非法输入到文本框中。

左对齐、中心对齐和右对齐

处理这三种对齐方法的最佳途径是每次一个。我们在每一个执行特定对齐类型的按钮的 Click 事件中放置代码，但这将造成大量的冗余代码。我们将建立一个函数来仔细考虑所有的水平对齐，并提供一个参数来指明要执行哪种对齐。

“对齐方式”参数可定义为字符串，使用“LEFT”、“RIGHT”或“CENTER”作为参数值。这样做是可以的，但还有一个更好的方法。

为了从 CadInputQueue 得到输入，我们要通过枚举指定输入类型，枚举是一个组合在一起的常数列表，这个列表常用于特定的方法或属性中。我们将建立自己的枚举来处理对齐，在本窗体的通用声明区声明这个枚举如下：

```
Enum AlignMode
    msvbaAlignModeLeft = 1
    msvbaAlignModeCenter = 2
    msvbaAlignModeRight = 3
End Enum
```

现在，当要声明对齐选择文本的过程时，该过程看起来像这样：

```
Sub AlignSelected(Optional ElemAlignMode As AlignMode = _
                                    msvbaAlignModeLeft)
```

把参数声明为可选，指定“Left”为缺省的对齐方式。在代码中使用这个 AlignSelected 方法时会看到：



枚举有助于保证所提供参数的合法性并使提供参数的编程更容易，因为参数选项能够显示出来。

窗体通用声明区中还需要添加另一个声明：

```
Dim pt3BasePoint As Point3d
```

用户选择一个点后，用这个变量保存选择的点。

下面是实际实现文本左对齐、中心对齐或右对齐的代码：

```
Sub AlignSelected(Optional ElemAlignMode As AlignMode = _
                                    msvbaAlignModeLeft)
    Dim myElemEnum As ElementEnumerator
    Dim myElem As Element
    Dim OriginPt As Point3d
    Dim myTextElem As TextElement
    Set myElemEnum = ActiveModelReference.GetSelectedElements
```

```
    While myElemEnum.MoveNext
        Set myElem = myElemEnum.Current
        Select Case myElem.Type
            Case msdElementTypeText
                Set myTextElem = myElem
                Select Case ElemAlignMode
                    Case msvbaAlignModeLeft
                        myTextElem.Move Point3dFromXY(pt3BasePoint.X - _
                                        myTextElem.Boundary.Low.X, 0)
                        myTextElem.Rewrite
                    Case msvbaAlignModeRight
                        myTextElem.Move Point3dFromXY(pt3BasePoint.X - _
                                        myTextElem.Boundary.High.X, 0)
                        myTextElem.Rewrite
                    Case msvbaAlignModeCenter
                        myTextElem.Move Point3dFromXY(pt3BasePoint.X - _
                                yTextElem.Boundary.Low.X - _
                                (myTextElem.Boundary.High.X - _
                                myTextElem.Boundary.Low.X) / 2, 0)
                        myTextElem.Rewrite
                End Select
        End Select
    Wend
End Sub
```

在开始执行这段代码前我们知道，用户除了选择文本元素外可能还选择了其它类型的元素。由于只想操作文本元素，所以使用了一个 Select Case 语句把文本元素从其它元素中分离了出来。接下来又使用了另一个 Select Case 语句根据指定的对齐类型和基准点的 X 值来移动文本元素。最后重写文本元素以使改变永久保存在文件中。

Left、Center 和 Right 按钮

当用户点击 Left、Center 和 Right 按钮时，相应按钮的 Click 事件被触发。注意调用“AlignSelected”时是如何使用枚举值的。

```
Private Sub btnAlignLeft_Click()
    AlignSelected msvbaAlignModeLeft
End Sub
Private Sub btnAlignCenter_Click()
    AlignSelected msvbaAlignModeCenter
End Sub
Private Sub btnAlignRight_Click()
    AlignSelected msvbaAlignModeRight
End Sub
```

复杂度

本程序有三级复杂程度。复杂度及其对应的任务如下：

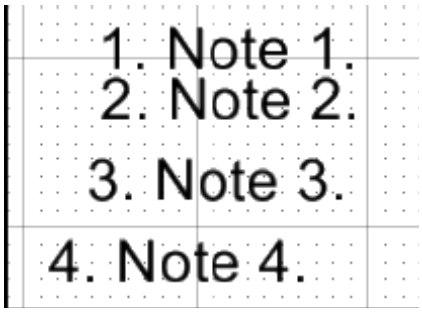
低：拾取一个点，将其 X 和 Y 分量放入文本框中；

中：将选择的文本元素左对齐、中对齐或右对齐；

高：垂直方向均匀分布选择的文本元素。

低和中复杂度的任务已经讨论过了，现在来讨论高复杂度的任务。该任务还不算太复杂因为它的难度不是非常大，只是需要许多因素才能正常工作。

垂向均匀分布选择的文本



右图是在文件 chapter17_AlignText.dgn 中的四个文本元素，该文件位于本书附带的 CD 中。已经讨论过的文本对齐代码关注的是水平对齐，现在面对的任务是使文本在垂直方向上看起来好看一些。Note 1 和 Note 2 的间距太紧，而 Note 2 与 Note 3 间又太松(松和紧都是参照 Note 3 和 Note 4 的间距来说的)。我们想让每个文本元素的间距相同。任务看起来简单但在继续前必须考虑一些问题。

1、用户选择文本后，我们想使最顶上和最底下元素间的元素均匀分布而不

想移动两端的这两个元素；

2、在屏幕上能很容易地看出哪个元素在最顶上，哪个元素在最底下，但在代码中考察选择集时，并不知道这些信息；

3、在屏幕上能看出元素的顺序，但在代码中考察选择集时，并不知道文本元素从上到下的顺序。

针对下列每个任务都将建立独立的函数来完成：

- 查找选择文本元素中的最小和最大点；
- 确定文本元素在垂直方向上出现的顺序；
- 确定选择文本元素的个数。

有了这些函数以后，我们将能用它们分布选择的文本元素。

```
Function GetMinMaxY(ElemType As Long, ElementsIn As Variant) _
                                                    As Point3d()
    Dim I As Long
    Dim pt3StartPoint As Point3d
    Dim pt3EndPoint As Point3d
    Dim myTextElem As TextElement
    Dim boolPointsSet As Boolean
    boolPointsSet = False
    For I = LBound(ElementsIn) To UBound(ElementsIn)
        Set myElem = ElementsIn(I)
        Select Case myElem.Type
            Case msdElementTypeText
                Set myTextElem = myElem
                If boolPointsSet = False Then
                    pt3StartPoint = myTextElem.Boundary.High
                    pt3EndPoint = myTextElem.Boundary.High
                    boolPointsSet = True
                End If
                If myTextElem.Boundary.High.Y > pt3StartPoint.Y Then
                    pt3StartPoint.Y = myTextElem.Boundary.High.Y
```

```
                End If
                If myTextElem.Boundary.High.Y < pt3EndPoint.Y Then
                    pt3EndPoint.Y = myTextElem.Boundary.High.Y
                End If
            End Select
        Next I
        Dim pt3Points(0 To 1) As Point3d
        pt3Points(0) = pt3StartPoint
        pt3Points(1) = pt3EndPoint
        GetMinMaxY = pt3Points
End Function
```

我们建立的这个函数能为将来所用，可扩展支持其它类型的元素。函数参数为元素类型和要处理的元素，从这些参数中找出最小和最大值并作为 Point3d 类型的数组返回。

下一项任务是按垂直方向排序元素。函数参数也是要考察的元素类型和要处理的元素，按元素在垂直方向的排序状态返回一个元素数组。

```
Function SortElementsVertically(ElemType As Long, _
                                ElementsIn As Variant) As Element()
    Dim I As Long
    Dim boolMadeChange As Boolean
    Dim lngElemID() As DLong
    Dim pt3BoundPt() As Point3d
    Dim myTextElem As TextElement
    Dim myTextElem2 As TextElement
    Dim tmpID As DLong
    Dim tmpPt As Point3d
    ReDim lngElemID(0) As DLong
    ReDim pt3BoundPt(0) As Point3d
    For I = LBound(ElementsIn) To UBound(ElementsIn)
        Select Case ElemType
```

```
            Case msdElementTypeText
                If ElementsIn(I).Type = msdElementTypeText Then
                    Set myTextElem = ElementsIn(I)
                    lngElemID(UBound(lngElemID)) = myTextElem.ID
                    pt3BoundPt(UBound(pt3BoundPt)) = _
                                        myTextElem.Boundary.High
                    ReDim Preserve lngElemID(UBound(lngElemID) + 1) As _
                        DLong
                    ReDim Preserve pt3BoundPt(UBound(pt3BoundPt) + 1) As _
                        Point3d
                End If
        End Select
    Next I
    ReDim Preserve lngElemID(UBound(lngElemID) - 1) As DLong
    boolMadeChange = True
    While boolMadeChange = True
        boolMadeChange = False
        For I = LBound(lngElemID) To UBound(lngElemID) - 1
            If pt3BoundPt(I + 1).Y > pt3BoundPt(I).Y Then
                tmpID = lngElemID(I)
                tmpPt = pt3BoundPt(I)
                lngElemID(I) = lngElemID(I + 1)
                pt3BoundPt(I) = pt3BoundPt(I + 1)
                lngElemID(I + 1) = tmpID
                pt3BoundPt(I + 1) = tmpPt
                boolMadeChange = True
            End If
        Next I
    Wend
    Dim ElemsIn() As Element
```


China BDN

```
    ReDim ElemsIn(0 To UBound(lngElemID))
    For I = LBound(lngElemID) To UBound(lngElemID)
        Set ElemsIn(I) = ActiveDesignFile.GetElementByID(lngElemID(I))
    Next I
    SortElementsVertically = ElemsIn
End Function
```

这里有太多的代码需要仔细分析。在把以上代码分成四小块以后就容易理解多了。

变量声明

```
    Dim I As Long
    Dim boolMadeChange As Boolean
    Dim lngElemID() As DLong
    Dim pt3BoundPt() As Point3d
    Dim myTextElem As TextElement
    Dim myTextElem2 As TextElement
    Dim tmpID As DLong
    Dim tmpPt As Point3d
```

两个变量被声明为动态数组(使用空括号)，动态数组能在不丢失值的前提下改变数组大小。还声明了其它一些变量。

用于冒泡排序的动态变量数组

```
    ReDim lngElemID(0) As DLong
    ReDim pt3BoundPt(0) As Point3d
    For I = LBound(ElementsIn) To UBound(ElementsIn)
        Select Case ElemType
            Case msdElementTypeText
                If ElementsIn(I).Type = msdElementTypeText Then
                    Set myTextElem = ElementsIn(I)
                    lngElemID(UBound(lngElemID)) = myTextElem.ID
                    pt3BoundPt(UBound(pt3BoundPt)) = _
                        myTextElem.Boundary.High
```

```
                ReDim Preserve _
                    lngElemID(UBound(lngElemID) + 1) As DLong
                ReDim Preserve _
                    pt3BoundPt(UBound(pt3BoundPt)+ 1) As Point3d
            End If
        End Select
    Next I
    ReDim Preserve lngElemID(UBound(lngElemID) - 1) As DLong
    ReDim Preserve pt3BoundPt(UBound(pt3BoundPt) - 1) As Point3d
```

我们查看 ElementIn 参数中的每个元素已确定其是否正确的类型(在本例中是查找文本元素)。如果是文本元素，就取得其 ID 属性放入一个动态数组变量中，同时还取得该元素的 Boundary.High 点放入另一个动态数组变量中。然后用 Preserve 关键字重新声明动态数组变量，这样能不丢失以前的值。在这段代码最后，我们重声明动态数组变量为其大小减 1。这是因为每当对数组上界的变量赋值后我们都对数组大小加 1，所以在整个赋值完成后需要拿掉一个数组元素，否则，数组中的最后一个元素里会没有值。

冒泡排序

```
    boolMadeChange = True
    While boolMadeChange = True
        boolMadeChange = False
        For I = LBound(lngElemID) To UBound(lngElemID) - 1
            If pt3BoundPt(I + 1).Y > pt3BoundPt(I).Y Then
                tmpID = lngElemID(I)
                tmpPt = pt3BoundPt(I)
                lngElemID(I) = lngElemID(I + 1)
                pt3BoundPt(I) = pt3BoundPt(I + 1)
                lngElemID(I + 1) = tmpID
                pt3BoundPt(I + 1) = tmpPt
                boolMadeChange = True
            End If
```

```
        Next I
    Wend
```

以前讨论过冒泡排序。这段程序考察两点的 Y 值，我们想让最高的 Y 值在列表的最顶上。所以，如果列表中一个靠下的 Y 值大于列表中仅靠其上的 Y 值，就交换这两个点。当发生一个交换时把变量 **boolMadeChange** 设为 True，这将意味着程序要再次遍历数组。我们一直遍历数组直到没有交换发生为止。当不再有交换发生时，排序也就完成了。

设置返回值

```
    Dim ElemsIn() As Element
    ReDim ElemsIn(0 To UBound(lngElemID))
    For I = LBound(lngElemID) To UBound(lngElemID)
        Set ElemsIn(I) = ActiveDesignFile.GetElementByID(lngElemID(I))
    Next I
    SortElementsVertically = ElemsIn
```

该函数返回一个元素数组。在进行冒泡排序时，交换点数组值的同时也交换 ID 数组值以保证 ID 始终匹配其对应的点。使用 GetElementByID 取回元素并把元素放入返回值数组。我们把 ID 以及点与元素分开，这样在排序时不需要保存占内存巨大的元素而只需较小的点和 ID 类型。

确定文本元素个数与刚讨论过的排序函数相比较为容易。我们给定要计数的元素类型、要计数的元素，返回匹配指定类型的元素个数。

```
    Function GetSelectedCount(ElemType As Long, _
        ElementsIn As Variant) As Long
        Dim I As Long
        For I = LBound(ElementsIn) To UBound(ElementsIn)
            If ElementsIn(I).Type = ElemType Then
                GetSelectedCount = GetSelectedCount + 1
            End If
        Next I
    End Function
```

以上三个函数将来都能扩展，现在不需要编写对它们进行扩展的代码。

现在需要在单个过程中利用这些函数实现“文本分布”任务。

```
Private Sub btnDistributeVert_Click()
    Dim MyPts As Variant
    Dim pt3StartPoint As Point3d
    Dim pt3EndPoint As Point3d
    Dim myElemEnum As ElementEnumerator
    Dim myElem As Element
    Dim myElems() As Element
    Dim I As Long
    Dim myTextElem As TextElement
    Dim lngSpaces As Long
    Dim dblSpacePerElement As Double
    Dim sortElems() As Element
    Set myElemEnum = ActiveModelReference.GetSelectedElements
    myElems = myElemEnum.BuildArrayFromContents
    MyPts = GetMinMaxY(msdElementTypeText, myElems)
    lngSpaces = GetSelectedCount(msdElementTypeText, myElems) - 1
    If lngSpaces > 1 Then
        dblSpacePerElement = (MyPts(0).Y - MyPts(1).Y) / lngSpaces
        sortElems = SortElementsVertically(msdElementTypeText, myElems)
        For I = LBound(sortElems) To UBound(sortElems)
            Set myTextElem = sortElems(I)
            myTextElem.Move Point3dFromXY(0, MyPts(0).Y - _
                    dblSpacePerElement * I - myTextElem.Boundary.High.Y)
            myTextElem.Rewrite
            myTextElem.Redraw
        Next I
    End If
End Sub
```

China BDN

仔细分析以上代码会发现，我们刚讨论过的三个函数(GetMinMaxY,、GetSelectedCount 和 SortElementsVertically)都用到了。在对元素按垂直方向排序后，移动它们以使其等间距。

向用户提供反馈和信息

与前一个程序的功能相似，需要向用户提供反馈和信息。

```
Private Sub UserForm_MouseMove(ByVal Button As Integer, _
        ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
    ShowStatus ""
    ShowPrompt ""
    ShowCommand Me.Caption
End Sub
Private Sub btnDistributeVert_MouseMove(ByVal Button As Integer, _
        ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
    ShowStatus ""
    ShowPrompt "Distribute Text Vertically"
    ShowCommand Me.Caption
End Sub
Private Sub fraHoriAlign_MouseMove(ByVal Button As Integer, _
        ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
    ShowStatus ""
    ShowPrompt ""
    ShowCommand Me.Caption
End Sub
Private Sub fraBasePoint_MouseMove(ByVal Button As Integer, _
        ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
    ShowStatus ""
    ShowPrompt ""
    ShowCommand Me.Caption
End Sub
Private Sub btnPickBasePoint_MouseMove(ByVal Button As Integer, _
```


China BDN

```
            ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
        ShowStatus ""
        ShowPrompt "Select Base Point:"
        ShowCommand Me.Caption
    End Sub
    Private Sub btnAlignRight_MouseMove(ByVal Button As Integer, _
            ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
        ShowStatus ""
        ShowPrompt "Align Selected Text Right as Base Point"
        ShowCommand Me.Caption
    End Sub
    Private Sub btnAlignLeft_MouseMove(ByVal Button As Integer, _
            ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
        ShowStatus ""
        ShowPrompt "Align Selected Text Left at Base Point"
        ShowCommand Me.Caption
    End Sub
    Private Sub btnAlignCenter_MouseMove(ByVal Button As Integer, _
            ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
        ShowStatus ""
        ShowPrompt "Align Selected Text Center at Base Point"
        ShowCommand Me.Caption
    End Sub
```

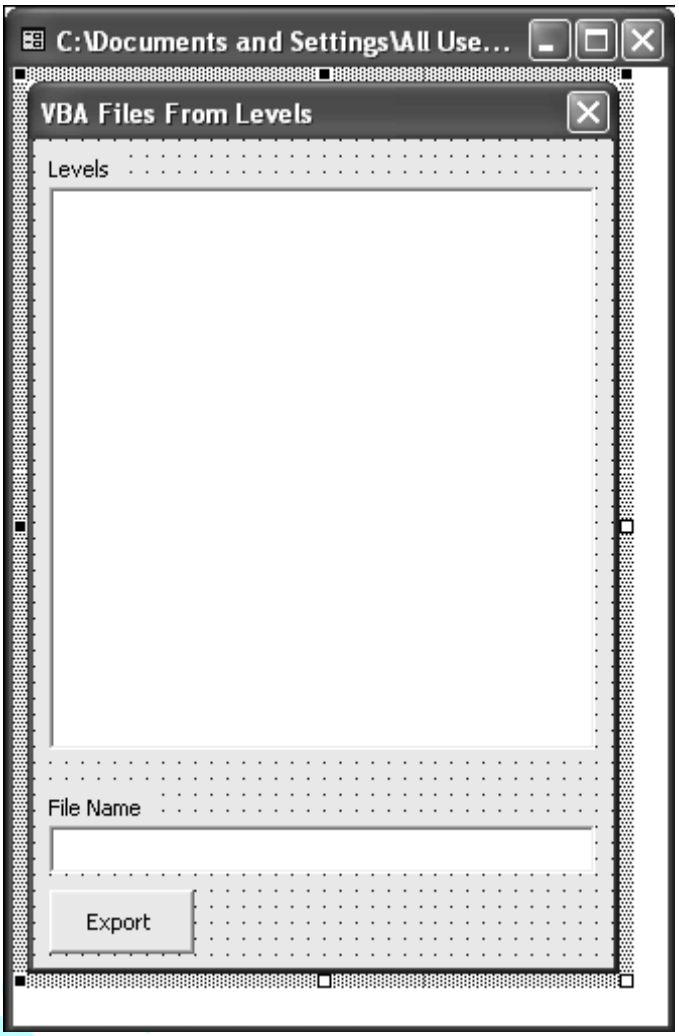
到此为止，如果文本元素高度相同的话，本代码均匀排布这些文本。调整文本元素间隔时使用的是每个元素的左上角点，如果一个文本元素比其它的大，排布后可能会压盖其下的文本，因为我们仅考虑各文本左上角间的距离，而不是一个文本元素的左上角与它上面元素的左下角间的距离。我们把扩展该宏的工作留给本书的读者来完成，这里的扩展是指均匀排布具有不同高度的文本。

China BDN

frmExportElements.frm

frmExportElements.frm 用户窗体实现一个简单任务：它导出指定层上的元素到一个新设计文件中。

本工程的任务简单，界面如右图所示。需要用户选择任意数量的层，输入一个文件名作为新建立的文件，然后点击 Export(导出)按钮。



控件名称

- lstLevels
- txtFileName
- btnExport

控件属性

- lstLevels 属性 MultiSelect 设置为 2-fmMultiSelectExtended
- lstLevels 属性 ListStyle 设置为 1-fmListStyleOption

程序开始执行时，需要把当前设计文件中所有的层名取到列表框中，这非常容易做到。由于刚取出的层名未按字母排序，所以我们使用一个冒泡排序按字母顺序将这些层名放入列表框中。

```
Private Sub UserForm_Initialize()
    Dim myLevel As Level
    Dim LevelNames() As String
    Dim MadeChange As Boolean
    Dim tmpName As String
    Dim I As Long
```

```
    ReDim LevelNames(0)
    For Each myLevel In ActiveDesignFile.Levels
        LevelNames(UBound(LevelNames)) = myLevel.Name
        ReDim Preserve LevelNames(UBound(LevelNames) + 1)
    Next
    ReDim Preserve LevelNames(UBound(LevelNames) - 1)
    MadeChange = True
    While MadeChange = True
        MadeChange = False
        For I = LBound(LevelNames) To UBound(LevelNames) - 1
            If StrComp(LevelNames(I), LevelNames(I + 1)) = 1 Then
                tmpName = LevelNames(I)
                LevelNames(I) = LevelNames(I + 1)
                LevelNames(I + 1) = tmpName
                MadeChange = True
            End If
        Next I
    Wend
    For I = LBound(LevelNames) To UBound(LevelNames)
        lstLevels.AddItem LevelNames(I)
    Next I
End Sub
```

当比较数字值时，可以用大于(>)和小于(>)比较符。这两个比较符也能用于文本的比较，但结果并不总是像我们期望的那样。所以，用标准 VBA 函数 StrComp 来比较两个字符串。

```
Private Sub btnExport_Click()
    Dim myFileName As String
    Dim myNewFile As DesignFile
    Dim I As Long
    Dim ElemID As DLong
```

```
    Dim myElems() As Element
    Dim myElemEnum As ElementEnumerator
    Dim myLevel As Level
    myFileName = txtFileName.Text
    If ActiveModelReference.Is3D Then
        CreateDesignFile "seed3d", myFileName, False
    Else
        CreateDesignFile "seed2d", myFileName, False
    End If
    Set myNewFile = OpenDesignFileForProgram(myFileName)
    Dim mySelCriteria As New ElementScanCriteria
    mySelCriteria.ExcludeAllLevels
    For I = 1 To lstLevels.ListCount
        If lstLevels.Selected(I - 1) Then
            mySelCriteria.IncludeLevel _
                ActiveModelReference.Levels(lstLevels.List (I - 1))
        End If
    Next I
    Set myElemEnum = ActiveModelReference.Scan(mySelCriteria)
    myElems = myElemEnum.BuildArrayFromContents
    For I = LBound(myElems) To UBound(myElems)
        myNewFile.Models(1).CopyElement myElems(I)
    Next I
    myNewFile.Save
    MsgBox UBound(myElems) + 1 & " elements created in file" & vbCr & _
            myFileName, vbInformation, Me.Caption
End Sub
```

以上代码中用 CreateDesignFile 建立一个新设计文件，用 OpenDesignFileForProgram 打开这个文件。OpenDesignFileForProgram 能让

China BDN

我们打开并操作一个文件而不会在 MicroStation 界面中看到该文件。

随后，我们建立了一个 ElementScanCriteria 对象，仅在选择的层上查找元素。先用 ExcludeAllLevels 方法排除所有层，然后将 lstLevels 列表框中选中的层添加进来。在向扫描准则添加层时也可以将这些层添加到新建的设计文件中，但这一步不需要，因为在复制元素到新文件时会建立那些没有的层。然而，假如我们在界面中选择了一个层而该层中没有任何元素，则新设计文件中也将没有该层。

向用户提供反馈和信息

与窗体一起导入的代码含有这些事件：

```
Private Sub UserForm_MouseMove(ByVal Button As Integer, _
        ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
    ShowPrompt ""
    ShowStatus ""
    ShowCommand ""
End Sub
Private Sub lstLevels_MouseMove(ByVal Button As Integer, _
        ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
    ShowPrompt ""
    ShowStatus ""
    ShowCommand ""
End Sub
Private Sub txtFileName_MouseMove(ByVal Button As Integer, _
        ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
    ShowPrompt ""
    ShowStatus ""
    ShowCommand ""
End Sub
Private Sub btnExport_MouseMove(ByVal Button As Integer, _
        ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
    ShowPrompt ""
```

```
        ShowStatus ""
        ShowCommand ""
    End Sub
```

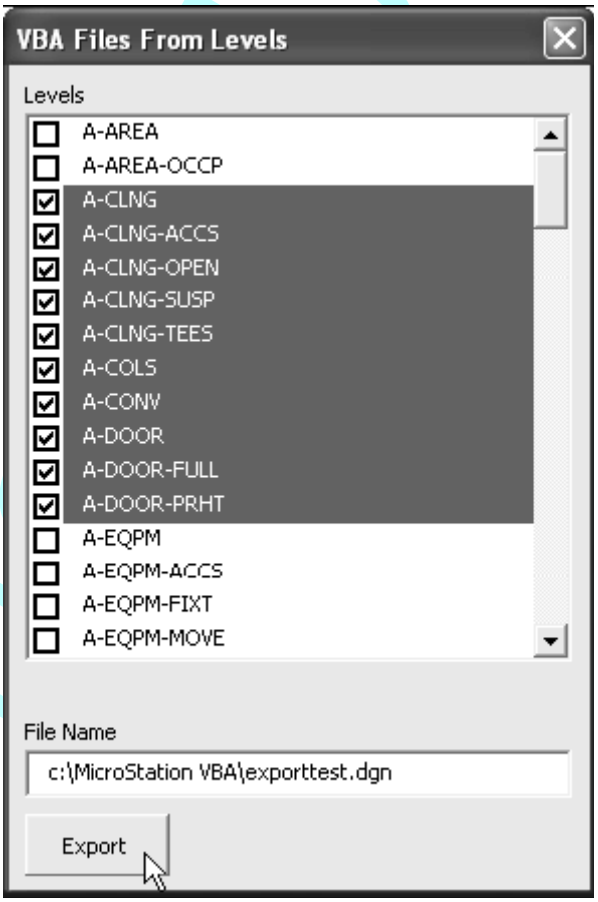
没有什么需要发送到 MicroStation 状态栏的提示区、状态区或命令区的。考虑到本程序的功能，这些区域应该给什么值呢？还是把答案留给它们自己吧。

下面是工作中的“VBA Files From Levels”窗体。该程序不需要非模态，因为窗体显示时不需要用户与 MicroStation 交互，但仍然需要在一个代码模块过程中显示它。

```
Sub TestFilesFromLevels()
    frmExportElements.Show
End Sub
```

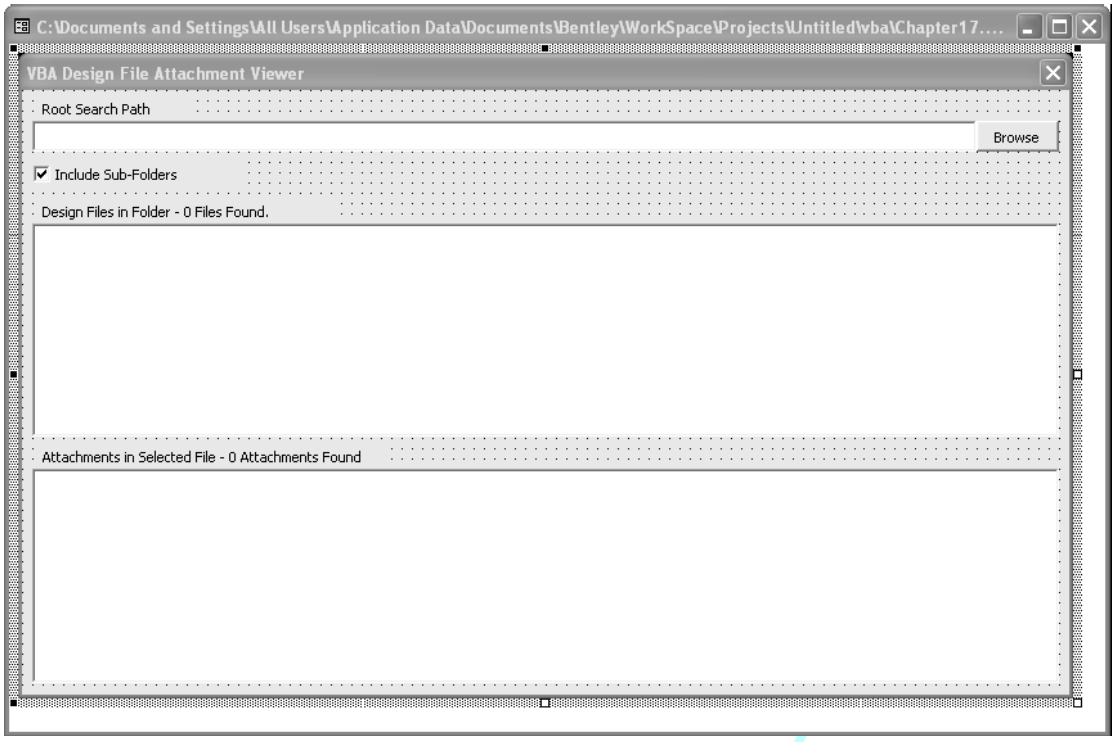
以上代码放在一个代码模块中，这样用户就能执行本程序了。

右图是使用中的界面。可选择任意多个层并导出到由输入名指定的文件中。



frmDFAV.frm

frmDFAV.frm 程序用于显示设计文件的附件。



用户通过点击 Browse(浏览)按钮选择一个文件夹，程序在被选的文件夹中浏览 MicroStation 的.dgn 文件。每个找到的设计文件被加入到 File(文件)列表框中。当用户在 File 列表框中点击一个文件时，程序查找该文件中的附件，所有找到的附件被添加到 Attachments(附件)列表框中。

控件属性

- txtFolder
- btnBrowse
- lstFiles
- lstAttachments



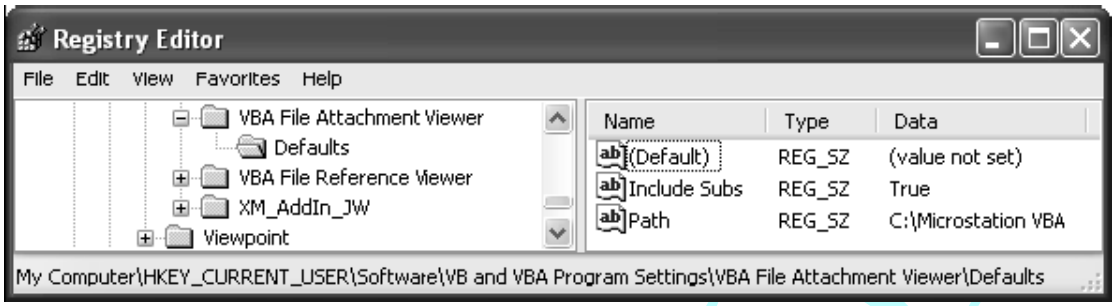
我们将在该程序中加入一些至今还未用到过的元素。我们可以让用户键入文件夹，但选择文件夹会更好些。右图是我们想要的文件夹选择对话框：

怎样得到它？使用 Windows API。

我们想做的另一件事是存储应用的设置，这样下次使用程序时就能恢复这些

设置。这些设置保存在 Windows 注册表中。

下图是注册表中看上去的样子：



程序组成

- 窗体初始化时从注册表中检索设置
- 让用户选择根文件夹
- 在文件夹中搜索.dgn 文件
- 在文件夹的子文件夹中搜索.dgn 文件
- 用户点击文件时显示文件的附件
- 程序关闭时保存设置到注册表中

窗体初始化时从注册表中检索设置

```
Private Sub UserForm_Initialize()
    txtFolder.Text = GetSetting("VBA File Attachment Viewer", _
                                "Defaults", "Path")
    If txtFolder.Text <> "" Then
        chkSubFolders.Value = GetSetting("VBA File Attachment Viewer", _
                                "Defaults", "Include Subs")
        PopulateFileList
    End If
End Sub
```

窗体初始化时查找保存的设置并把它们放入窗体中。

如果在注册表中发现了保存的“path(路径)”，就进一步设置复选框的值并用我们自己的 PopulateFileList 方法来赋值文件列表。

让用户选择根文件夹

使用 Windows API 来让用户选择一个文件夹。在通用声明区作如下声明：

```
Private Type BrowseInfo
    hWndOwner As Long
    pidlRoot As Long
    sDisplayName As String
    sTitle As String
    ulFlags As Long
    lpfn As Long
    lParam As Long
    iImage As Long
End Type
Private Declare Function SHBrowseForFolder Lib "shell32.dll" _
        (bBrowse As BrowseInfo) As Long
Private Declare Function SHGetPathFromIDList Lib "shell32.dll" _
        (ByVal lItem As Long, ByVal sDir As String) As Long
Private Const Bif_ReturnOnlyFSDirs = 1
Private Const Bif_DontGoBelowDomain = 2
Private Const Bif_EditBox = 16
Private Const Bif_NewDialogStyle = 64
Private Const Bif_UseNewui = 80
Private Const Bif_BrowseForComputer = 4096
Private Const Bif_BrowseForPrinter = 8192
Private Const Bif_BrowseIncludeFiles = 16384
```

省明晚常数、类型和函数后就能在代码中使用它们了。下面是 Browse 按钮的 Click 事件。

```
Private Sub btnBrowse_Click()
    Dim MyBI As BrowseInfo
    Dim FList As Long
    Dim DirName As String
    Dim SelFolder As Long
    DirName = Space(255)
```


China BDN

```
        MyBI.sTitle = "Select Root Folder:"
        MyBI.sDisplayName = Space(255)
        MyBI.ulFlags = Bif_ReturnOnlyFSDirs
        FList = SHBrowseForFolder(MyBI)
        SelFolder = SHGetPathFromIDList(FList, DirName)
        DirName = Left(DirName, InStr(1, DirName, Chr(0)) - 1)
        If DirName <> "" Then
            txtFolder.Text = DirName
        Else
            txtFolder.Text = ""
        End If
        PopulateFileList
    End Sub
```

在文件夹中搜索.dgn 文件

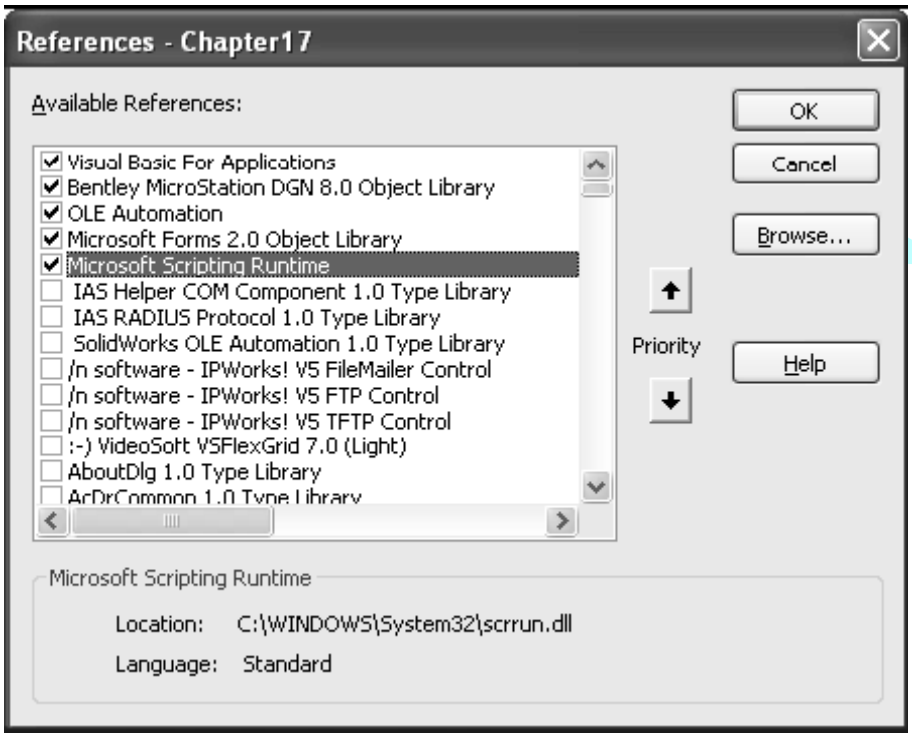
PopulateFileList 是一个取得文本框中的文件夹并开始查找.dgn 文件的过程。

```
    Sub PopulateFileList()
        lstFiles.Clear
        Dim myFolder As Folder
        Dim myFSO As New FileSystemObject
        If txtFolder.Text <> "" Then
            Set myFolder = myFSO.GetFolder(txtFolder.Text)
            FilesInFolder myFolder, "dgn", chkSubFolders, lstFiles
        End If
        lblFiles.Caption = "Design Files in Folder - " & lstFiles.ListCount & _
                           " Files Found."
    End Sub
```

PopulateFileList 使用了 FileSystemObject。FileSystemObject 是一个 Windows 组建，使用它前必须将其加入到引用中。

China BDN

在 VBA 中，选 **Tools > References (工具 > 引用)**显示引用对话框：



Microsoft Scripting Runtime(微软脚本运行时)提供了一种查找文件和遍历文件夹及其子文件夹的简易方法。

PopulateFileList 方法利用了我们自己的 FilesInFolder 方法。

FilesInFolder 方法是一个递归方法，意思是它调用它自己。下面就是这个方法。

```
Sub FilesInFolder(FolderIn As Folder, FileExtension As String, _
                  IncludeSubs As Boolean, _
                  ListToPopulate As ListBox)
    Dim myFile As File
    For Each myFile In FolderIn.Files
        Select Case UCase(Right(myFile.Name, 3))
            Case UCase(FileExtension)
                ListToPopulate.AddItem myFile.Path
        End Select
    Next
    If IncludeSubs = True Then
        Dim subFolder As Folder
```

```
            For Each subFolder In FolderIn.SubFolders
                FilesInFolder subFolder, FileExtension, _
                    IncludeSubs, ListToPopulate
            Next
        End If
    End Sub
```

FilesInFolder 做的第一件事就是查找“FolderIn”参数文件夹下的文件。我们考察文件扩展名看其是否匹配 FileExtension 参数，如果匹配，就把该文件加入列表中。查看完所有文件后，检查是否还应该考察子文件夹。如果不考察子文件夹就完成该函数并返回上级；如果要考察子文件夹，我们启动一个 For ... Each 语句来考察当前文件夹中的每一个子文件夹。

对于用 FilesInFolder 找到的每个子文件夹，我们用该子文件夹作为 FolderIn 参数来调用 FilesInFolder。这就是该过程为何是递归之所在，过程调用其本身。当处理递归过程或函数时，需要保证有一种途径来完成执行。否则，我们将会面对调用堆栈中成百上千的过程无法中止执行的局面。

显示附件

当用户在 File 列表框中选择一个文件后，我们取得被选文件的附件并将它们显示在 Attachments 列表框中。

```
Private Sub lstFiles_Click()
    lstAttachments.Clear
    If lstFiles.Text <> "" Then
        Dim myDesFile As DesignFile
        Set myDesFile = OpenDesignFileForProgram(lstFiles.Text, True)
        Dim myAttachment As Attachment
        For Each myAttachment In _
                myDesFile.DefaultModelReference.Attachments
            lstAttachments.AddItem myAttachment.AttachName
        Next
        myDesFile.Close
    End If
```

```
    lblAttachments.Caption = "Attachments in Selected File - " & _
                             lstAttachments.ListCount & " Attachments Found."
End Sub
```

在考察一个文件的附件前需要先打开该文件。有两种打开文件的方式：一种是能让用户看到并操作的打开，另一种是用户看不到文件但程序可操作的打开。本例用 OpenDesignFileForProgram 打开文件，因为我们不想在文件每次被选取后在 MicroStation 的编辑窗口中打开它。

用户查看完文件的附件后会关闭程序，我们想在程序被关闭时保存设置，这样下次执行程序时这些设置就可以恢复。

```
    Private Sub UserForm_QueryClose(Cancel As Integer, _
                                    CloseMode As Integer)
        SaveSetting "VBA File Attachment Viewer", "Defaults", _
                    "Path", txtFolder.Text
        SaveSetting "VBA File Attachment Viewer", "Defaults",
                    "Include Subs", _ chkSubFolders.Value
    End Sub
```

我们在 Windows 注册表中保存了两个设置，它们在窗体初始化时被读出。

与 MDL 应用交互

让我们记录一个导入图像的宏：选 MicroStation 菜单 **File > Import > Image (文件 > 导入 > 图像)**，浏览到 C:\Program Files\Bentley\MicroStation\bentleyb.jpg 并将其放入 MicroStation 中。图像放置完后停止记录宏。

在考察这个刚建立的宏之前，应该先讨论两种与 MDL 应用交互的方法。第一种方法看起来像以前用过的“SendCommand”方法，命令开始后给以所需的点或其它输入；第二种方法使用对话框，要求一个类模块处理对话框事件。

因为导入图像的 MDL 应用使用了对话框，所以我们建立了一个新类，该类将被记录的宏所使用。让我们先考察记录宏，尔后考察宏记录器建立的类模块。

```
Sub Macro5()
    Dim startPoint As Point3d
    Dim point As Point3d, point2 As Point3d
```

```
    Dim lngTemp As Long
    Dim modalHandler As New Macro5ModalHandler
    AddModalDialogEventsHandler modalHandler
    ' The following statement opens modal dialog "Select Image File"
    ' Start a command
    CadInputQueue.SendCommand "MDL LOAD PLAIMAGE"
    ' Coordinates are in master units
    startPoint.X = -6.270784
    startPoint.Y = 23.160278
    startPoint.Z = 0#
    ' Send points to simulate a down-drag-up action
    point.X = startPoint.X
    point.Y = startPoint.Y
    point.Z = startPoint.Z
    point2.X = point.X + 2.938037
    point2.Y = point.Y - 2.980928
    point2.Z = point.Z
    CadInputQueue.SendDragPoints point, point2, 1
    RemoveModalDialogEventsHandler modalHandler
    CommandState.StartDefaultCommand
End Sub
```

建立的类模块叫做 **Macro5ModalHandler**。声明完几个变量后，宏记录器声明了一个“Macro5ModalHandler”类型的变量并给该类添加事件。下一步发出“MDL LOAD PLAIMAGE”命令，该命令显示对话框。对话框显示后，类模块处理文件路径和名称的输入并关闭对话框。对话框关闭后又返回Macro5过程，通过“拖拽”点将图像放入设计文件中。最后移除事件处理器并结束过程。

让我们详细分析建立的类模块。

```
Implements IModalDialogEvents
Private Sub IModalDialogEvents_OnDialogClosed(ByVal _
    DialogBoxName As String, ByVal DialogResult As MsdDialogBoxResult)
```


China BDN

```
End Sub
Private Sub IModalDialogEvents_OnDialogOpened(ByVal _
        DialogBoxName As String, DialogResult As MsdDialogBoxResult)
    If DialogBoxName = "Select Image File" Then
        CadInputQueue.SendCommand _
            "MDL COMMAND MGDSHOOK,fileList_setFilterCmd *.cal"
        CadInputQueue.SendCommand _
            "MDL COMMAND MGDSHOOK,fileList_setDirectoryCmd " & _
            "C:\Program Files\Bentley\MicroStation\"
        CadInputQueue.SendCommand _
            "MDL COMMAND MGDSHOOK,fileList_setFileNameCmd " & _
            "bentleyb.jpg"
        ' Remove the following line to let the user close the dialog box.
        DialogResult = msdDialogBoxResultOK
    End If ' Select Image File
End Sub
```

宏 Macro5 每次运行时，都在同样的位置放置了同样的图像。让我们稍微修改一下代码以便将来的工程中也能用，我们要建立一个更灵活且功能强大的类模块。

下面是新的类模块代码，取名叫做 **clsImageInsertion**。添加了两个公共变量作为类模块的属性。

```
Implements IModalDialogEvents
Public FilePath As String
Public FileName As String
Private Sub IModalDialogEvents_OnDialogClosed(ByVal _
    DialogBoxName As String, ByVal DialogResult As MsdDialogBoxResult)
End Sub
Private Sub IModalDialogEvents_OnDialogOpened(ByVal _
        DialogBoxName As String, DialogResult As MsdDialogBoxResult)
    If DialogBoxName = "Select Image File" Then
```

```
        CadInputQueue.SendCommand _
            "MDL COMMAND MGDSHOOK,fileList_setDirectoryCmd " & _
            FilePath
        CadInputQueue.SendCommand _
            "MDL COMMAND MGDSHOOK,fileList_setFileNameCmd " & _
            FileName
        DialogResult = msdDialogBoxResultOK
    End If
End Sub
```

路径和文件名不再硬编码(即不再写成固定的内容)。这就意味着每当想在文件中插入图像时就能使用这个类模块。下面是其使用方法：

```
Sub TestImageInsertion()
    Dim point1 As Point3d, point2 As Point3d
    Dim modalHandler As New clsImageInsertion
    modalHandler.FilePath = "C:\Program Files\Bentley\MicroStation\"
    modalHandler.FileName = "bentleyb.jpg"
    AddModalDialogEventsHandler modalHandler
    CadInputQueue.SendCommand "MDL LOAD PLAIMAGE"
    point1.X = 0: point1.Y = 0: point1.Z = 0
    point2.X = 1: point2.Y = 1: point2.Z = 0
    CadInputQueue.SendDragPoints point1, point2, 1
    RemoveModalDialogEventsHandler modalHandler
    CommandState.StartDefaultCommand
End Sub
```

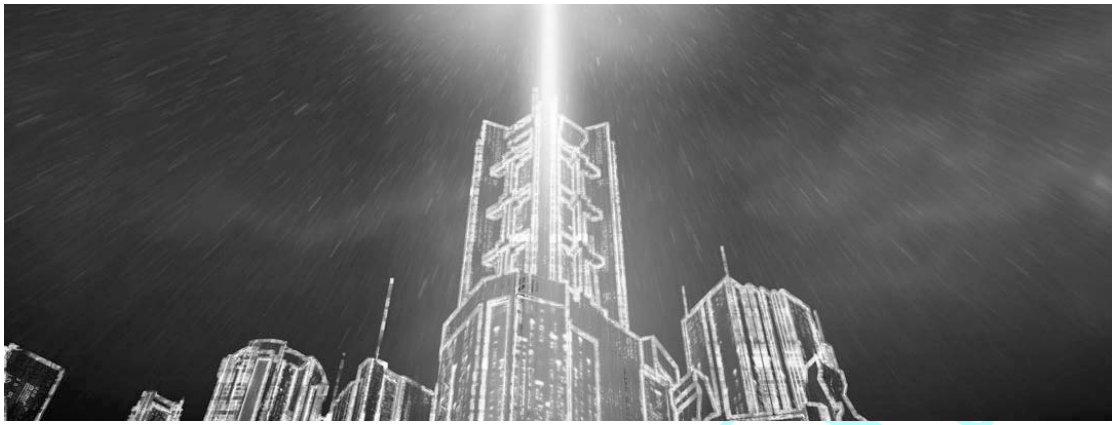
借助类模块的 FilePath 和 FileName 属性能让类模块使用任何文件路径或文件名，在以前，路径和名称都是硬编码的。

本章回顾

MicroStation VBA API 功能强大，这种强大能让我们创造性地完成编程任务。例如，当用户选择两个点后启动“PLACE LINE”命令并向用户提供更图形

化的丰富体验，这种情况甚至能用于不是在画一条线的时候。使用非模态窗体能让用户在与我们自己的 GUI 交互的同时，还能完全自由地操作 MicroStation 界面。

China BDN



18. 接口的本质

接口是什么？这个术语在编程中用来描述几件不同的事情，让我们解释并定义这个术语。

我们已经利用用户窗体和控件建立过图形用户界面(译者注：“界面”和“接口”的英文都是 interface)，它能让用户与控件交互。图形向程序代码提供了接口(或称之为界面)。

另一种接口类型能与程序代码交互但没有图形组件，它让我们在 MicroStation 中与用户的活动进行交互。例如，当用户在 MicroStation 中选择一个元素时，我们能借助于一个叫做“ILocateCommandEvents”的接口捕获这个动作；当用户在 MicroStation 中拾取点时，我们能借助于一个叫做“IPrimitiveCommandEvents”的接口捕获这些点。

用户与 MicroStation 中的一些对话框交互能借助于“IModalDialogEvents”来考察。

本章内容：

- 接口基础
- 类模块回顾
- 类模块的生命期
- 动态事件
- LocateCriteria 对象
- IPrimitiveCommandEvents 接口
- 优化动态事件

接口基础

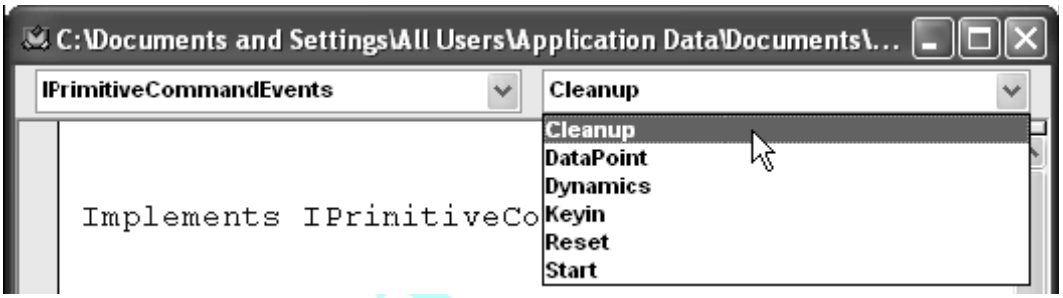
在 MicroStation 中捕获用户交互这种功能很强大。为了利用这种强大功能，我们建立一个新的实现接口的类模块。例如，为了捕获 MicroStation 中的点选择集，我们在 VBA 工程中插入一个新的类模块，并把如下行放入类模块的通用声明区中：

```
Implements IPrimitiveCommandEvents
```

在类模块中使用“Implements”关键字表示该类模块继承接口的方法或事件。



当使用 Implements 关键字时，接口名出现在类模块的对象复合框中。



在如上所示的对象复合框中选择接口后，方法复合框中会显示可用的接口方法。

正如我们能看到的，IPrimitiveCommandEvents 接口公开了六个方法或事件，它们是 Cleanup、DataPoint、Dynamics、Keyin、Reset 和 Start。

和用户窗体控件不同，接口的每个方法即使不使用也都必须声明，而在用户窗体控件中只需要选取想显示和操作的事件即可。声明接口每个方法的最简单的途径是，选择方法复合框中的每个方法，每选择一个方法，VBA 自动写出 Sub ... End Sub 代码。按此方式操作 IPrimitiveCommandEvents 接口，将在类模块中看到如下内容：

```
Implements IPrimitiveCommandEvents
Private Sub IPrimitiveCommandEvents_Cleanup()
```

```
End Sub
Private Sub IPrimitiveCommandEvents_DataPoint(Point As Point3d, _
                ByVal View As View)
End Sub
Private Sub IPrimitiveCommandEvents_Dynamics(Point As Point3d, _
        ByVal View As View, ByVal DrawMode As MsdDrawingMode)
End Sub
Private Sub IPrimitiveCommandEvents_Keyin(ByVal Keyin As String)
End Sub
Private Sub IPrimitiveCommandEvents_Reset()
End Sub
Private Sub IPrimitiveCommandEvents_Start()
End Sub
```

现在就能在事件中输入代码了。

类模块回顾

接口必须用一个类模块予以公开。我们已经讨论过类模块，但快速回顾一下还是有必要的。

用 VBA 菜单 **Insert > Class Module (插入 > 类模块)**建立新的类模块。缺省情况下，VBA 把新的类模块命名为“Class1”、“Class2”、“Class3”、“Class4”等，应将它们改为有助于理解其含义的名字。例如，把写文件的类命名为“clsFileWriter”。在随后的一个例子中，命名类为“LCE_Text”，表明我们操作的是 ILocateCommandEvents 接口(LCE)，该类处理文本。

插入类模块并改名后开始编写代码。方法和函数的编写非常类似于代码模块。可以为类模块建立事件，可以在代码模块的通用声明区中声明“Public”变量来建立属性，还可以通过“Get”和“Set”(或“Let”)语句来建立属性。

建立类的最终结果是一个新对象。类不能单独运行，它们需要别的代码来初始化、设置属性并使用其方法。下面将演示两种调用类模块的方式：第一种是建立一个叫做“clsNetNode”的类模块。下面是代码：

```
Private Type IPAddr
```

```
        Set1 As Byte
        Set2 As Byte
        Set3 As Byte
        Set4 As Byte
    End Type
    Public Name As String
    Private IPAddress As IPAddr
    Sub Ping()
        MsgBox "Pinging " & IPAddress.Set1 & "." & _
                            IPAddress.Set2 & "." & _
                            IPAddress.Set3 & "." & _
                            IPAddress.Set4, , Name
    End Sub
    Sub SetIPAddress(IPA As Byte, IPB As Byte, IPC As Byte, _
                        IPD As Byte)
        IPAddress.Set1 = IPA
        IPAddress.Set2 = IPB
        IPAddress.Set3 = IPC
        IPAddress.Set4 = IPD
    End Sub
```

该类有一个属性(Name)和两个方法(Ping 和 SetIPAddress)。SetIPAddress 方法设置私有变量“IPAddress”的 IP 地址值，Ping 方法在消息框中显示输入的 IP 地址并以 Name 属性作为消息框的标题。

第一种调用类模块的方式是声明一个类类型的变量，设置变量为“New”类类型，然后设置属性和使用方法。

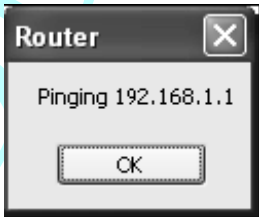
```
    Sub TestClsNetNodeA()
        Dim myNetNode As clsNetNode
        Set myNetNode = New clsNetNode
        myNetNode.SetIPAddress 192, 168, 1, 1
        myNetNode.Name = "Router"
```


China BDN

```
    myNetNode.Ping
End Sub
```

第二种利用类模块的方式是声明一个“New”类类型的变量，然后开始设置属性和使用方法。如下所示：

```
Sub TestClsNetNodeB()
    Dim myNetNode As New clsNetNode
    myNetNode.SetIPAddress 192, 168, 1, 1
    myNetNode.Name = "Router"
    myNetNode.Ping
End Sub
```



这两种声明和初始化类模块的方式区别很小。在本例中它们的结果是相同的。

类模块的生命期

当实现一个简单的类模块后(如“clsNetNode”)，仅当声明为类模块的变量在作用范围内时该类才有效。在以上两个测试过程中，我们在过程内声明了变量**myNetNode**，这样，clsNetNode 类仅在变量声明的过程内有效。

在通用声明区中声明为类模块的变量对模块内的所有方法都有效，其它模块把该变量当作“Public”。

一旦声明为特定类的变量超出了其作用范围，对象就自动终结。这不是终结一个声明类的唯一途径，还可将代表类的变量设置为 Nothing 来终结类，如下所示：

```
Set myNetNode = Nothing
```

对类的生命期的这一解释与利用类模块的环境有很大关系。该规则的一个例外是使用 MicroStation 的 CommandState 对象的 StartLocate 和 StartPrimitive 方法的类。当使用 StartLocate 和 StartPrimitive 时，MicroStation 保持类并通报事件直到 MicroStation 关闭或我们使用“CommandState.StartDefaultCommand”或使用“Remove...”方法之一。在本书稍后部分将会看到“Remove...”方法的例子。

明白用于实现 MicroStation 接口的类的生命期很重要，因为我们用于调用这

些类的代码执行后过程结束了，但类仍存在因为 MicroStation 正保持着它。

让我们讨论两个 MicroStation 的接口。第一个是 ILocateCommandEvents 接口。

ILocateCommandEvents

IlocateCommandEvents 接口能让用户选择或“Locate(定位)”元素。下面是通过 IlocateCommandEvents 接口公开的事件：

- Private Sub ILocateCommandEvents_Accept(ByVal Element As Element, Point As Point3d, ByVal View As View)
- Private Sub ILocateCommandEvents_Cleanup()
- Private Sub ILocateCommandEvents_Dynamics(Point As Point3d, ByVal View As View, ByVal DrawMode As MsdDrawingMode)
- Private Sub ILocateCommandEvents_LocateFailed()
- Private Sub ILocateCommandEvents_LocateFilter(ByVal Element As Element, Point As Point3d, Accepted As Boolean)
- Private Sub ILocateCommandEvents_LocateReset()
- Private Sub ILocateCommandEvents_Start()

每一事件都在特定时间触发。有些事件提供诸如定位到了哪个元素之类的信息。每当使用 MicroStation 的接口对象时，无论用还是不用，每个事件或方法都必须声明。让我们看一个例子。

以转换文本元素为大写例子开始。

该类模块的名字是 **LCE_Text**。下面是类模块中的代码：

```
Implements ILocateCommandEvents
Private SelElement As Element
Private Sub ILocateCommandEvents_Accept(ByVal _
        Element As Element, Point As Point3d, ByVal View As View)
    Dim elemText As TextElement
    Set elemText = Element
    elemText.Redraw msdDrawingModeErase
```

```
        elemText.Text = UCase(elemText.Text)
        elemText.Redraw msdDrawingModeNormal
        elemText.Rewrite
        ActiveModelReference.UnselectAllElements
        CommandState.StartDefaultCommand
End Sub
Private Sub ILocateCommandEvents_Cleanup()
End Sub
Private Sub ILocateCommandEvents_Dynamics(Point As Point3d, ByVal
                        View As View, ByVal DrawMode As MsdDrawingMode)
End Sub
Private Sub ILocateCommandEvents_LocateFailed()
        If SelElement Is Nothing = False Then
            ActiveModelReference.UnselectAllElements
            Set SelElement = Nothing
        End If
        ShowCommand "CAP Text"
        ShowPrompt "Select Text to be Capitalized"
End Sub
Private Sub ILocateCommandEvents_LocateFilter(ByVal _
            Element As Element, Point As Point3d, Accepted As Boolean)
        Accepted = False
        If Element.IsTextElement = True Then
            Set SelElement = Element
            Accepted = True
            ActiveModelReference.SelectElement    Element, True
            ShowCommand "CAP Text"
            ShowPrompt "Click again to confirm..."
        End If
End Sub
```

China BDN

```
Private Sub ILocateCommandEvents_LocateReset()
    CommandState.StartDefaultCommand
End Sub
Private Sub ILocateCommandEvents_Start()
End Sub
```

LocateFilter 事件

我们使用的第一个事件是 LocateFilter 事件，该事件能使我们指定所选的元素是否符合准则。缺省情况下，accepted 属性为真。如果 accepted 属性一直为真，用户就能通过在 MicroStation 中再次点击来“Accept(接受)”选择，用户“接受”选择后就触发 accept 事件，其中的代码将被执行。如果 LocateFilter 事件中的 accepted 参数设置为假，将触发 LocateFailed 事件。LocateFilter 事件返回一个假的 accepted 值从而使接口对象重新启动，这是很通常的一种情况。

Accept 事件

Accept 事件被触发的两个条件：第一、LocateFilter 事件必须以 accepted 属性为真退出；第二、用户必须在 MicroStation 中用左键点击来“接受”已经过滤的元素。如果 LocateFilter 成功退出后用户在 MicroStation 中右击，则 LocateFilter 被复位但并不完全退出接口。当上两个条件(LocateFilter 和用户接受)满足时，Accept 事件中的代码被执行。

LocateReset 事件

LocateReset 事件是 ILocateCommandEvents 接口中最后被触发的事件。在进入 LocateFilter 事件前或进入 LocateFilter 事件后但 accepted 属性被设置为假时，用户在 MicroStation 中右击发出一个复位动作时触发 LocateReset 事件。请记住，LocateReset 事件告知我们用户要求复位，此时我们的代码应该发出“CommandState. StartDefaultCommand”来退出接口。

LocateFailed 事件

当用户点击选择但什么也没有找到时触发 LocateFailed 事件。通过使用“CommandState.StartDefaultCommand”，该事件能用于退出接口。

Start 事件

Start 事件是使用 ILocateCommandEvents 接口时最先被触发的事件，在其中可以设置变量或其它对象。

Cleanup 事件

仅在 LocateReset 事件前触发 Cleanup 事件。正如其名称所指的那样，该事件用于清理变量、对象或接口所用的引用。

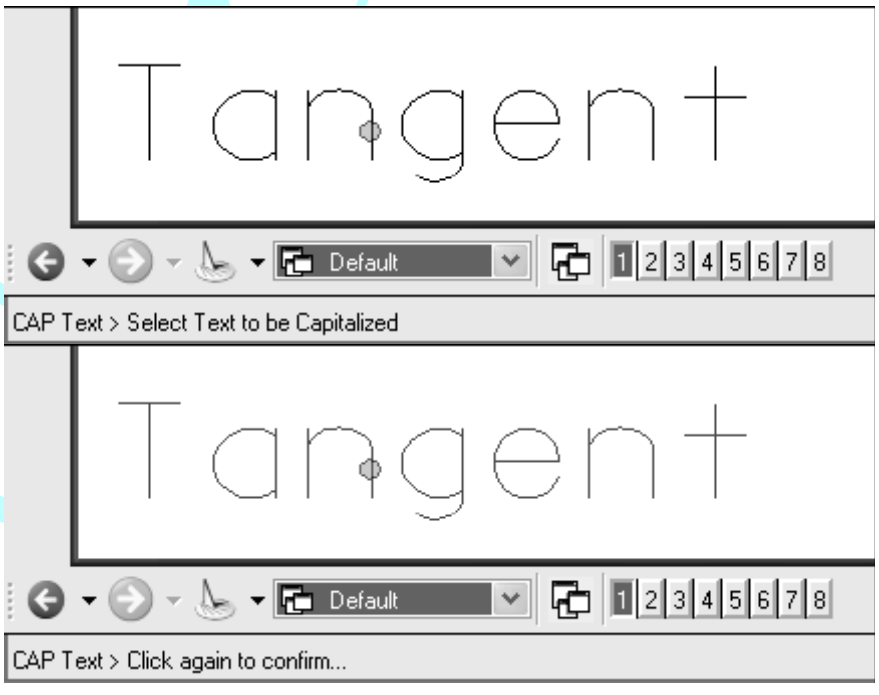
Dynamics 事件

Dynamics 事件提供动态实时反馈。本章稍后的一个例子演示了如何使用该事件。

类模块仅靠自己是不能工作的——它们需要由其它的代码来建立。下面是使用新 **LCE_Text** 类的过程。

```
Sub tstLCE_Text()
    CommandState.StartLocate New LCE_Text
    ShowCommand "CAP Text"
    ShowPrompt "Select Text to be Capitalized"
End Sub
```

右侧是程序运行的屏幕快照。注意命令和提示对用户的指导。





下面是对大写文本程序的另一种变化，仅有的不同点是处理文本大写部分的代码。现在把处理文本的代码放入了 LocateFilter 事件中，这样使得选择的文本元素不用等待用户的确认就被大写化。

```
Implements ILocateCommandEvents
Private SelElement As Element
Private Sub ILocateCommandEvents_Accept(ByVal Element As Element,
                                        Point As Point3d, ByVal View As View)
End Sub
Private Sub ILocateCommandEvents_Cleanup()
End Sub
Private Sub ILocateCommandEvents_Dynamics(Point As Point3d, _
        ByVal View As View, ByVal DrawMode As MsdDrawingMode)
End Sub
Private Sub ILocateCommandEvents_LocateFailed()
    If SelElement Is Nothing = False Then
        ActiveModelReference.UnselectAllElements
        Set SelElement = Nothing
    End If
    ShowCommand "CAP Text"
    ShowPrompt "Select Text to be Capitalized"
End Sub
Private Sub ILocateCommandEvents_LocateFilter(ByVal Element As _
                        Element, Point As Point3d, Accepted As Boolean)
    Accepted = False
```

```
    Dim elemText As TextElement
    If Element.IsTextElement = True Then
        Set elemText = Element
        elemText.Redraw msdDrawingModeErase
        elemText.Text = UCase(elemText.Text)
        elemText.Redraw msdDrawingModeNormal
        elemText.Rewrite
        ActiveModelReference.UnselectAllElements
        CommandState.StartDefaultCommand
    End If
End Sub
Private Sub ILocateCommandEvents_LocateReset()
    CommandState.StartDefaultCommand
End Sub
Private Sub ILocateCommandEvents_Start()
End Sub
```

下面是初始化接口对象的代码。

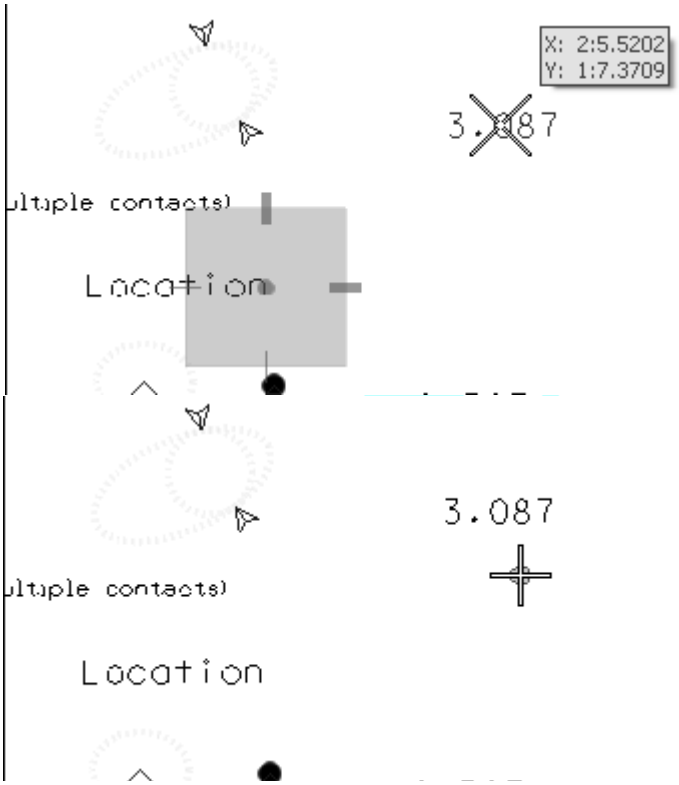
```
Sub tstLCE_Text2()
    CommandState.StartLocate New LCE_Text2
    ShowCommand "CAP Text"
    ShowPrompt "Select Text to be Capitalized"
End Sub
```

我们没有使用前面提及的 Dynamics 事件，现在让我们来使用它。

这段代码动态绘制一个新的文本元素，该文本显示初始选取点与光标位置间的距离。这种显示是实时的，文本随光标移动而变化。

China BDN

右图是这个接口运行中的两个例子。第一幅图是在初始选择后光标正在拖动的状态；第二幅图是在鼠标按钮点击后，“距离文字”被放置的状态。



下面是这个叫做 LCE_DistanceText 的类模块的代码。

```
Implements ILocateCommandEvents
Private selElem As Element
Private pt3StPoint As Point3d
Private dblDistance As Double
Private Sub ILocateCommandEvents_Accept(ByVal Element As Element, _
                                        Point As Point3d, ByVal View As View)
    Dim txtElem As TextElement
    Dim rotMatrix As Matrix3d
    dblDistance = Point3dDistance(Point, pt3StPoint)
    Set txtElem = CreateTextElement1(selElem, Round(dblDistance, _
                                          3), Point, rotMatrix)
    ActiveModelReference.AddElement txtElem
    txtElem.Rewrite
    txtElem.Redraw
    CommandState.StartLocate Me
```

```
End Sub
Private Sub ILocateCommandEvents_Cleanup()
End Sub
Private Sub ILocateCommandEvents_Dynamics(Point As Point3d, _
                ByVal View As View, ByVal DrawMode As MsdDrawingMode)
    Dim tmpTxtElem As TextElement
    Dim rotMatrix As Matrix3d
    dblDistance = Point3dDistance(Point, pt3StPoint)
    Set tmpTxtElem = CreateTextElement1(selElem, Round(dblDistance, 3), _
                                        Point, rotMatrix)
    tmpTxtElem.Redraw DrawMode
    ShowPrompt "Select Distance Point:"
End Sub
Private Sub ILocateCommandEvents_LocateFailed()
    CommandState.StartLocate Me
End Sub
Private Sub ILocateCommandEvents_LocateFilter(ByVal -
                Element As Element, Point As Point3d, Accepted As Boolean)
    Set selElem = Element
    pt3StPoint = Point
    CommandState.StartDynamics
End Sub
Private Sub ILocateCommandEvents_LocateReset()
    CommandState.StartDefaultCommand
End Sub
Private Sub ILocateCommandEvents_Start()
    ShowCommand "Text Distance"
    ShowPrompt "Select Element for Base Point"
End Sub
```

China BDN

下面是通过类调用该接口的过程：

```
Sub tstLCE_DistanceText()
    CommandState.StartLocate New LCE_DistanceText
End Sub
```

这个宏演示了 Dynamics 事件的用法。仔细查看 LocateFilter 事件会发现 StartDynamics 方法，没有这个方法，Dynamics 事件就不会被触发。Dynamics 事件在光标点处建立了一个新文本元素，该文本显示初始定位点与光标位置间的距离。

LocateCriteria(定位准则)

当“定位”到一个元素时进入 LocateFilter 方法。前面的例子中我们使用这种方法来确定被选取元素的类型，这样做是可以的。但是假如知道了想要的元素种类，可以用 LocateCriteria 在选取前就指定要获取的元素。

```
Implements ILocateCommandEvents
Private SelElement As Element
Dim myLC As LocateCriteria
Private Sub ILocateCommandEvents_Accept(ByVal Element As Element, _
                                        Point As Point3d, ByVal View As View)
End Sub
Private Sub ILocateCommandEvents_Cleanup()
End Sub
Private Sub ILocateCommandEvents_Dynamics(Point As Point3d, _
                ByVal View As View, ByVal DrawMode As MsdDrawingMode)
End Sub
Private Sub ILocateCommandEvents_LocateFailed()
    If SelElement Is Nothing = False Then
        ActiveModelReference.UnselectAllElements
        Set SelElement = Nothing
    End If
    ShowCommand "CAP Text"
```

```
    ShowPrompt "Select Text to be Capitalized"
End Sub
Private Sub ILocateCommandEvents_LocateFilter(ByVal Element As _
                        Element, Point As Point3d, Accepted As Boolean)
    Dim elemText As TextElement
    Set elemText = Element
    elemText.Redraw msdDrawingModeErase
    elemText.Text = UCase(elemText.Text)
    elemText.Redraw msdDrawingModeNormal
    elemText.Rewrite
    ActiveModelReference.UnselectAllElements
    CommandState.StartDefaultCommand
End Sub
Private Sub ILocateCommandEvents_LocateReset()
    CommandState.StartDefaultCommand
End Sub
Private Sub ILocateCommandEvents_Start()
    Set myLC = CommandState.CreateLocateCriteria(True)
    myLC.ExcludeAllTypes
    myLC.IncludeType (msdElementTypeText)
    CommandState.SetLocateCriteria myLC
End Sub
```

我们使用 LocateCriteria 对象来预先过滤用户的选择，LocateCriteria 是由 SetLocateCriteria 方法启动的。这种做法优于在 LocateFilter 事件中构造选择集、查看元素属性然后接收或拒绝选择集。通过使用 LocateCriteria 对象，在到达 LocateFilter 事件时我们知道用户已经作出合法的选择。

下面是 LocateCriteria 对象的方法：

- Sub ExcludeAllClasses()
- Sub ExcludeAllLevels()
- Sub ExcludeAllTypes()

China BDN

- Sub ExcludeClass(ElemClass As MsdElementClass)
- Sub ExcludeLevel(Level As Level)
- Sub ExcludeType(Type As MsdElementType)
- Sub IncludeClass(ElemClass As MsdElementClass)
- Sub IncludeLevel(Level As Level)
- Sub IncludeOnlyHole()
- Sub IncludeOnlyLocked()
- Sub IncludeOnlyModified()
- Sub IncludeOnlyNew()
- Sub IncludeOnlyNonPlanar()
- Sub IncludeOnlyNonSnappable()
- Sub IncludeOnlyOld()
- Sub IncludeOnlyPlanar()
- Sub IncludeOnlySnappable()
- Sub IncludeOnlySolid()
- Sub IncludeOnlyUnlocked()
- Sub IncludeOnlyUnmodified()
- Sub IncludeType(Type As MsdElementType)

MicroStation VBA 帮助文件解释了上述每个方法的用法以及使用样例。

IPrimitiveCommandEvents

刚刚讨论完了 ILocateCommandEvents 接口，其根本用法是在设计文件中选择(或定位)元素。IprimitiveCommandEvents 对象则是捕获命令输入和点的选择。

下面是要操作的事件：

- Private Sub IPrimitiveCommandEvents_Cleanup()
- Private Sub IPrimitiveCommandEvents_DataPoint(Point As _
  Point3d, ByVal View As View)
- Private Sub IPrimitiveCommandEvents_Dynamics(Point As _

Point3d, ByVal View As View, ByVal DrawMode As MsdDrawingMode)

China BDN

- Private Sub IPrimitiveCommandEvents_Keyin(ByVal Keyin As String)
- Private Sub IPrimitiveCommandEvents_Reset()
- Private Sub IPrimitiveCommandEvents_Start()

有些事件看起来很熟悉：Start、Reset、Cleanup、Dynamics，我们已经使用过这些事件。还没有使用过的两个事件是 DataPoint 和 Keyin。让我们看几个关于这些事件如何工作的例子。

PCE_LineTest

PCE_LineTest 类从选择的第一点到当前光标位置绘制一条橡皮筋线，当用户选择第二点后，使用 StartDefaultCommand 退出这个类：

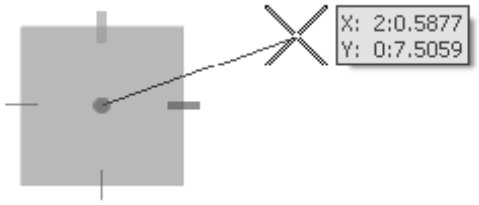
```
Implements IPrimitiveCommandEvents
Dim pt3BasePoint As Point3d
Dim boolSet As Boolean
Private Sub IPrimitiveCommandEvents_Cleanup()
End Sub
Private Sub IPrimitiveCommandEvents_DataPoint(Point As Point3d, _
                                              ByVal View As View)
    If boolSet = False Then
        pt3BasePoint = Point
        CommandState.StartDynamics
        boolSet = True
    Else
        CommandState.StartDefaultCommand
    End If
End Sub
Private Sub IPrimitiveCommandEvents_Dynamics(Point As Point3d, _
          ByVal View As View, ByVal DrawMode As MsdDrawingMode)
    Dim myLineElem As LineElement
   Set myLineElem = CreateLineElement2(Nothing, pt3BasePoint, Point)
    myLineElem.Redraw DrawMode
End Sub
```


China BDN

```
Private Sub IPrimitiveCommandEvents_Keyin(ByVal Keyin As String)
End Sub
Private Sub IPrimitiveCommandEvents_Reset()
End Sub
Private Sub IPrimitiveCommandEvents_Start()
End Sub
```

本例的大部分代码都在 DataPoint 事件和 Dynamics 事件中。请记住，我们仅想选择两个点。我们使用变量 **boolSet** 来记录是否选择了第一点。如果还没有选择基准点，**boolSet** 等于假，我们取得 Point 参数后存入变量 **pt3BasePoint** 中，用 StartDynamics 方法启动动态显示并将 **boolSet** 改为真。

当光标在 MicroStation 中移动时触发 Dynamics 事件，该事件每秒发生许多次。必须确保 Dynamics 事件中的代码不能太耗时。本例中，我们在初始选择点和当前光标位置(由 Point 参数给出)间建立一个新的 LineElement。



接口对象不能独自运行，它们需要代码模块或窗体中的代码来调用。

```
Sub PlaceLine()
    CommandState.StartPrimitive New PCE_LineTest
End Sub
```

运行这段代码能演示其工作的实际情况。第一点选择后，线随着 MicroStation 中的光标的移动在绘制；选择第二点后该对象退出。通常我们不会让这个对象就这样了，我们将利用两个点做点事情：可能在两点间绘制一条线，或者编写代码将选择的两个点分成四等分段并在这些分点处绘制圆。在将来的例子中会看到这一点的。

PCE_RecTest

下一个例子利用与上一个例子中看到的相同的两个选择点。然而，不再是画线了而是以两点为边界绘制一个矩形。代码中仅有的不同是用点的 X 和 Y 分量生成形从而建立矩形。该类模块的名称是 **PCE_RecTest**。

```
Implements IPrimitiveCommandEvents
```

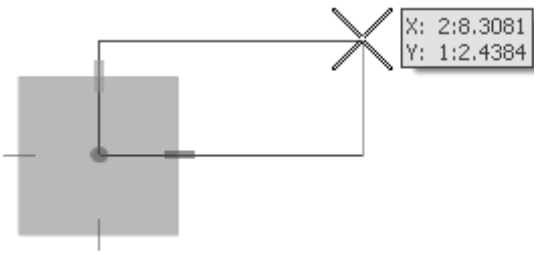
```
Dim pt3BasePoint As Point3d
Dim boolSet As Boolean
Private Sub IPrimitiveCommandEvents_Cleanup()
End Sub
Private Sub IPrimitiveCommandEvents_DataPoint(Point As Point3d, _
                                              ByVal View As View)
    If boolSet = False Then
        pt3BasePoint = Point
        CommandState.StartDynamics
        boolSet = True
    Else
        CommandState.StartDefaultCommand
    End If
End Sub
Private Sub IPrimitiveCommandEvents_Dynamics(Point As Point3d, _
        ByVal View As View, ByVal DrawMode As MsdDrawingMode)
    Dim pt3RecPoints(0 To 3) As Point3d
    Dim myShapeElem As ShapeElement
    pt3RecPoints(0) = pt3BasePoint
    pt3RecPoints(1).X = Point.X
    pt3RecPoints(1).Y = pt3BasePoint.Y
    pt3RecPoints(2) = Point
    pt3RecPoints(3).X = pt3BasePoint.X
    pt3RecPoints(3).Y = Point.Y
    Set myShapeElem = CreateShapeElement1(Nothing, pt3RecPoints)
    myShapeElem.Redraw DrawMode
End Sub
Private Sub IPrimitiveCommandEvents_Keyin(ByVal Keyin As String)
End Sub
Private Sub IPrimitiveCommandEvents_Reset()
```


China BDN

```
End Sub
Private Sub IPrimitiveCommandEvents_Start()
End Sub
```

注意形的每个顶点的 X 和 Y 分量是如何来自于基准点和当前光标点的。

```
Sub PlaceRec()
    CommandState.StartPrimitive New PCE_RecTest
End Sub
```



过程 PlaceRec 启动了 PCE_RecTest 类模块。

选择完第一点后，从第一点到光标处拖拽出一个矩形。由于未在 Reset 事件中作任何事情，所以退出该接口的唯一途径是选择第二点。

PCE_CircTest

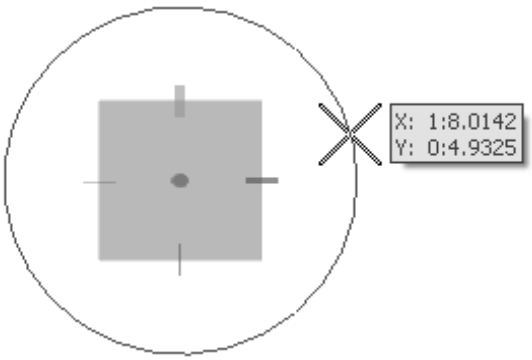
PCE_CircTest 类绘制一个以选择点为中心经过光标的圆。

```
Implements IPrimitiveCommandEvents
Dim pt3BasePoint As Point3d
Dim boolSet As Boolean
Private Sub IPrimitiveCommandEvents_Cleanup()
End Sub
Private Sub IPrimitiveCommandEvents_DataPoint(Point As Point3d, _
                                              ByVal View As View)
    If boolSet = False Then
        pt3BasePoint = Point
        CommandState.StartDynamics
        boolSet = True
    Else
        CommandState.StartDefaultCommand
    End If
End Sub
```

```
Private Sub IPrimitiveCommandEvents_Dynamics(Point As Point3d, _
        ByVal View As View, ByVal DrawMode As MsdDrawingMode)
    Dim myCircle As EllipseElement
    Dim rotMatrix As Matrix3d
    Dim dblRadius As Double
    dblRadius = Point3dDistance(pt3BasePoint, Point)
    Set myCircle = CreateEllipseElement2(Nothing, pt3BasePoint, _
                                         dblRadius, dblRadius, rotMatrix)
    myCircle.Redraw DrawMode
End Sub
Private Sub IPrimitiveCommandEvents_Keyin(ByVal Keyin As String)
End Sub
Private Sub IPrimitiveCommandEvents_Reset()
    CommandState.StartDefaultCommand
End Sub
Private Sub IPrimitiveCommandEvents_Start()
End Sub
```

本例使用了 Reset 事件，如果用户发出复位命令，程序将调用 StartDefaultCommand 退出本接口对象。

```
Sub PlaceCirc()
    CommandState.StartPrimitive New PCE_CircTest
End Sub
```



PCE_PolyTest

PolyTest 例子绘制一个正多边形，该正多边形外接于一个以第一点为中心的假象圆，以光标位置为边界。正多边形有正方形、三角形或六角形，到底应该绘制哪个呢？PolyTest 类能绘制任何正多边形，因为顶点数由我们自己来指定。类模块中的代码写得足够清楚了。调用这个类模块的方式不同于以前例子中看到的。让我们先从类模块开始：

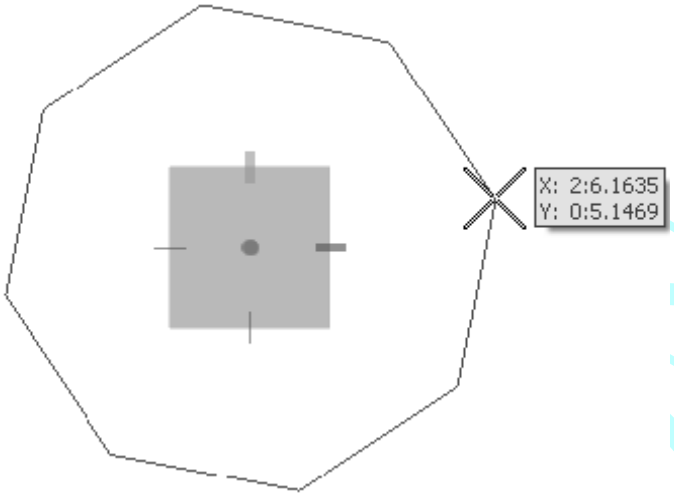
```
Implements IPrimitiveCommandEvents
Dim pt3BasePoint As Point3d
Dim boolSet As Boolean
Public Vertices As Long
Private Sub IPrimitiveCommandEvents_Cleanup()
End Sub
Private Sub IPrimitiveCommandEvents_DataPoint(Point As Point3d, _
                                              ByVal View As View)
    If boolSet = False Then
        pt3BasePoint = Point
        CommandState.StartDynamics
        boolSet = True
    Else
        CommandState.StartDefaultCommand
    End If
End Sub
Private Sub IPrimitiveCommandEvents_Dynamics(Point As Point3d, _
        ByVal View As View, ByVal DrawMode As MsdDrawingMode)
    Dim pt3PolyPoints() As Point3d
    ReDim pt3PolyPoints(0 To Vertices - 1) As Point3d
    Dim myShapeElem As ShapeElement
    Dim I As Long
    Dim dblBaseAngle As Double
    dblBaseAngle = Atn((Point.Y - pt3BasePoint.Y) / _
                       (Point.X - pt3BasePoint.X))
```

```
        For I = 0 To Verticies - 1
            pt3PolyPoints(I) = Point3dAddAngleDistance(pt3BasePoint, _
                    dblBaseAngle + Radians(360 / Verticies * I), _
                    Point3dDistance(pt3BasePoint, Point), 0)
        Next I
        Set myShapeElem = CreateShapeElement1(Nothing, pt3PolyPoints)
        myShapeElem.Redraw DrawMode
    End Sub
    Private Sub IPrimitiveCommandEvents_Keyin(ByVal Keyin As String)
    End Sub
    Private Sub IPrimitiveCommandEvents_Reset()
        CommandState.StartDefaultCommand
    End Sub
    Private Sub IPrimitiveCommandEvents_Start()
    End Sub
```

注意顶点变量是在类模块通用声明区被声明为公用变量的，这使其扮演了类模块属性的角色。

```
    Sub PlacePoly()
        Dim myPolyTest As New PCE_PolyTest
        myPolyTest.Verticies = 8
        CommandState.StartPrimitive myPolyTest
    End Sub
```

正如前面提及的，我们使用 PCE_PolyTest 类的方式与以前的类有点不同。在这个例子中，把一个变量声明为 New **PCE_PolyTest**。这样做能在开始“StartPrimitive”动作前来指定要绘制的顶点个数。在启动类和设置属性后，用带“StartPrimitive”方法的 myPolyTest 变量开始捕获 PrimitiveCommandEvents (基本命令事件)。



指定的顶点数用来计算一个角度，这个角度用于投影正多边形的各顶点。

PCE_PointStringTest

截至目前为止，每个 PrimitiveCommandEvents 接口例子都是基于用户选择的两个点。在两点间画线、把两点作为对角点绘制矩形、用两点画圆、用两点画多边形。

PCE_PointStringTest 类允许选择多个点。实际上，用户可选择的点数是没有限制的。

```
Implements IPrimitiveCommandEvents
Dim pt3BasePoint As Point3d
Dim pt3Points() As Point3d
Dim boolSet As Boolean
Private Sub IPrimitiveCommandEvents_Cleanup()
End Sub
Private Sub IPrimitiveCommandEvents_DataPoint(Point As Point3d, _
                                              ByVal View As View)
    If boolSet = False Then
        pt3BasePoint = Point
        pt3Points(0) = Point
        ReDim Preserve pt3Points(UBound(pt3Points) + 1)
        CommandState.StartDynamics
        boolSet = True
```

```
        Else
            pt3Points(UBound(pt3Points)) = Point
            ReDim Preserve pt3Points(UBound(pt3Points) + 1)
        End If
End Sub
Private Sub IPrimitiveCommandEvents_Dynamics(Point As Point3d, _
            ByVal View As View, ByVal DrawMode As MsdDrawingMode)
        Dim myPointString As PointStringElement
        pt3Points(UBound(pt3Points)) = Point
        Set myPointString = CreatePointStringElement1( _
                                Nothing, pt3Points, False)
        myPointString.Redraw DrawMode
End Sub
Private Sub IPrimitiveCommandEvents_Keyin(ByVal Keyin As String)
        Select Case UCase(Keyin)
            Case "PLCLOSE"
                Dim myPointString As PointStringElement
                pt3Points(UBound(pt3Points)) = pt3BasePoint
                Set myPointString = CreatePointStringElement1(Nothing, _
                                                        pt3Points, False)
                ActiveModelReference.AddElement myPointString
                CommandState.StartDefaultCommand
        End Select
End Sub
Private Sub IPrimitiveCommandEvents_Reset()
CommandState.StartDefaultCommand
End Sub
Private Sub IPrimitiveCommandEvents_Start()
        ReDim pt3Points(0)
End Sub
```

使用这个例子时，用户可能输入多于一或两个点。我们可以声明一个最多能容纳 10 个点或 50 个点或 100 个点的变量，然后让用户选择点数，例如：“最多 10 个点”。这样做也行或在某些条件下可能还必须这样做，但若想允许任意多个点，则需要一个不同的解决方案。

把 pt3Points 变量声明为动态数组(声明时用空括号)能在需要时改变数组的大小。用“Preserve”关键字来改变数组大小使得 VBA 在改变数组大小时不丢弃已经存在的数组元素。

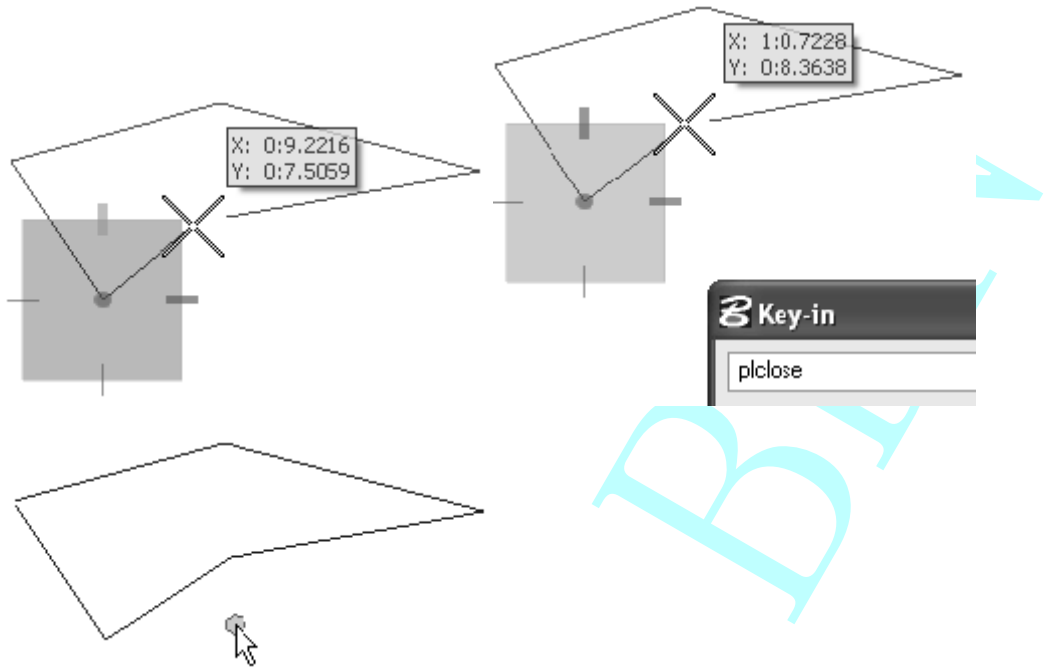
每当选择一个新点时，把选择的点放入数组的上界元素中，然后立即对数组大小加一。每次输入新点后对数组大小加一很重要，因为动态事件中要用到这个新的上界元素。

在以前的例子中，只想让用户选择两个点，这样用“StartDefaultCommand”能轻易地退出类模块，放置的复位事件工作得也很好。然而，现在想让用户在不需复位的前提下关闭点串并完成命名，怎样做到这一点呢？一个方法就是使用 Keyin(键入)事件。

```
Private Sub IPrimitiveCommandEvents_Keyin(ByVal Keyin As String)
    Select Case UCase(Keyin)
        Case "PLCLOSE"
            Dim myPointString As PointStringElement
            pt3Points(UBound(pt3Points)) = pt3BasePoint
            Set myPointString = CreatePointStringElement1(Nothing, _
                                                pt3Points, False)
            ActiveModelReference.AddElement myPointString
            CommandState.StartDefaultCommand
    End Select
End Sub
```

以上是 Keyin 事件中的代码。假如用户输入了“plclose”、“PlClose”、“PLColse”等字样，程序会将基准点放入 pt3Points 变量的上界元素中来关闭点串，然后以 pt3Points 变量为点串的顶点建立一个新的点串元素。我们看到过类似的代码但还需要添加一些以前没有做过的内容。除了建立点串元素外还将其

加入到了 ActiveModelReference 中，这使得点串成为 ActiveModelReference 中永久的部分。



从以上图中能看到，用户选择了多个点。随着点的选择，我们建立了一个点串元素但并未将其加入模型中，只是建立并显示这个点串。任何时候用户复位了命令，程序就退出该类，点串消失。当用户在键入对话框中输入“plclose”并按**<Enter>**后，我们才用选取的点建立点串元素并将其加入模型。

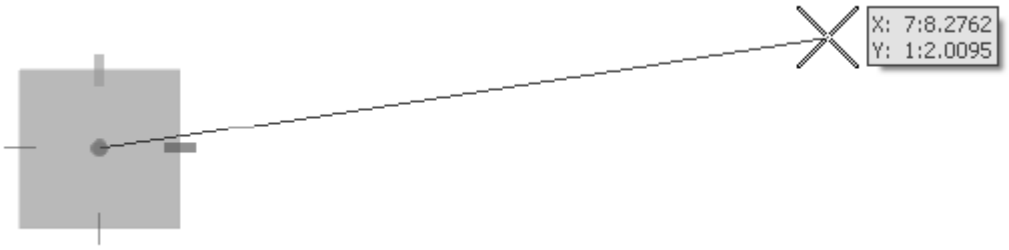
我们已经看了类模块代码，也看了类工作的结果，那么如何调用它呢？调用方法与本章其它的例子都不一样。

```
Sub PlacePointString()
    CommandState.StartPrimitive New PCE_PointStringTest, True
End Sub
```

有什么不同呢？通过给 StartPrimitive 方法的可选参数“WantKeyins”以 True(真)值来表明我们想捕获键入，该参数的缺省值为 False。所以，在使用 StartPrimitive 方法时如果想捕获键入就必须指定一个 True 值。

PCE_LineTest2

让用户选择两个点，然后将两点间距分成等长的几段并在等分点处绘制圆。我们从接口开始然后讨论代码。



选择第一点后，在选择点和光标位置间绘制一条橡皮筋线。



第二点选择后，在等分点处绘制圆。本例中指定等分份数为四段。

下面是类模块的代码：

```
Implements IPrimitiveCommandEvents
Dim pt3BasePoint As Point3d
Dim boolSet As Boolean
Public lngDivisions As Long
Private Sub IPrimitiveCommandEvents_Cleanup()
End Sub
Private Sub IPrimitiveCommandEvents_DataPoint(Point As Point3d, _
                                              ByVal View As View)
    If boolSet = False Then
        pt3BasePoint = Point
        CommandState.StartDynamics
        boolSet = True
    Else
        Dim pt3EndPoint As Point3d
        Dim dblLineAngle As Double
        Dim dblSegDist As Double
        Dim DivPoints() As Point3d
        Dim I As Long
        ReDim DivPoints(0 To lngDivisions - 2) As Point3d
```

```
            pt3EndPoint = Point
            dblLineAngle = Atn((pt3EndPoint.Y - pt3BasePoint.Y) / _
                                        (pt3EndPoint.X - pt3BasePoint.X))
            dblSegDist = Point3dDistanceXY(pt3BasePoint, pt3EndPoint) / _
                                        lngDivisions
            For I = LBound(DivPoints) To UBound(DivPoints)
                DivPoints(I) = Point3dAddAngleDistance(pt3BasePoint, _
                                    dblLineAngle, dblSegDist * (I + 1), 0)
            Next I
            DrawCircle pt3BasePoint, 0.25
            For I = LBound(DivPoints) To UBound(DivPoints)
                DrawCircle DivPoints(I), 0.25
            Next I
            DrawCircle pt3EndPoint, 0.25
            CommandState.StartDefaultCommand
        End If
End Sub
Private Sub DrawCircle(CenPt As Point3d, Radius As Double)
        Dim myEllipse As EllipseElement
        Dim rotMatrix As Matrix3d
        Set myEllipse = CreateEllipseElement2(Nothing, CenPt, Radius, Radius, _
                                        rotMatrix)
        ActiveModelReference.AddElement myEllipse
End Sub
Private Sub IPrimitiveCommandEvents_Dynamics(Point As Point3d, _
            ByVal View As View, ByVal DrawMode As MsdDrawingMode)
        Dim myLineElem As LineElement
        Set myLineElem = CreateLineElement2(Nothing, pt3BasePoint, Point)
        myLineElem.Redraw DrawMode
End Sub
```

```
Private Sub IPrimitiveCommandEvents_Keyin(ByVal Keyin As String)
End Sub
Private Sub IPrimitiveCommandEvents_Reset()
    CommandState.StartDefaultCommand
End Sub
Private Sub IPrimitiveCommandEvents_Start()
End Sub
```

仔细查看以上代码能发现一个叫做“DrawCircle”的方法，每当想绘制圆时就使用这个方法。把特定的功能明确的代码独立为一个过程能使 DataPoint 事件清晰一些。

下面是两个使用 PCE_LineTest2 类的例子：

```
Sub PlaceLine2A()
    Dim myLineTest2 As New PCE_LineTest2
    myLineTest2.lngDivisions = 4
    CommandState.StartPrimitive myLineTest2
End Sub
Sub PlaceLine2B()
    Dim myLineTest2 As New PCE_LineTest2
    myLineTest2.lngDivisions = 12
    CommandState.StartPrimitive myLineTest2
End Sub
```

过程 PlaceLine2A 将选择的两点间距分成四等份，过程 PlaceLine2B 将选择的两点间距分成 12 等份。

PCE_LineTest3

到目前为止，我们的目标是学习如何使用 IprimitiveCommandEvents 对象。前几个例子实现了在让用户选择点后显示线、圆和多边形，最近的例子是把选择的两点间距平分成指定的段数并在分段点处放置圆。下个例子将扩展 PCE_LineTest2 类。

PCE_LineTest2 用于在分段点处绘制指定半径的圆。如果想在分段点处绘

制正方形，可以建立一个新类，从 PCE_LineTest2 复制并粘贴代码，然后修改新类以绘制正方形；要绘制六边形也可以如法炮制：建立新类，复制并粘贴，然后修改代码；要绘制八边形，也是建立新类，复制并粘贴，然后修改代码。我们能建立一百个新的类模块，每个类在分段点处绘制一种元素。或者这样做：建立一个新类，复制并粘贴，然后只修改代码一次，以后可以反复使用这个新类。

PCE_LineTest3 基于 PCE_LineTest2，但不是在分段点绘制圆而是把分段点放入一个变量中，调用该类的过程能使用这个变量。让我们先看一些如何使用新类的例子，然后考察类本身。

```
Sub PlaceLine3A()
    Dim myDivPoints() As Point3d
    Dim myLineTest3 As New PCE_LineTest3
    Dim I As Long
    myLineTest3.lngDivisions = 12
    CommandState.StartPrimitive myLineTest3
    While myLineTest3.ClassComplete = False
        DoEvents
    Wend
    myDivPoints = myLineTest3.DivPts
    For I = LBound(myDivPoints) To UBound(myDivPoints)
        Dim myEllipse As EllipseElement
        Dim rotMatrix As Matrix3d
        Set myEllipse = CreateEllipseElement2(Nothing, myDivPoints(I), _
                                              0.25, 0.25, rotMatrix)
        ActiveModelReference.AddElement myEllipse
    Next I
End Sub
```

让我们分段来讨论这个过程。

1、声明变量，用 New 关键字启动 PCE_LineTest3 类。

```
    Dim myDivPoints() As Point3d
    Dim myLineTest3 As New PCE_LineTest3
```

```
    Dim I As Long
```

2、通过设置类的 lngDivisions 属性来设置分段数。

```
    myLineTest3.lngDivisions = 12
```

3、启动 PrimitiveCommandEvents 对象。

```
    CommandState.StartPrimitive myLineTest3
```

4、考察类 **PCE_LineTest3** 的 ClassComplete 属性，当 ClassComplete 属性为假时允许用户继续交互。

```
    While myLineTest3.ClassComplete = False
        DoEvents
    Wend
```

5、取得由选择的两点生成的一系列分段点。

```
    myDivPoints = myLineTest3.DivPts
```

6、在 myDivPoints 数组的每个点上绘制半径为 0.25 的圆。

```
For I = LBound(myDivPoints) To UBound(myDivPoints)
    Dim myEllipse As EllipseElement
    Dim rotMatrix As Matrix3d
    Set myEllipse = CreateEllipseElement2(Nothing, myDivPoints(I), _
                                          0.25, 0.25, rotMatrix)
    ActiveModelReference.AddElement myEllipse
Next I
```

这就是调用代码。对 PCE_LineTest3 作出改变的一个好处是，我们使用它来返回分段点。类返回分段点后，用这些点能做任何事情。下面是使用 **PCE_LineTest3** 的另一个例子：

```
Sub PlaceLine3B()
    Dim myDivPoints() As Point3d
    Dim myLineTest3 As New PCE_LineTest3
    Dim I As Long
    myLineTest3.lngDivisions = 16
    CommandState.StartPrimitive myLineTest3
    While myLineTest3.ClassComplete = False
```


China BDN

```
        DoEvents
    Wend
    myDivPoints = myLineTest3.DivPts
    For I = LBound(myDivPoints) To UBound(myDivPoints) – 1
        Dim myLineElem As LineElement
        Set myLineElem = CreateLineElement2(Nothing, myDivPoints(I), _
                                            myDivPoints(I + 1))
        ActiveModelReference.AddElement myLineElem
    Next I
End Sub
```

通读 PlaceLine3B 中的代码。它用类 **PCE_LineTest3** 返回的点做什么了？

PlaceLine3B 在类 **PCE_LineTest3** 指定的每一段上画线。

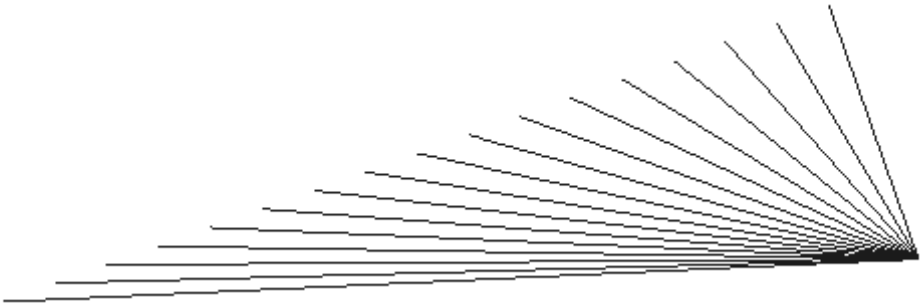
让我们再多看一个例子：

```
Sub PlaceLine3C()
    Dim myDivPoints() As Point3d
    Dim myLineTest3 As New PCE_LineTest3
    Dim I As Long
    Dim LineBasePt As Point3d
    myLineTest3.lngDivisions = 16
    CommandState.StartPrimitive myLineTest3
    While myLineTest3.ClassComplete = False
        DoEvents
    Wend
    myDivPoints = myLineTest3.DivPts
    LineBasePt.X = 3: LineBasePt.Y = 4: LineBasePt.Z = 5
    For I = LBound(myDivPoints) To UBound(myDivPoints)
        Dim myLineElem As LineElement
        Set myLineElem = CreateLineElement2(Nothing, LineBasePt, _
                                            myDivPoints(I))
        ActiveModelReference.AddElement myLineElem
```


China BDN

```
    Next I
End Sub
```

这段代码做了什么？当然，我们使用了类 **PCE_LineTest3**，但用返回点干了什么事情？



我们从每个分段点向同一个基准点绘制直线。

在使用类 **PCE_LineTest3** 的每个例子中，都用了一个 While ... Wend 结构来让用户选择两个点。两个点被选取后，“ClassComplete”的值不再为假，我们就可以利用返回的点了。每个例子都在没有修改类模块的前提下工作得很好，这正是我们最期望的情形：一个类模块能不用修改地在多种环境下使用。

回顾以上代码能看出：这个类模块有三个属性。一个叫“lngDivisions”，另一个叫“ClassComplete”，最后一个叫“DivPts”。

我们已经看过了如何使用 **PCE_LineTest3**。现在来看一看类模块背后的代码。

```
Implements IPrimitiveCommandEvents
Dim pt3BasePoint As Point3d
Dim boolSet As Boolean
Public lngDivisions As Long
Public DivPts As Variant
Public ClassComplete As Boolean
Private Sub IPrimitiveCommandEvents_Cleanup()
End Sub
Private Sub IPrimitiveCommandEvents_DataPoint(Point As Point3d, _
                                              ByVal View As View)
    If boolSet = False Then
```

```
            pt3BasePoint = Point
            CommandState.StartDynamics
            boolSet = True
        Else
            Dim pt3EndPoint As Point3d
            Dim dblLineAngle As Double
            Dim dblSegDist As Double
            Dim DivPoints() As Point3d
            Dim I As Long
            ReDim DivPoints(0 To lngDivisions) As Point3d
            DivPoints(0) = pt3BasePoint
            DivPoints(UBound(DivPoints)) = Point
            pt3EndPoint = Point
            dblLineAngle = Atn((pt3EndPoint.Y - pt3BasePoint.Y) / _
                               (pt3EndPoint.X - pt3BasePoint.X))
            dblSegDist = Point3dDistanceXY(pt3BasePoint, pt3EndPoint) / _
                               lngDivisions
            For I = LBound(DivPoints) + 1 To UBound(DivPoints) - 1
                DivPoints(I) = Point3dAddAngleDistance(pt3BasePoint, _
                                   dblLineAngle, dblSegDist * (I), 0)
            Next I
            DivPts = DivPoints
            ClassComplete = True
            CommandState.StartDefaultCommand
        End If
End Sub
Private Sub IPrimitiveCommandEvents_Dynamics(Point As Point3d, _
            ByVal View As View, ByVal DrawMode As MsdDrawingMode)
        Dim myLineElem As LineElement
        Set myLineElem = CreateLineElement2(Nothing, pt3BasePoint, Point)
```


China BDN

```
    myLineElem.Redraw DrawMode
End Sub
Private Sub IPrimitiveCommandEvents_Keyin(ByVal Keyin As String)
End Sub
Private Sub IPrimitiveCommandEvents_Reset()
    CommandState.StartDefaultCommand
End Sub
Private Sub IPrimitiveCommandEvents_Start()
End Sub
```

在以前的类 **PCE_LineTest2** 中已经使用过这段代码的大部分，在此让我们集中于 DataPoint 事件。该事件以前是绘制圆，现在则是向数组中放点，随后把该数组赋给一个声明为公共 variant 类型的变量。在类模块中把一个变量声明为公共变量，就能使该变量像对象的一个属性一样来使用。除了把坐标点放入 **DivPts** 变量中外，我们还设置 **ClassComplete** 变量为真。这个 **ClassComplete** 变量被调用 **PCE_LineTest3** 的过程读取，该变量位于 While ... Wend 结构中。虽然前面讨论过 PlaceLine3A，下面再次出现该过程：

```
Sub PlaceLine3A()
    Dim myDivPoints() As Point3d
    Dim myLineTest3 As New PCE_LineTest3
    Dim I As Long
    myLineTest3.lngDivisions = 12
    CommandState.StartPrimitive myLineTest3
    While myLineTest3.ClassComplete = False
        DoEvents
    Wend
    myDivPoints = myLineTest3.DivPts
    For I = LBound(myDivPoints) To UBound(myDivPoints)
        Dim myEllipse As EllipseElement
        Dim rotMatrix As Matrix3d
        Set myEllipse = CreateEllipseElement2(Nothing, myDivPoints(I), _
```

```
                                        0.25, 0.25, rotMatrix)
            ActiveModelReference.AddElement myEllipse
        Next I
End Sub
```

一旦使用了 StartPrimitive，就开始查看 ClassComplete 属性，一直等到 ClassComplete 为真。当 ClassComplete 为真时，从类的 DivPts 属性中得到分段点。

优化 Dynamics(动态)事件

Dynamics 事件非常令人激动。每当鼠标移动非常小的距离，Dynamics 事件中的代码就会被执行。这意味着什么？需要 VBA 代码迅速执行！

如何能实现“迅速执行”？这是个好问题。有一个测试演示了把光标从 MicroStation 窗口的左边移动到右边，这引起 Dynamics 事件执行了超过 390 次。这表明我们需要非常注意放在动态事件中的代码。

在动态事件中放入简单且快速执行的代码不会产生问题。然而，尝试复杂计算或执行其它耗时操作的代码可能会出现问题。虽然本章中演示的例子都运行良好，但在处理动态事件时还是需要尽量减少处理器的负荷。

```
Private Sub IPrimitiveCommandEvents_Dynamics(Point As Point3d, _
                    ByVal View As View, ByVal DrawMode As MsdDrawingMode)
    Dim myLineElem As LineElement
    Set myLineElem = CreateLineElement2(Nothing, pt3BasePoint, Point)
    myLineElem.Redraw DrawMode
End Sub
```

每当触发动态事件时做了如下工作：

1、声明了一个 LineElement 变量；

2、建立了一个 LineElement 元素；

3、重画 LineElement。

事件中有三行代码——每行都会占用处理器时间。我们在代码中看不到的是，当退出动态事件后 LineElement 就超出了其作用范围。这将花费时间，因为 VBA 必须清理已经赋给对象的内存。设想如下情形：仅仅因为光标从左移到了

右就需要设置内存、绘制直线然后再清理内存超过 390 次。

让我们比较“PCE_LineTest”与“PCE_LineTest4”，在 PCE_LineTest4 中以不同的方式实现了 Dynamics 事件。

PCE_LineTest

```
Implements IPrimitiveCommandEvents
Dim pt3BasePoint As Point3d
Dim boolSet As Boolean
Private Sub IPrimitiveCommandEvents_Cleanup()
End Sub
Private Sub IPrimitiveCommandEvents_DataPoint(Point As Point3d, _
                                              ByVal View As View)
    If boolSet = False Then
        pt3BasePoint = Point
        CommandState.StartDynamics
        boolSet = True
    Else
        CommandState.StartDefaultCommand
    End If
End Sub
Private Sub IPrimitiveCommandEvents_Dynamics(Point As Point3d, _
            ByVal View As View, ByVal DrawMode As MsdDrawingMode)
    Dim myLineElem As LineElement
    Set myLineElem = CreateLineElement2(Nothing, pt3BasePoint, Point)
    myLineElem.Redraw DrawMode
End Sub
Private Sub IPrimitiveCommandEvents_Keyin(ByVal Keyin As String)
End Sub
Private Sub IPrimitiveCommandEvents_Reset()
End Sub
Private Sub IPrimitiveCommandEvents_Start()
```

```
End Sub
```

每当动态事件触发时就声明变量、建立直线、重画直线然后再结束该变量(因为超出了作用范围)。现在让我们看一看 PCE_LineTest 与 PCE_LineTest4 之间的不同。

PCE_LineTest4

```
Implements IPrimitiveCommandEvents
Dim pt3BasePoint As Point3d
Dim boolSet As Boolean
Dim myLineElem As LineElement
Private Sub IPrimitiveCommandEvents_Cleanup()
End Sub
Private Sub IPrimitiveCommandEvents_DataPoint(Point As Point3d, _
                                              ByVal View As View)
    If boolSet = False Then
        pt3BasePoint = Point
        Set myLineElem = CreateLineElement2(Nothing, Point, Point)
        CommandState.StartDynamics
        boolSet = True
    Else
        CommandState.StartDefaultCommand
    End If
End Sub
Private Sub IPrimitiveCommandEvents_Dynamics(Point As Point3d, _
            ByVal View As View, ByVal DrawMode As MsdDrawingMode)
    myLineElem.Vertex(1) = Point
    myLineElem.Redraw DrawMode
End Sub
Private Sub IPrimitiveCommandEvents_Keyin(ByVal Keyin As String)
End Sub
Private Sub IPrimitiveCommandEvents_Reset()
```


China BDN

```
End Sub
Private Sub IPrimitiveCommandEvents_Start()
End Sub
```

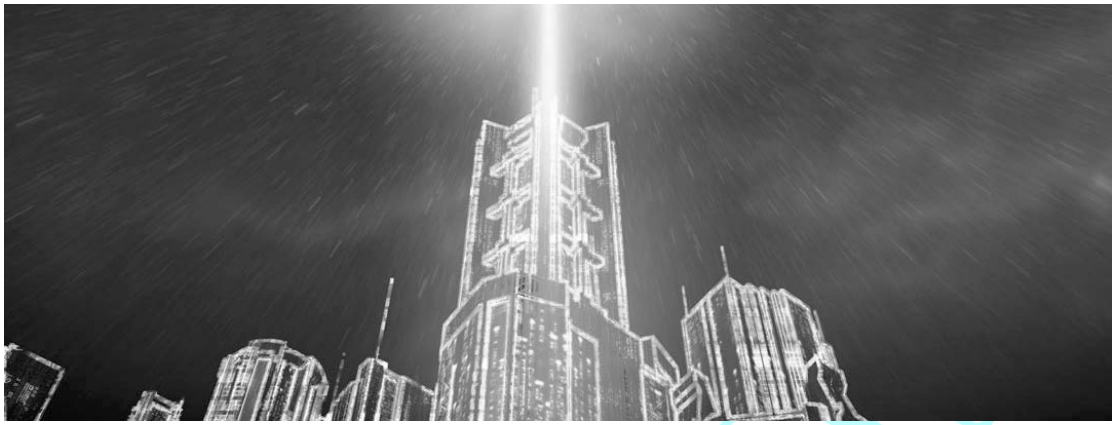
在 PCE_LineTest4 中，我们在通用声明区声明了 LineElement 变量，在 DataPoint 事件中执行了一次建立直线操作。当进入 Dynamics 事件时，仅仅是改变线的终点并重画之。这是使用 Dynamics 事件非常有效的方法。

PCE_LineTest 和 PCE_LineTest4 既简单又小。虽然在使用这两个类时效果不明显，但 PCE_LineTest4 还是极大地减少了处理器时间。像我们对 PCE_LineTest4 所作的小改变可能不会对代码效率有一个立即的提升，但微小效率的提升累积起来就是显著的性能提高。反之亦然，多处低效率累积起来就是极大的性能降低。

本章回顾

用类模块实现接口。每个接口的属性或方法必须在类模块中声明。当我们实现了本章讨论的接口时，随着应用更加交互，我们的程序也更具有控制力、更灵活、更强大。

China BDN



19. 使用 MicroStation 内置用户窗体

提高开发程序速度的一种途径就是使用现有的代码、对象和接口。

本章涉及如下内容：

- 声明 MicroStation 用户窗体函数
- FileOpen(文件打开)对话框
- FileCreate(文件建立)对话框
- FileCreateFromSeed(由种子文件建立文件)对话框
- OpenAlert(打开警告)对话框
- OpenInfoBox(打开信息框)对话框

声明 MicroStation 用户窗体函数

使用 DLL(动态链接库)能访问 MicroStation 内置的用户窗体函数。当函数包装在一个 DLL 中时，使用前必须声明这个函数，这些声明应该在代码模块的通用声明区。

```
Declare Function mdlDialog_openAlert Lib _
"stdmdlbltin.dll" (ByVal stringP As String) As Long
```

这是对 OpenAlert 对话框的声明。让我们把这个声明分成几个独立的部分：

Declare Function	告诉 VBA 将要声明一个位于 DLL 文件中的函数
mdlDialog_openAlert	声明的函数名
Lib "stdmdlbltin.dll"	包含函数的 DLL 库名
(ByVal stringP As String)	函数参数。可以为空()或一到多个参数

As Long	函数的返回类型

函数的特征之一是有返回值。“mdlDialog_openAlert”返回一个长整型值。DLL 中的有些函数需要返回多个值，通过改变函数参数的变量值来做到这一点。

下面是 FileOpen 函数的声明：

```
Declare Function mdlDialog_fileOpen Lib _
"stdmdlbltin.dll" (ByVal fileName As String, ByVal rFileH As Long, ByVal _
resourceId As Long, ByVal suggestedFileName As String, _
ByVal filterString As String, ByVal defaultDirectory As String, _
ByVal titleString As String) As Long
```

用 FileOpen 函数来打开一个文件。我们如何知道选择了哪个文件？用户是否点击了 Cancel(取消)按钮？通过 fileName 参数能知道选择了哪个文件。长整型返回值告诉我们是按了 OK 按钮还是 Cancel 按钮。

让我们每次考察一个对话框函数。

mdlDialog_fileOpen 函数

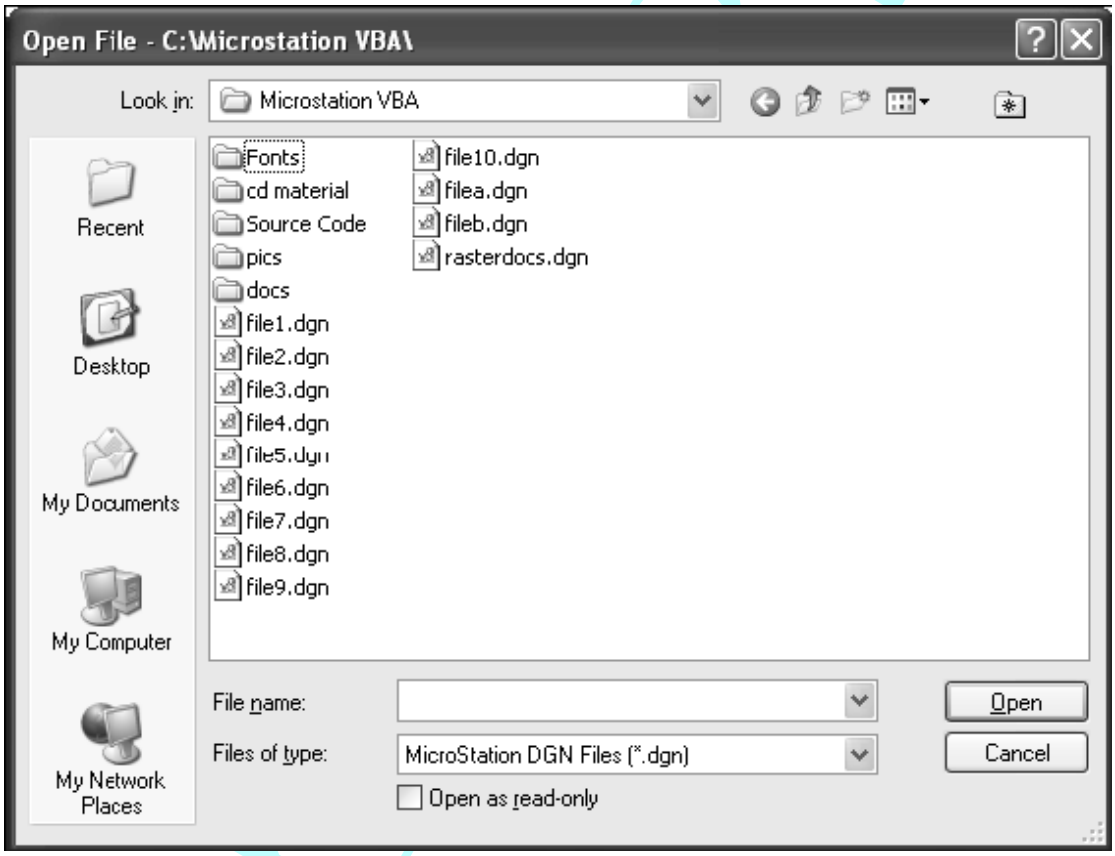
```
Declare Function mdlDialog_fileOpen Lib "stdmdlbltin.dll" (ByVal _
fileName As String, ByVal rFileH As Long, ByVal _
resourceId As Long, ByVal suggestedFileName As String, _
ByVal filterString As String, ByVal defaultDirectory As String, _
ByVal titleString As String) As Long
```

第一个例子提示用户选择一个文件。我们提供了文件扩展名、缺省目录和对话标题。

```
Sub TestFileOpenA()
    Dim strFName As String
    Dim lngfhandle As Long
    Dim lngrid As Long
    Dim retVal As Long
    strFName = Space(255)
    retVal = mdlDialog_fileOpen(strFName, lngfhandle, lngrid, _
                "", "*.dgn", "C:\MicroStation VBA", "Open File")
```

```
    Select Case retVal
        Case 0 'Open
            strFName = Left(strFName, InStr(1, strFName, Chr(0)) - 1)
            MsgBox "File Selected:" & vbCr & strFName
        Case 1 'Cancel
            MsgBox "No File Selected."
    End Select
End Sub
```

TestFileOpenA 执行时看到如下对话框：



从这个对话框能看到使用了多少个过程参数。我们能看到标题、缺省目录、过滤器(*.dgn)，也能看到没有提供一个缺省的文件名，因为对话框中的文件名是空的。

到目前为止，我们只完成了显示对话框。现在必须问几个问题。

用户点击了 Open 按钮还是 Cancel 按钮？

```
    Select Case retVal
        Case 0 'Open
```

```
        Case 1 'Cancel
    End Select
```

查看函数的返回值能知道是选择了 Open 按钮(产生返回值 0)还是选择了 Cancel 按钮(返回 1)。

如果选择了 Cancel 按钮，没有太多事情要做，因为用户取消了操作。在我们的例子中，显示一个消息框说明“No File Selected(未选择文件)”。

如果用户点击 Open 按钮，则下一个问题是“选择了哪个文件？”

当调用 FileOpen 函数时使用了 **strFName** 变量作为参数，该参数的目的就是用选择的路径和文件名来填充它。当一个变量被声明为字符串时，它是一个空的字串。此时它是一个能包含字母、数字和其它字符的变量但却是空的，直到用一个字符串填充它前它一直保持为空。所以，在把字符串变量提供给 FileOPen 函数(以及其它类似函数)之前，要用空格来填充这个变量，这样才能使函数对变量赋值并返回路径和文件名。

```
    strFName = Space(255)
```

Space 函数用给定个数的空格来填充变量。

如果向 OpenFile 函数提供一个由空格组成的变量，该变量将被选择文件的全路径来填充。提供一个有 255 个空格的变量后，即使路径和文件名只有 20 个字符的长度，它也返回一个有 255 个字符的变量。使用 Left 函数能取得第一个 Null 字符(ASCII 字符码为 0)左侧的所有字符。

```
    strFName = Left(strFName, InStr(1, strFName, Chr(0)) - 1)
    MsgBox "File Selected:" & vbCr & strFName
```



运行过程 TestFileOpenA 并选择一个要打开的文件后能看到右图所示的消息框：

我们所做的就是生成一个文件打开对话框。用户选择一个文件并点击“Open(打开)”后，该对话框并不打开这个文件而仅仅是告诉我们选择了哪个文件。打开选择的文件或执行一些其它的操作是我们的事。在第一个例子中，我们仅仅是在消息框中显示了文件名。

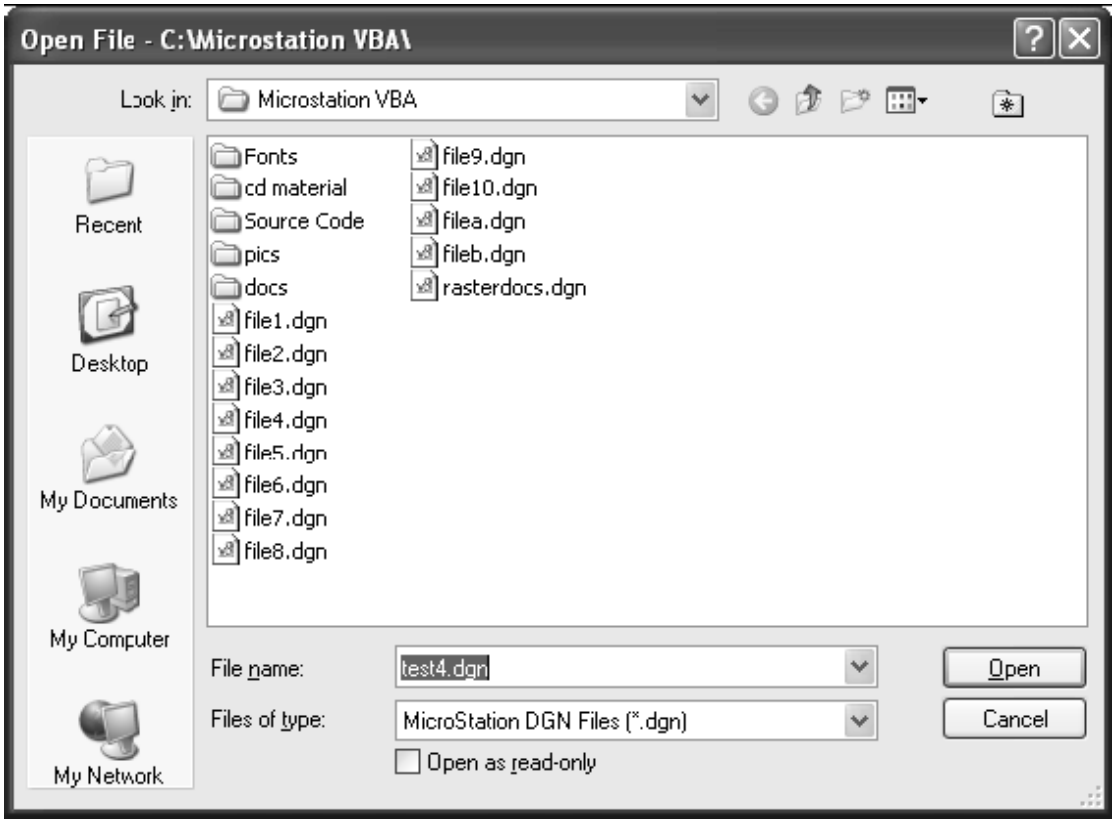
下面是对 TestFileOpenA 的一个小改动，只做了一处变动：

```
Sub TestFileOpenB()
```

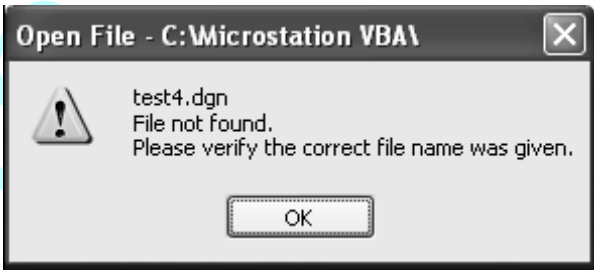
```
    Dim strFName As String
    Dim lngfhandle As Long
    Dim lngrid As Long
    Dim retVal As Long
    strFName = Space(255)
    retVal = mdlDialog_fileOpen(strFName, lngfhandle, lngrid, _
                "test4.dgn", "*.dgn", "C:\MicroStation VBA", "Open File")
    Select Case retVal
        Case 0 'Open
            strFName = Left(strFName, InStr(1, strFName, Chr(0)) - 1)
            MsgBox "File Selected:" & vbCr & strFName
        Case 1 'Cancel
            MsgBox "No File Selected."
    End Select
End Sub
```

本例提供了一个缺省的文件名 **test4.dgn**。对话框中显示了这个缺省的文件名。

China BDN



文件打开对话框用于选择现有的文件。在 TestFileOpenB 中，我们指定了一个缺省的文件名 **test4.dgn**，该文件在以上所示的文件夹中不存在。如果用户点击“Open”将看到右图所示对话框：



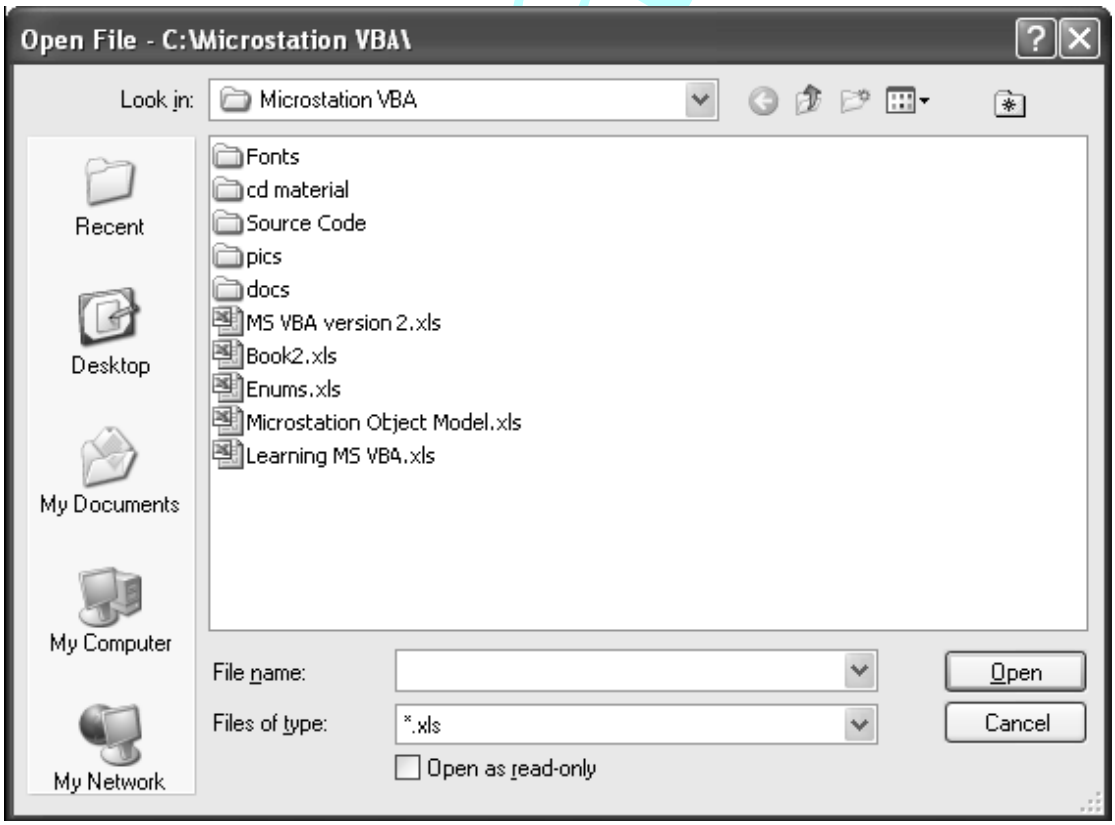
文件打开对话框不允许任何人“打开”一个不存在的文件。所以，如果我们得到了一个返回值 0，则能确信返回的文件是存在的。

前面的例子使用了文件扩展名“*.dgn”，也叫作文件过滤器。MicroStation 明白这个扩展名是“MicroStation DGN File”并将该信息显示在“File of type(文件类型)”复合框中。TestFileOpenC 使用文件过滤器“*.xls”，该过滤器在对话框中显示 Microsoft Excel 文件。

```
Sub TestFileOpenC()
    Dim strFName As String
    Dim lngfhandle As Long
    Dim lngrid As Long
```

```
    Dim retVal As Long
    strFName = Space(255)
    retVal = mdlDialog_fileOpen(strFName, lngfhandle, lngrid,  "", _
                "*.xls", "C:\MicroStation VBA", "Open File")
    Select Case retVal
        Case 0 'Open
            strFName = Left(strFName, InStr(1, strFName, Chr(0)) - 1)
            MsgBox "File Selected:" & vbCr & strFName
        Case 1 'Cancel
            MsgBox "No File Selected."
            ' User hit the Cancel Button
    End Select
End Sub
```

现在不是显示 MicroStation DGN 文件了，而是显示 Microsoft Excel(XLS 文件)文件了。



让我们看一个显示多个文件类型的例子。

```
Sub TestFileOpenD()
    Dim strFName As String
    Dim lngfhandle As Long
    Dim lngrid As Long
    Dim retVal As Long
    strFName = Space(255)
    retVal = mdlDialog_fileOpen(strFName, lngfhandle, lngrid,  "", _
                "*.xls;*.mdb;*.dbf", "C:\MicroStation VBA", "Open File")
    Select Case retVal
        Case 0 'Open
            strFName = Left(strFName, InStr(1, strFName, Chr(0)) - 1)
            MsgBox "File Selected:" & vbCr & strFName
        Case 1 'Cancel
            MsgBox "No File Selected."
            ' User hit the Cancel Button
    End Select
End Sub
```

现在，除了 Microsoft Excel 文件外，还允许选择 Access 数据库和.dbf 数据库文件。

最后一个例子不仅选择一个 DGN 文件而且在内存(不是在 MicroStation 窗口)中打开该文件为程序所用。

```
Sub TestFileOpenE()
    Dim strFName As String
    Dim lngfhandle As Long
    Dim lngrid As Long
    Dim retVal As Long
    strFName = Space(255)
    retVal = mdlDialog_fileOpen(strFName, lngfhandle, lngrid, _
                        "", "*.dgn", "C:\MicroStation VBA", "Open File")
    Select Case retVal
```

```
        Case 0 'Open
            strFName = Left(strFName, InStr(1, strFName, Chr(0)) - 1)
            Dim myDesFile As DesignFile
            Set myDesFile = OpenDesignFileForProgram(strFName, True)
            MsgBox "Do something with this file."
            myDesFile.Close
        Case 1 'Cancel
            MsgBox "No File Selected."
    End Select
End Sub
```

文件“为程序”而打开后，显示了一个消息框就关闭了。

mdlDialog_fileCreate 函数

就像 mdlDialog_fileOpen 不实际打开文件一样，mdlDialog_fileCreate 也不实际建立一个新文件，它仅告知用户选择或输入的文件名。在文件已经存在的情况下，mdlDialog_fileCreate 返回文件名和返回值前会提示用户是否覆盖该文件。

下面是 FileCreate 函数的声明，它看起来和执行起来都很像 FileOpen。

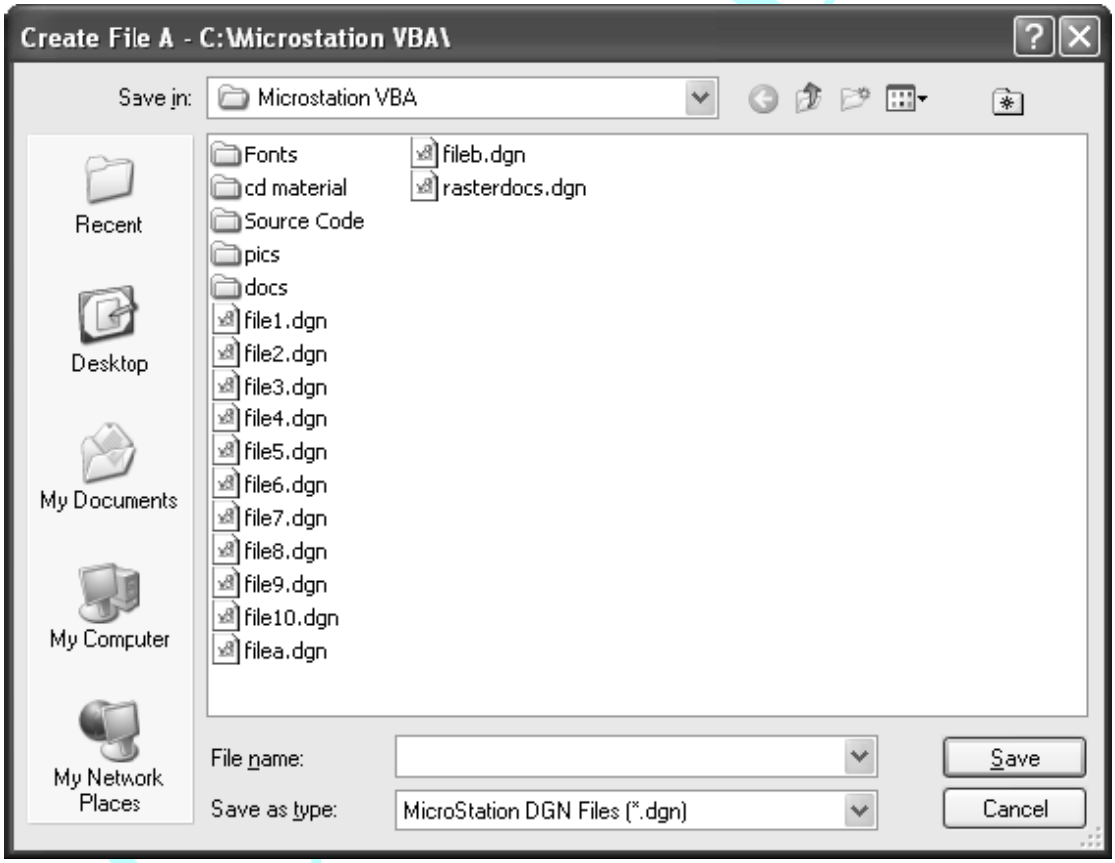
```
Declare Function mdlDialog_fileCreate Lib _
"stdmdlbltin.dll" (ByVal fileName As String, ByVal rFileH As Long, _
ByVal resourceId As Long, ByVal suggestedFileName As String, _
ByVal filterString As String, ByVal defaultDirectory As String, _
ByVal titleString As String) As Long
```

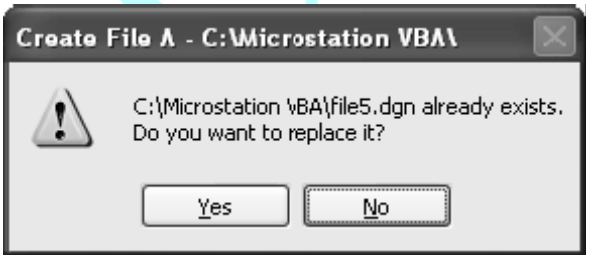
看一看如何使用这个函数。

```
Sub TestFileCreateA()
    Dim strFName As String
    Dim lngfhandle As Long
    Dim lngrid As Long
    Dim retVal As Long
    strFName = Space(255)
    retVal = mdlDialog_fileCreate(strFName, lngfhandle, lngrid, _
```

```
"", "*.dgn", "C:\MicroStation VBA", "Create File A")
    Select Case retVal
        Case 0 'Open
            strFName = Left(strFName, InStr(1, strFName, Chr(0)) - 1)
            MsgBox "File Selected:" & vbCr & strFName
        Case 1 'Cancel
            MsgBox "No File Selected."
            'User hit the Cancel Button
    End Select
End Sub
```

FileCreate 对话框与 FileOpen 对话框非常相似。





建立文件对话框的返回值含义也与文件打开对话框相同。如果选择的文件已经存在，会提问用户该文件是否要被覆盖。

如果文件存在，那么要想返回文件名和返回值 0，就需要用户点击“Yes”来覆盖现有的文件。

在文件过滤器参数中放入其它的文件扩展名将能显示与之关联的文件。

mdlDialog_fileCreateFromSeed 函数

在使用 MicroStation 标准的 **File > New** 对话框时，我们能选择一个种子文件。mdlDialog_fileCreateFromSeed 就能提供与之相同的功能。

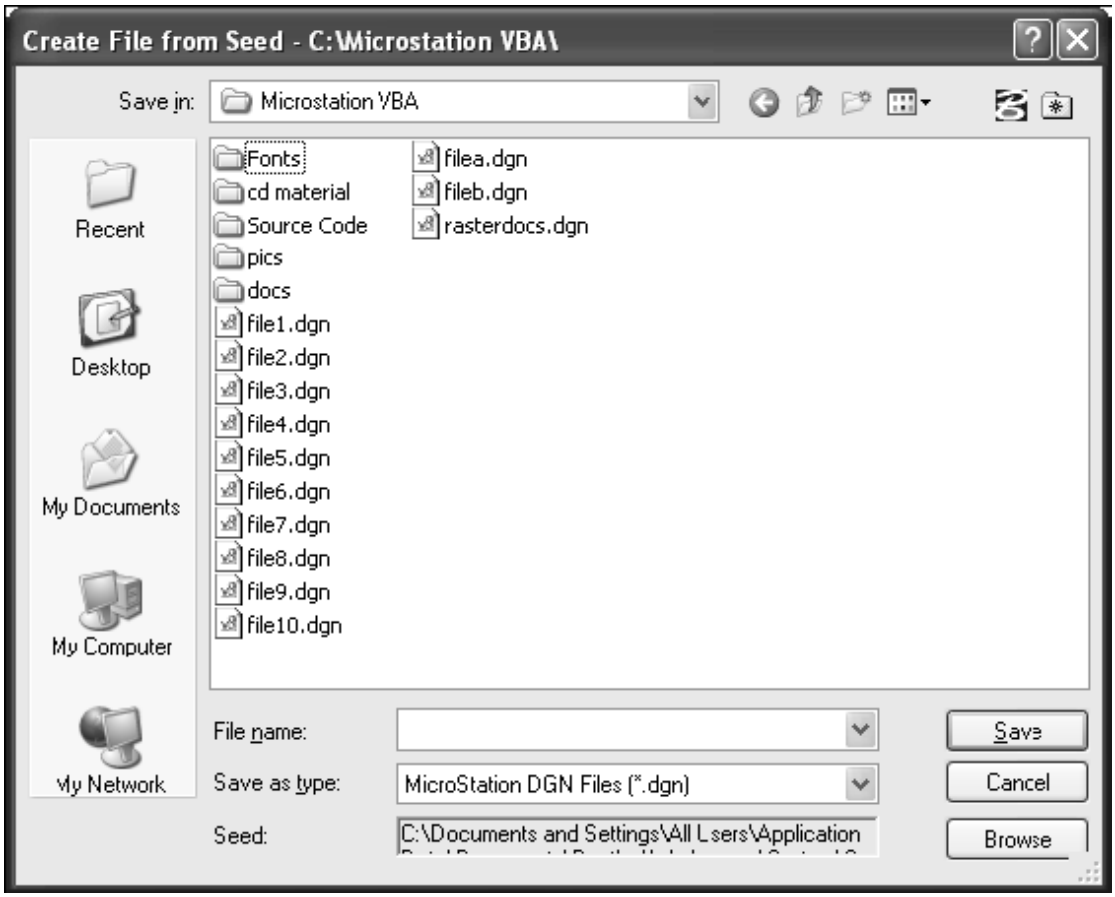
```
Declare Function mdlDialog_fileCreateFromSeed Lib _
"stdmdlbltin.dll" (ByVal fileName As String, _
ByVal rFileH As Long, ByVal resourceId As Long, _
ByVal suggestedFileName As String, ByVal filterString As String, _
ByVal defaultDirectory As String, ByVal titleString As String, _
ByVal seedFile As String, ByVal seedDirectory As String, _
ByVal seedFilter As String) As Long
```

现在来测试一下 CreateFromSeed 函数。

```
Sub TestFileCreateFromSeedA()
    Dim strFName As String
    Dim lngfhandle As Long
    Dim lngrid As Long
    Dim retVal As Long
    Dim strSeedFile As String
    Dim strSeedDir As String
    Dim strSeedFilter As String
    strFName = Space(255)
    strSeedFile = "seed2d.dgn"
    strSeedDir = _
            "C:\Documents and Settings\All Users\Application " & _
            "Data\Documents\Bentley\Workspace\System\Seed\"
    strSeedFilter = "*.dgn"
    retVal = mdlDialog_fileCreateFromSeed(strFName, lngfhandle, _
```

```
            lngrid, "", "*.dgn", "C:\MicroStation VBA", _
            "Create File from Seed", strSeedFile, strSeedDir, strSeedFilter)
    Select Case retVal
        Case 0 'Open
            strFName = Left(strFName, InStr(1, strFName, Chr(0)) - 1)
            MsgBox "File Selected:" & vbCr & strFName
        Case 1 'Cancel
            MsgBox "No File Selected."
            'User hit the Cancel Button
    End Select
End Sub
```

除了能选择要建立的文件外，选择种子文件的按钮也出现在对话框中。



mdlDialog_openAlert 函数

mdlDialog_openAlert 函数生成一个标准的 MicroStation 对话框，允许用户

选择“OK(确定)”或“Cancel(取消)”。

```
Declare Function mdlDialog_openAlert Lib _
"stdmdlbltin.dll"    (ByVal stringP As String) As Long
```

惟一的参数指定在对话框中要显示的内容，返回值表明用户按了 OK 按钮还是 Cancel 按钮。

```
Sub TestOpenAlertA()
    Dim retVal As Long
    retVal = mdlDialog_openAlert("Standard Message Box")
    Select Case retVal
        Case 3 'OK
            MsgBox "User clicked 'OK'"
        Case 4 'Cancel
            MsgBox "User clicked 'Cancel'"
    End Select
End Sub
```

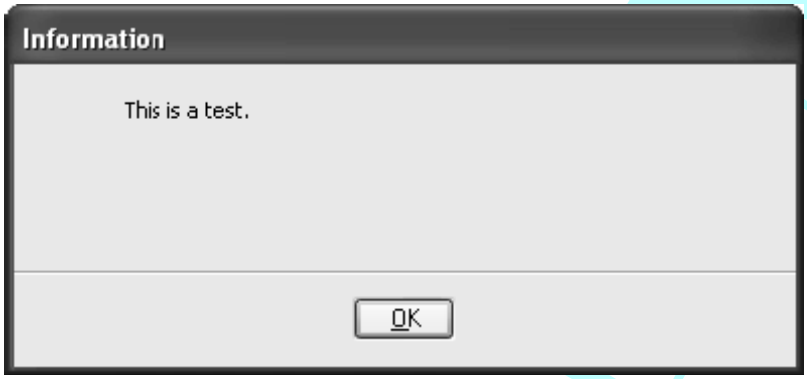


mdlDialog_openInfoBox 函数

信息对话框用来提供信息，所以不需要 Cancel 按钮，唯一显示的按钮是 OK 按钮。但因其比 VBA 的消息框更接近于 MicroStation 的本身界面而有用。

```
    Declare Function mdlDialog_openInfoBox Lib _
    "stdmdlbltin.dll" (ByVal stringP As String) As Long
Sub TestOpenInfoBox()
    Dim retVal As Long
    retVal = mdlDialog_openInfoBox("This is a test.")
```

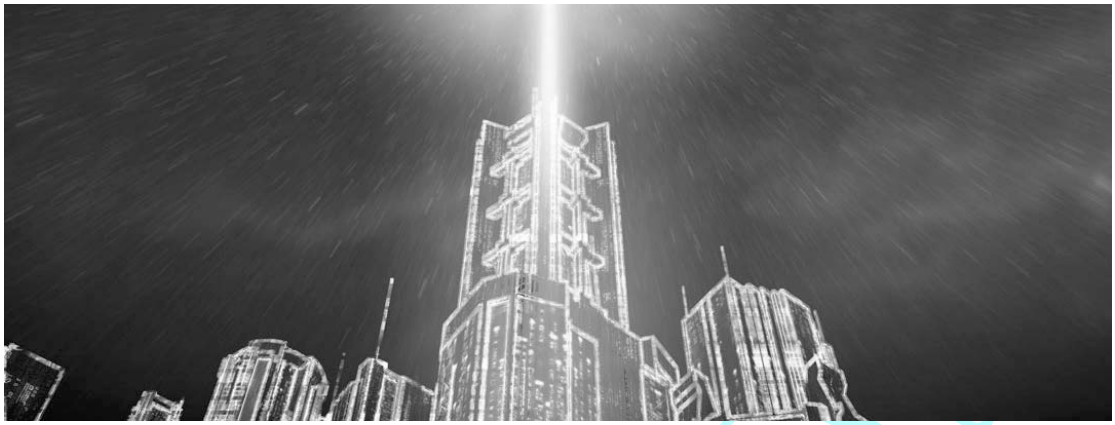
```
        Select Case retVal
            Case 3 'OK
                MsgBox "User clicked 'OK'"
        End Select
    End Sub
```



本章回顾

声明了显示标准 MicroStation 对话框的函数后，使用起来是很简单的。还有其它途径显示文件打开、文件建立类型的对话框(如使用 Windows API)，但在 VBA 中开发时使用标准的 MicroStation 对话框是最佳的方法，而且实现起来又是如此的简单。

China



20. 类模块

类模块有多种用途，主要用于：

- 封装相似功能到单个对象中；
- 建立带有属性、方法和事件的对象；
- 特为自定义集合建立类模块。

以上每个主题都是本章关注的焦点。编写的代码将定位于对 MicroStation 内置对话框以及其它 MicroStation 特有的对象和功能的使用上。

本章涉及：

- 封装相似功能
- 建立带有属性、方法和事件的对象
- 在集合中使用类模块

封装相似功能

我们能为如下目的设计类：把相似功能封装在单个类或对象中以便重用代码，这比重写代码要快。让我们以一个叫做 **clsUStationDialog** 的类开始，该类用于显示在 FileOpen 和 FileCreate 一章中使用的对话框。

在分析代码之前，先指出这个类能做什么事情。

- 显示打开 MicroStation DGN 文件的 FileOpen 对话框；
- 显示打开 Microsoft Excel 文件的 FileOpen 对话框；
- 显示打开 ASCII .txt 文件的 FileOpen 对话框；
- 显示打开用户指定扩展名的 FileOpen 对话框；

- 文件名需要的属性；
- 仅路径需要的属性；
- 路径/文件名需要的属性；
- 被选择文件的大小需要的属性；
- 注册用于保存最近访问过的路径的项；
- 注册用于保存最近使用过的文件的项。

以上每个任务在本书中都已经实现过，这个练习的主要目的是将它们包装到单个类中。

我们想在对话框中显示.dgn 文件、.xls 文件和.txt 文件，也想显示多个自定义的文件扩展名。可以设计“OpenDGN”方法、“OpenXLS”方法、“OpenTXT”方法和“OpenCustom”方法，这些方法之间的主要不同点是提供的文件扩展名不一样。所以，不要给要浏览的每种文件类型建立一个新方法，我们只建立一个叫做 OpenDialog 的方法，它处理任何文件扩展名。

先从文件扩展名着手。使用 VBA 时通常有多种途径来完成同样的任务。在我们的类中将使用一个动态数组来保存所期望的文件扩展名。我们应能让用户(在此情况下，我们是开发者)添加文件扩展名和清除文件扩展名列表。

```
'General Declarations
Dim pFileExts() As String
Private Sub Class_Initialize()
    ReDim pFileExts(0)
End Sub
Public Sub ClearFileExts()
    ReDim pFileExts(0)
End Sub
Public Sub AddFileExt(FileExt As String)
    Dim I As Long
    Dim tmpFileExt As String
    tmpFileExt = LCase(Replace(FileExt, ".", ""))
    For I = 1 To UBound(pFileExts)
        If tmpFileExt = pFileExts(I) Then
```


China-pub.com

```
                Exit Sub
            End If
        Next I
        ReDim Preserve pFileExts(UBound(pFileExts) + 1)
        pFileExts(UBound(pFileExts)) = tmpFileExt
    End Sub
```

我们在类模块的通用声明区声明了动态数组变量 **pFileExts**。类被初始化时用零(0)上界重声明了 **pFileExts**。如果使用 ClearFileExts 方法，则也重声明了 **pFileExts** 的上界为零。这样做就清除了文件扩展名列表，因为数组的第一个元素(索引为 0)总是被赋值为空字符串。

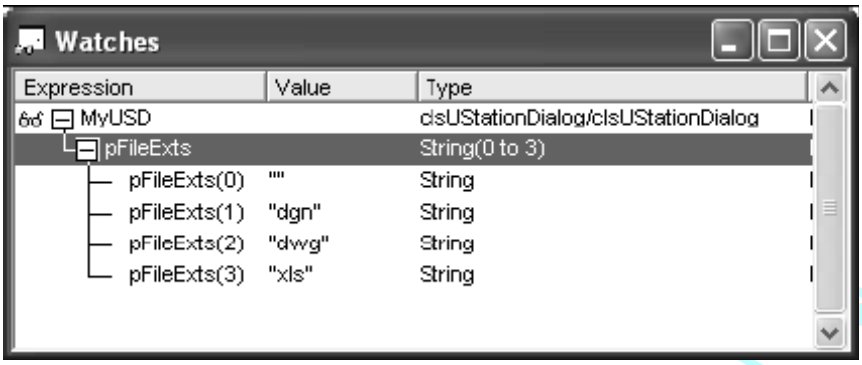
当想添加文件扩展名时，先查看现有的扩展名中是否存在这个扩展名。如果已经存在，则退出过程，对文件扩展名列表不进行任何操作；如果不存在要添加的扩展名，就增大数组并把文件扩展名放入数组的上界元素中。

一种做法是：把所有的代码都编写完然后再测试。但写一小段代码就测试比较好。下面是测试文件扩展名功能的代码，它放在代码模块中。

```
Sub TestFileExts()
    Dim MyUSD As New clsUStationDialog
    MyUSD.AddFileExt ".dgn"
    MyUSD.AddFileExt ".DWg"
    MyUSD.AddFileExt "Xls"
    MyUSD.ClearFileExts
End Sub
```

当单步调试代码时，通过添加对变量 MyUSD 的监视能看到增加文件扩展名后的效果。

chinaBDN



数组中的第一个元素总是空字符串。增加的文件扩展名被移去点号后以小写形式添加到数组中。

当调用 ClearFileExts 方法时，通过重声明变量 **pFileExts** 的上界索引为 0 而移去了除第一个元素外的所有元素。

现在能向类模块中添加文件扩展名了，也能清除扩展名列表。下面要给类模块增加两个属性，让用户能发现有什么扩展名以及有多少扩展名。第一个属性是“ExtCount”，它告诉您类中添加了多少个文件扩展名；另一个属性是“GetExts”，该属性返回一个添加到类中的文件扩展名数组。

过去我们把变量声明为公用的来为类模块建立属性，这样可以，但有更好的处理属性的方式。

ExtCount 属性告诉您类中添加了多少个文件扩展名。如果把变量 **ExtCount** 声明为公用的，则能读和写该变量。这样做不太好，因为这个属性值应当基于添加的实际扩展名的个数。您不想去写该属性，因为它应当被指定为只读的。

```
Property Get    ExtCount()    As Long
        ExtCount = UBound(pFileExts)
End Property
```

现在使用了真正的属性代码，因为属性“ExtCount”完全基于添加到类中的文件扩展名个数。

用 Let 和 Get 语句来指定属性的读/写能力。如果有一个“Property Get”语句而没有关联的“Property Let”语句，则属性为只读；如果有一个“Property Let”语句而没有关联的“Property Get”语句，则属性为只写。只写属性并不普遍，但可用于存储像口令一样的机密信息，这种情况下，您能写属性，类能使用该属

性但该属性不能被用户读出。当提供有“Property Get”也有“Property Let”时，建立了一个读/写属性。

现在用“GetExts”属性来取得文件扩展名，该属性为只读，所以用“Property Get”语句。

```
Property Get GetExts() As String()
    If UBound(pFileExts) = 0 Then
        Exit Property
    End If
    Dim tmpGetExts() As String
    ReDim tmpGetExts(UBound(pFileExts) - 1) As String
    Dim I As Long
    For I = 1 To UBound(pFileExts)
        tmpGetExts(I - 1) = pFileExts(I)
    Next I
    GetExts = tmpGetExts
End Property
```

首先检查是否添加有文件扩展名，如果数组 **pFileExts** 的上界元素为零(0)，则什么也没有添加，此时立即退出属性。否则，建立一个临时动态数组用来存储已添加的文件扩展名。由于数组 **pFileExts** 的第一个元素为空，所以对 **pFileExts** 循环时从数组的第二个元素(索引为 1)开始到其上界元素为止。在对临时动态数组赋完值后，把该值设置给“GetExts”属性从而返回给取属性的代码。

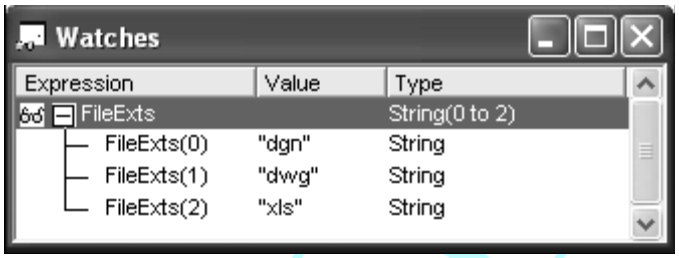
下面是取 GetExts 属性的代码：

```
Sub TestGetExts()
    Dim MyUSD As New clsUStationDialog
    Dim FileExts() As String
    MyUSD.AddFileExt ".dgn"
    MyUSD.AddFileExt "DGN"
    MyUSD.AddFileExt ".DWg"
    MyUSD.AddFileExt "Xls"
    FileExts = MyUSD.GetExts
```

End Sub

注意我们是如何试着添加了两次.dgn 文件扩展名的。如果 AddFileExt 方法工作正常，将只能看到一个 dgn 扩展名。

右图是添加了变量“FileExts”的监视窗口。



三个不重复的文件扩展名被添加，用 GetExts 属性也正确地取出了它们。

现在到了让用户设置和取得文件打开对话框缺省目录的时候了。用“Property Let”和“Property Get”使这一属性为可读/写。

在类的通用声明区声明变量 **pDefFilePath**。

```
Private pDefFilePath As String
```

这个变量将用来保存缺省路径。

从 Property Get 语句开始。

```
Property Get DefaultPath() As String
    DefaultPath = pDefFilePath
End Property
```

这足够简单，就是把变量 **pDefFilePath** 保存的值放入属性“DefaultPath”中。现在让我们看一下 DefaultPath 属性的“Property Let”语句。

```
Property Let DefaultPath(strPathIn As String)
    pDefFilePath = strPathIn
End Property
```

这里是 Let 语句，取得传递过来的值后把它放入私有变量 **pDefFilePath** 中。

Let 和 Get 语句工作得很好，让我们试一下。下面的代码应该放入代码模块中。

```
Sub TestFilePathA()
    Dim MyUSD As New clsUStationDialog
    MyUSD.DefaultPath = "abc:\/?test"
End Sub
```

假如运行了这段代码，“abc:\/?test”就被设置为类中的缺省路径。代码与设计时工作的一模一样，它取得所提供的值并将该值插入类中。所以代码工作完后一切就绪了。是这样的吗？不是！

“abc:\/?test”是一个合法的路径吗？至少在本书出版的时候，它还不是一个合法的路径。那么，我们还要做些什么呢？

当一个属性是可读/写时，通过在类中声明一个公用变量勉强还可以。它允许我们读取和写入变量，就像属性的动作一样。但对象(类)的属性绝不仅仅是能读取和写入的变量。在属性被真正实现以前，先考虑是否需要对用户提供的数据进行校验。本例中，我们需要确认该路径是否存在。有多种方法能做到这一点，下面是其中一种。

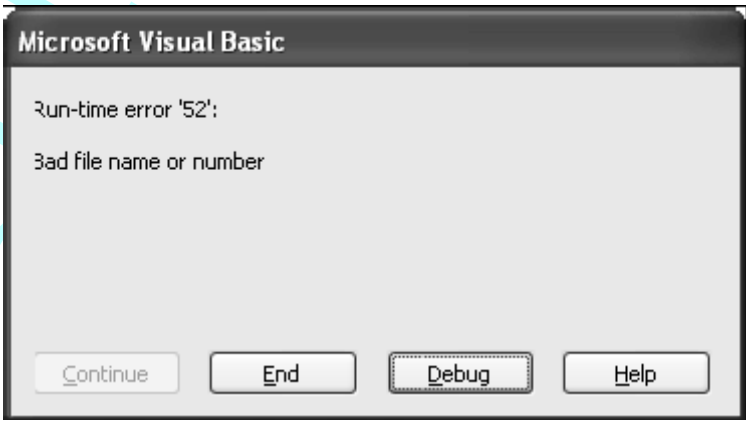
修改您的“Property Let DefaultPath”语句。

```
Property Let DefaultPath(strPathIn As String)
    If Dir(strPathIn, vbDirectory) <> "" Then
        pDefFilePath = strPathIn
    End If
End Property
```

现在不是盲目地把用户给的值赋给路径，而是检查它是否为有效目录。如果是有效目录就把它赋给变量 **pDefFilePath**。如果不是一个合法路径就什么也不做。

让我们运行 TestFilePathA 看一看发生了什么。

我们得到了一个错误，不是因为路径不存在，而是因为所提供的路径不是一个合法路径。让我们建立一个新的测试过程来验证。




```
Sub TestFilePathB()
    Dim MyUSD As New clsUStationDialog
    MyUSD.DefaultPath = "c:\test54321"
```

```
End Sub
```

即使路径“c:\test54321”不存在，但它也是一个能被建立的路径，因为它符合驱动器字母和文件夹名的规则。所以，即使我们期望能给类以合法的路径，但也应该处理错误 52 以防万一。

```
Property Let DefaultPath(strPathIn As String)
    On Error GoTo errhnd
    If Dir(strPathIn, vbDirectory) <> "" Then
        pDefFilePath = strPathIn
    End If
    Exit Property
errhnd:
    Select Case Err.Number
        Case 52 'Bad file name or number
            Err.Clear
    End Select
End Property
```

现在，即使给以非法路径，程序也不会崩溃。下面实现“DefaultFile”属性。

```
Property Get DefaultFile() As String
    DefaultFile = pDefFileName
End Property
Property Let DefaultFile(strFileIn As String)
    pDefFileName = strFileIn
End Property
```

让我们回顾一下到目前为止都实现了什么功能：我们仔细处理了文件扩展名，能设置缺省路径，也能设置缺省文件。这些是开始显示文件打开对话框所需要的所有准备。

我们需要在类模块的通用声明区中声明函数“mdlDialog_fileOpen”如下：

```
Private Declare Function mdlDialog_fileOpen _
    Lib "stdmdlbltin.dll" (ByVal fileName As String, _
    ByVal rFileH As Long, ByVal _
```

```
    resourceId As Long, ByVal suggestedFileName As String, _
    ByVal filterString As String, ByVal defaultDirectory As String, _
    ByVal titleString As String) As Long
```

既然声明了这个函数就能使用它了。

```
Sub OpenDialog()
    Dim tmpFilter As String
    pRetVal = 1
    tmpFilter = "*." & Join(GetExts, "; *.")
    pFileNameSelected = Space(255)
    pRetVal = mdlDialog_fileOpen(pFileNameSelected, 0, 0, _
                pDefFileName, tmpFilter, pDefFilePath, "Open File")
    Select Case pRetVal
        Case 1 'Cancel
        Case 0 'Open
            Dim tmpFile As String
            Dim xSplit As Variant
            tmpFile = Left(pFileNameSelected, InStr(1, _
                                pFileNameSelected, Chr(0)) - 1)
            xSplit = Split(tmpFile, "\")
            pFileName = xSplit(UBound(xSplit))
            xSplit(UBound(xSplit)) = ""
            pFilePath = Join(xSplit, "\")
    End Select
End Sub
```

该类还需要几个属性：

```
Property Get SelectedPath() As String
    SelectedPath = pFilePath
End Property
Property Get SelectedFile() As String
    SelectedFile = pFileName
```

```
End Property
Property Get OpenSuccess() As Boolean
    Select Case pRetVal
        Case 1 'Cancel
            OpenSuccess = False
        Case 0 'Open
            OpenSuccess = True
    End Select
End Property
```

我们至今讨论了该类的大量代码，让我们完整地看一下代码以确保没有遗漏任何内容。

```
'General Declarations
Private Declare Function mdlDialog_fileOpen Lib "stdmdlbltin.dll" (ByVal _
        fileName As String, ByVal rFileH As Long, ByVal _
        resourceId As Long, ByVal suggestedFileName As String, _
        ByVal filterString As String, ByVal defaultDirectory As String, _
        ByVal titleString As String) As Long
Private pFilePath As String
Private pFileName As String
Private pDefFilePath As String
Private pDefFileName As String
Private pFileNameSelected As String
Private pRetVal As Long
Private pFileExts() As String
Property Get SelectedPath() As String
    SelectedPath = pFilePath
End Property
Property Get SelectedFile() As String
    SelectedFile = pFileName
End Property
```

```
Property Get OpenSuccess() As Boolean
    Select Case pRetVal
        Case 1 'Cancel
            OpenSuccess = False
        Case 0 'Open
            OpenSuccess = True
    End Select
End Property
Sub OpenDialog()
    Dim tmpFilter As String
    pRetVal = 1
    tmpFilter = "*." & Join(GetExts, "; *.")
    pFileNameSelected = Space(255)
    pRetVal = mdlDialog_fileOpen(pFileNameSelected, 0, 0, _
                pDefFileName, tmpFilter, pDefFilePath, "Open File")
    Select Case pRetVal
        Case 1 'Cancel
        Case 0 'Open
            Dim tmpFile As String
            Dim xSplit As Variant
            tmpFile = Left(pFileNameSelected, InStr(1, _
                                pFileNameSelected, Chr(0)) - 1)
            xSplit = Split(tmpFile, "\")
            pFileName = xSplit(UBound(xSplit))
            xSplit(UBound(xSplit)) = ""
            pFilePath = Join(xSplit, "\")
    End Select
End Sub
Property Get DefaultFile() As String
    DefaultFile = pDefFileName
```

China BDN

```
End Property
Property Let DefaultFile(strFileIn As String)
    pDefFileName = strFileIn
End Property
Property Get DefaultPath() As String
    DefaultPath = pDefFilePath
    End Property
    Property Let DefaultPath(strPathIn As String)
    On Error GoTo errhnd
    If Dir(strPathIn, vbDirectory) <> "" Then
        pDefFilePath = strPathIn
    End If
    Exit Property
errhnd:
    Select Case Err.Number
        Case 52 'Bad file name or number
            Err.Clear
    End Select
End Property
Property Get ExtCount() As Long
    ExtCount = UBound(pFileExts)
End Property
Property Get GetExts() As String()
    If UBound(pFileExts) = 0 Then
        Exit Property
    End If
    Dim tmpGetExts() As String
    ReDim tmpGetExts(UBound(pFileExts) - 1) As String
    Dim I As Long
    For I = 1 To UBound(pFileExts)
```

China BDN

```
            tmpGetExts(I - 1) = pFileExts(I)
        Next I
        GetExts = tmpGetExts
End Property
Private Sub Class_Initialize()
        ReDim pFileExts(0)
End Sub
Public Sub ClearFileExts()
        ReDim pFileExts(0)
End Sub
Public Sub AddFileExt(FileExt As String)
        Dim I As Long
        Dim tmpFileExt As String
        tmpFileExt = LCase(Replace(FileExt, ".", ""))
        For I = 1 To UBound(pFileExts)
            If tmpFileExt = pFileExts(I) Then
                Exit Sub
            End If
        Next I
        ReDim Preserve pFileExts(UBound(pFileExts) + 1)
        pFileExts(UBound(pFileExts)) = tmpFileExt
End Sub
```

TestShowDialogA 是使用该类模块的代码。

```
Sub TestShowDialogA()
        Dim MyUSD As New clsUStationDialog
        MyUSD.AddFileExt "dgn"
        MyUSD.DefaultPath = "c:\"
        MyUSD.DefaultFile = "test.dgn"
        MyUSD.OpenDialog
        Select Case MyUSD.OpenSuccess
```


China BDN

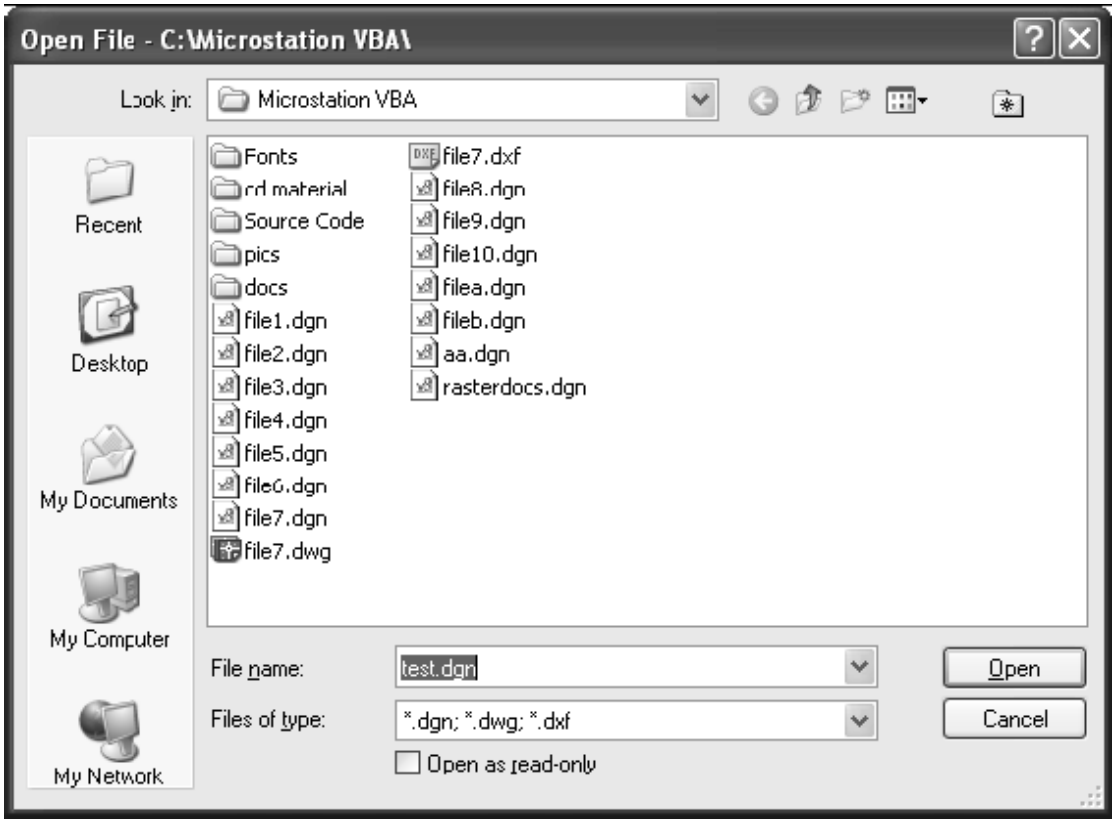
```
        Case True
            MsgBox MyUSD.SelectedPath & MyUSD.SelectedFile
    End Select
End Sub
```

对 TestShowDialogA 作一修改，以确认类的各个方面都运行正常。

```
Sub TestShowDialogB()
    Dim MyUSD As New clsUStationDialog
    MyUSD.AddFileExt "dgn"
    MyUSD.AddFileExt "dwg"
    MyUSD.AddFileExt "dxf"
    MyUSD.DefaultPath = "c:\MicroStation VBA"
    MyUSD.DefaultFile = "test.dgn"
    MyUSD.OpenDialog
    Select Case MyUSD.OpenSuccess
        Case True
            MsgBox MyUSD.SelectedPath & MyUSD.SelectedFile
    End Select
End Sub
```

每个文件扩展名都加入到了文件打开对话框显示的类中。

China BDN



本节主要集中于将特定类型的功能组合到单个类模块中。我们已经给文件打开添加功能，现在给文件建立添加功能。

先在类模块通用声明区中声明函数 mdlDialog_fileCreate。

```
Private Declare Function mdlDialog_fileCreate Lib _
        "stdmdlbltin.dll" (ByVal _
        fileName As String, ByVal rFileH As Long, _
        ByVal resourceId As Long, _
        ByVal suggestedFileName As String, _
        ByVal filterString As String, _
        ByVal defaultDirectory As String, _
        ByVal titleString As String) As Long
```

下一步是用已经声明的变量和属性在类模块中建立一个新方法。

```
Sub CreateDialog()
    Dim tmpFilter As String
    pRetVal = 1
    tmpFilter = "*." & Join(GetExts, "; *.")
```

```
    pFileNameSelected = Space(255)
    pRetVal = mdlDialog_fileCreate(pFileNameSelected, 0, 0, _
                pDefFileName, tmpFilter, pDefFilePath, "Create File")
    Select Case pRetVal
        Case 1 'Cancel
        Case 0 'Open
            Dim tmpFile As String
            Dim xSplit As Variant
            tmpFile = Left(pFileNameSelected, InStr(1, _
                            pFileNameSelected, Chr(0)) - 1)
            xSplit = Split(tmpFile, "\")
            pFileName = xSplit(UBound(xSplit))
            xSplit(UBound(xSplit)) = ""
            pFilePath = Join(xSplit, "\")
    End Select
End Sub
```

现在是测试类中 CreateDialog 方法的时候。

```
Sub TestShowDialogC()
    Dim MyUSD As New clsUStationDialog
    MyUSD.AddFileExt "dgn"
    MyUSD.DefaultPath = "c:\"
    MyUSD.DefaultFile = "test.dgn"
    MyUSD.CreateDialog
    Select Case MyUSD.OpenSuccess
        Case True
            MsgBox MyUSD.SelectedPath & MyUSD.SelectedFile
    End Select
End Sub
```

TestShowDialogC 几乎是 TestShowDialogA 的一个翻版，唯一的不同是用

CreateDialog 代替了 OpenDialog，其它都一样。

复制并粘贴 TestShowDialogB 以建立 TestShowDialogD 并作同样的修改。

```
Sub TestShowDialogD()
    Dim MyUSD As New clsUStationDialog
    MyUSD.AddFileExt "dgn"
    MyUSD.AddFileExt "dwg"
    MyUSD.AddFileExt "dxf"
    MyUSD.DefaultPath = "c:\MicroStation VBA"
    MyUSD.DefaultFile = "test.dgn"
    MyUSD.CreateDialog
    Select Case MyUSD.OpenSuccess
        Case True
            MsgBox MyUSD.SelectedPath & MyUSD.SelectedFile
    End Select
End Sub
```

再说一次，唯一的改变是用 CreateDialog 取代了 OpenDialog。

让我们在一个新的过程 TestShowDialogE 中来扩展 TestShowDialogA。

```
Sub TestShowDialogE()
    Dim MyUSD As New clsUStationDialog
    MyUSD.AddFileExt "dgn"
    MyUSD.DefaultPath = "c:\"
    MyUSD.DefaultFile = "test.dgn"
    MyUSD.OpenDialog
    Select Case MyUSD.OpenSuccess
        Case True
            MsgBox "Open " & MyUSD.SelectedPath & _
                   MyUSD.SelectedFile
        Case False
            If MsgBox("Create a new file?", vbYesNo) = vbYes Then
                MyUSD.CreateDialog
```


China BDN

```
                If MyUSD.OpenSuccess = True Then
                    MsgBox "Create " & MyUSD.SelectedPath & _
                        MyUSD.SelectedFile
                End If
            End If
        End Select
    End Sub
```

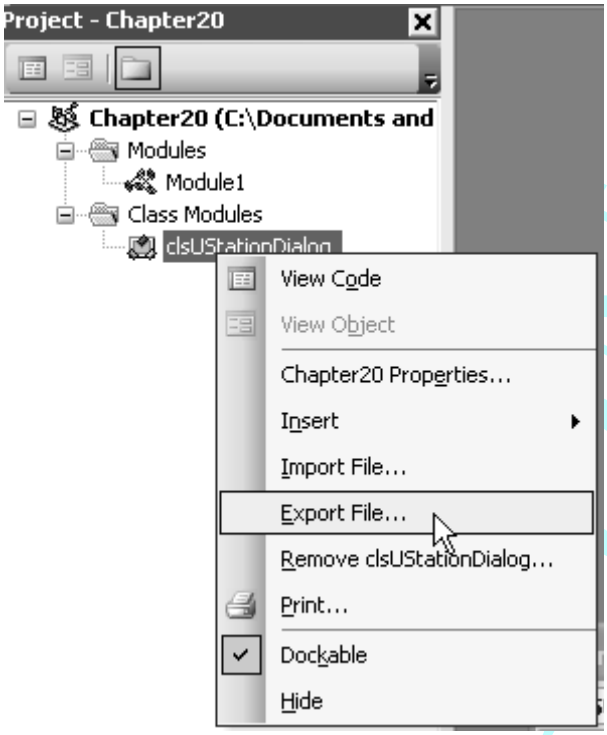
TestShowDialogE 做了什么？它提示用户选择一个要打开的文件。如果用户选择了一个现有的文件，就在消息框中显示这个文件名。如果用户点击了取消(Cancel)按钮，MyUSD.OpenSuccess 属性为 False，此时我们问用户是否想建立一个新文件。如果用户回答 No，就什么也不做；如果用户回答 Yes，就显示文件建立对话框并允许用户建立新文件。如果用户用文件建立对话框成功地建立了一个文件，就在消息框中显示文件路径和文件名。

请注意，给类模块添加 FileCreate(文件建立)功能以及在过程中使用 CreateDialog 方法是多么的容易。一旦您草拟出了要包含在类模块中的功能大纲，就能快速而轻易地扩展它们。

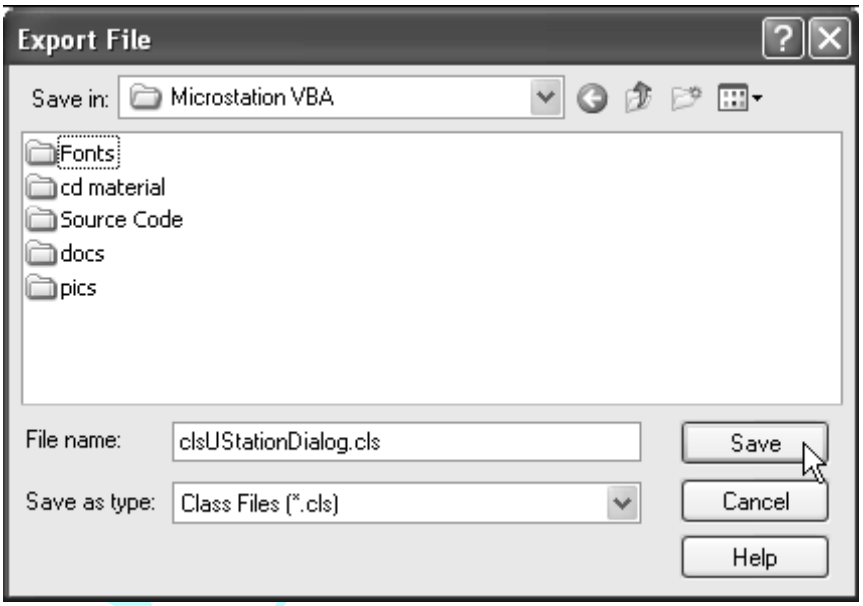
从现在开始，每当您需要使用文件打开和文件建立对话框时，都可以使用刚刚建立的类。但如何在将来的工程中使用它呢？需要每次使用时都复制和粘贴代码吗？这里有个更好的方法。

从 VBA 工程窗口(不要与 VBA 工程管理器混淆)中右击类并选择 Export File(导出文件)。

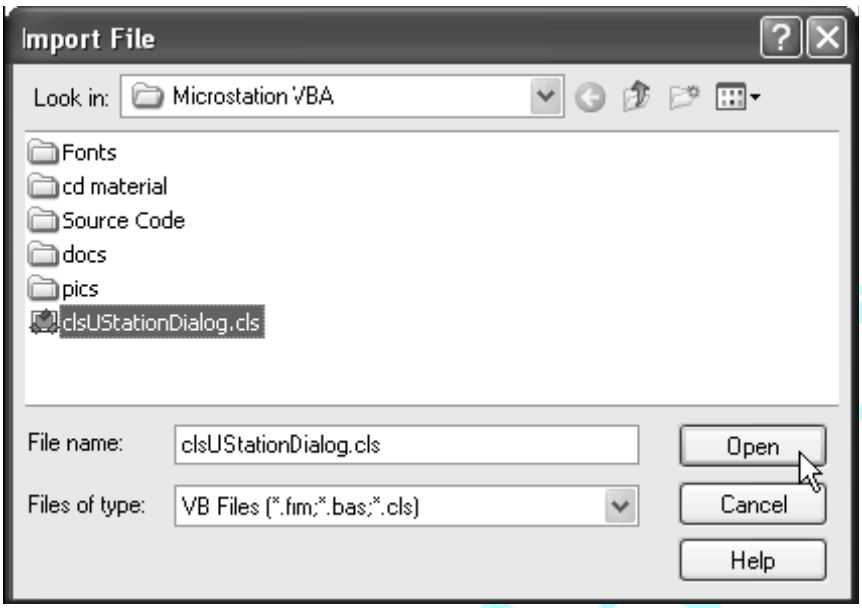
ChinaBDN



选择一个位置并保存文件。



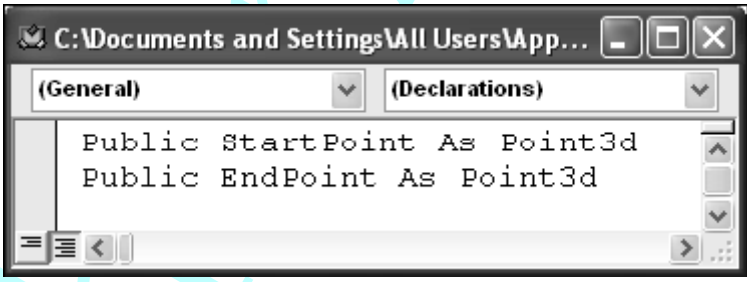
现在，建立完一个新的 VBA 工程后就用 VBA 菜单 **File > Import (文件 > 导入)**。选择刚导出的类，它就会被导入到新的工程中来。



关于建立类模块以把相似功能包装到单个对象或类中的讨论到此为止。让我们看另一种使用类的方式。

建立带有属性、方法和事件的对象

MicroStation 为我们提供了大量可用的类(或对象)。例如，LineElement 对象具有 StartPoint 和 EndPoint 属性但没有 MidPoint 属性，也没有 ChangeInX、ChangeInY 或 ChangeInZ 属性。让我们建立一个新的 clsLineElem 类并包含一些不属于 MicroStation 对象模块的属性。



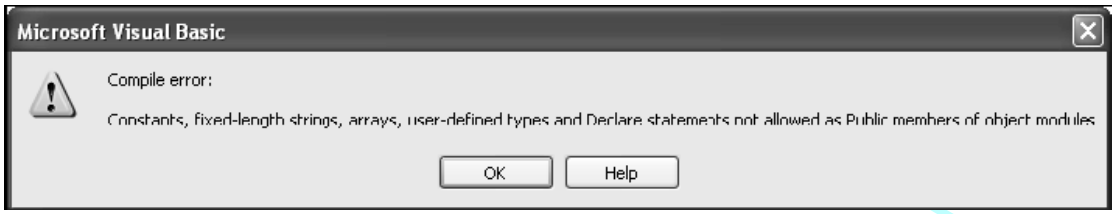
为简单起见，在类模块的通用声明区声明两个公用变量来实现 StartPoint 和 EndPoint 属性。

为确保事情做得对，我们建立一个测试过程来使用该类。

下一步才是建立 MidPoint 属性。

```
Sub TestNewLineA()
    Dim myLE As New clsLineElem
End Sub
```

当运行这个宏时，将看到如下消息框：



发生了什么问题？我们在通用声明区声明了两个 Point3d 类型的公用变量，Point3d 变量类型不是标准 VBA 变量类型，所以不能这样声明。把声明改为私有的并改变量名如下：

```
Private pStartPoint As Point3d
Private pEndPoint As Point3d
```

现在再运行 TestNewLineA，就不会出现错误了。

那么如何实现 StartPoint 和 EndPoint 属性呢？在通用声明区中声明完 pStartPoint 和 pEndPoint 变量后，如下所示实现 StartPoint 和 EndPoint 属性：

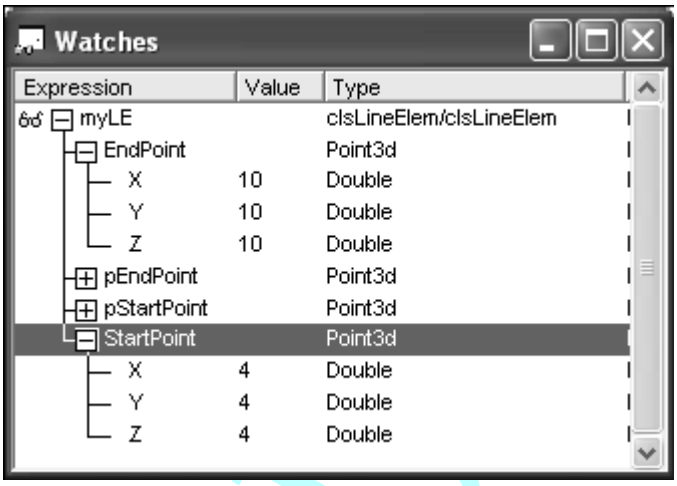
```
Property Let StartPoint(StPt As Point3d)
    pStartPoint = StPt
End Property
Property Get StartPoint() As Point3d
    StartPoint = pStartPoint
End Property
Property Let EndPoint(EnPt As Point3d)
    pEndPoint = EnPt
End Property
Property Get EndPoint() As Point3d
  EndPoint = pEndPoint
End Property
```

让我们在代码模块中建立一个过程来测试属性。

```
Sub TestNewLineB()
    Dim myLE As New clsLineElem
    myLE.StartPoint = Point3dFromXYZ(4, 4, 4)
    myLE.EndPoint = Point3dFromXYZ(10, 10, 10)
```

End Sub

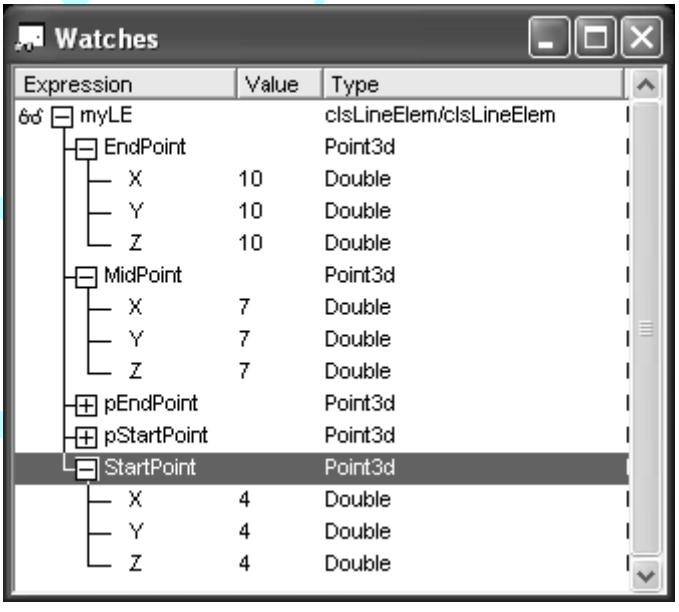
这段代码运行无误。为了确认正确性，在 End Sub 行放置一个停止(或叫断点)，这样可以对变量 myLE 添加监控。右图是得到的结果：



既然 StartPoint 和 EndPoint 属性好像工作正常了，让我们继续增加 MidPoint 属性。MidPoint 属性从 StartPoint 和 EndPoint 计算得来，由于该属性是计算得来，所以只用“Property Get”而没有“Property Let”来使该属性只读。

```
Property Get MidPoint() As Point3d
    Dim tmpPoint As Point3d
    tmpPoint.X = StartPoint.X + (EndPoint.X - StartPoint.X) / 2
    tmpPoint.Y = StartPoint.Y + (EndPoint.Y - StartPoint.Y) / 2
    tmpPoint.Z = StartPoint.Z + (EndPoint.Z - StartPoint.Z) / 2
    MidPoint = tmpPoint
End Property
```

单步运行 TestNewLineB 过程显示 MidPoint 属性的结果。



计算结果像是正确的。让我们再添加几个属性：ChangeInX、ChangeInY 和 ChangeInZ。

```
Property Get ChangeInX() As Double
    ChangeInX = pEndPoint.X - pStartPoint.X
End Property
Property Get ChangeInY() As Double
    ChangeInY = pEndPoint.Y - pStartPoint.Y
End Property
Property Get ChangeInZ() As Double
    ChangeInZ = pEndPoint.Z - pStartPoint.Z
End Property
```

又添加两个属性：LineAngleRads 和 LineAngleDegs 属性：

```
Property Get LineAngleRads() As Double
    LineAngleRads = Atn((pEndPoint.Y - pStartPoint.Y) / _
                        (pEndPoint.X - pStartPoint.X))
End Property
Property Get LineAngleDegs() As Double
    LineAngleDegs = Degrees(LineAngleRads)
End Property
```

属性足够多了，来看一下如何建立方法。第一个方法是 DrawLine。

```
Sub DrawLine()
    Dim LineElem As LineElement
    Set LineElem = CreateLineElement2(Nothing, _
                                      pStartPoint, pEndPoint)
    ActiveModelReference.AddElement LineElem
End Sub
```

DrawLine 方法基于 pStartPoint 和 pEndPoint 变量建立了一个线元素，这条线随后被加入到 MicroStation 的当前模型中。现在在代码模块中添加过程 TestNewLineD 来做一个例子：

```
Sub TestNewLineD()
```


China BDN

```
    Dim myLE As New clsLineElem
    myLE.StartPoint = Point3dFromXYZ(0, 0, 0)
    myLE.EndPoint = Point3dFromXYZ(10, 10, 0)
    myLE.DrawLine
    myLE.EndPoint = Point3dFromXYZ(-10, 10, 0)
    myLE.DrawLine
End Sub
```

设置起点和终点值，然后使用 DrawLine 方法，这很简单。接下来试着为类模块建立另一个方法。

这个方法叫做“DrawLinePerp”。它绘制一条与类中的 pStartPoint 和 pEndPoint 属性所定义的直线垂直且通过该直线中点的线。

```
Sub DrawLinePerp()
    Dim PerpSt As Point3d
    Dim PerpEn As Point3d
    Dim PerpMid As Point3d
    Dim LineAng As Double
    Dim LineLength As Double
    LineAng = LineAngleRads
    PerpMid = MidPoint
    LineLength = Point3dDistance(pStartPoint, pEndPoint)
    PerpSt = Point3dAddAngleDistance(PerpMid, LineAng + Pi / 2, _
                                     LineLength / 2, 0)
    PerpEn = Point3dAddAngleDistance(PerpMid, LineAng - Pi / 2, _
                                     LineLength / 2, 0)
    Dim LineElem As LineElement
    Set LineElem = CreateLineElement2(Nothing, PerpSt, PerpEn)
    ActiveModelReference.AddElement LineElem
End Sub
```

现在，为了测试 DrawLinePerp 方法，在代码模块中建立一个新过程。

```
Sub TestNewLineE()
```


China BDN

```
    Dim myLE As New clsLineElem
    myLE.StartPoint = Point3dFromXYZ(0, 0, 0)
    myLE.EndPoint = Point3dFromXYZ(8, 8, 0)
    myLE.DrawLine
    myLE.DrawLinePerp
End Sub
```

这段代码绘制了两条线。一条是从(0,0,0)到(8,8,0)，另一条是与第一条垂直且过其中点的线。

给类模块添加另一个方法，先是代码后是解释。

```
Sub DrawCircle()
    Dim CircElem As EllipseElement
    Dim RotMatrix As Matrix3d
    Set CircElem = CreateEllipseElement2(Nothing, MidPoint, _
                Point3dDistance(pStartPoint, pEndPoint) / 2, _
                Point3dDistance(pStartPoint, pEndPoint) / 2, RotMatrix)
    ActiveModelReference.AddElement CircElem
End Sub
```

DrawCircle 在线元素对象的端点绘制一个圆。

类模块的强大功能之一是，当需要方法和属性时就能添加它们。我们的类模块目前有三个方法，它们就是应用所需要的全部方法。随后，如果下一个应用需要一个叫做“DrawLineSegments”的在起点和终点间指定线段个数的方法，我们也能添加它。能向类模块中添加任意多个方法和属性，但是需要小心谨慎。修改现有的方法和属性时尤其要更加小心。

举个例子，我们有一个叫做 StartPoint 的属性被声明为 Point2d 类型，它应用在许多场合，把它改成 Point3d 类型似乎能支持更多的方法和属性。但是，把变量从 Point2d 改成 Point3d 会引起使用该类模块的代码出错，因为在现有代码的函数或过程中期望一个 Point2d 的类型。小心规划类模块中的属性和方法能有助于它们被使用后再被修改。

我们讨论过了属性和方法，但什么是事件呢？当特定的活动发生时一个事件

就被触发。我们通常在事件中编写代码来响应用户的交互动作。

让我们在类中建立一个事件，当在 MicroStation 中建立并绘制一条线时触发该事件。

在类的通用声明区使用如下代码建立一个事件：

```
Public Event LineAdded(AddedLine As LineElement)
```

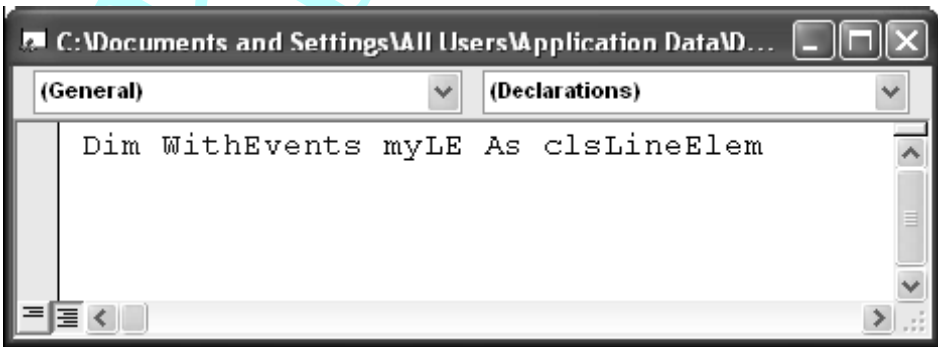
叫做“LineAdded”的事件现在是类的一部分了，但如何触发这个事件呢？

```
Sub DrawLine()
    Dim LineElem As LineElement
    Set LineElem = CreateLineElement2(Nothing, _
                                      pStartPoint, pEndPoint)
    ActiveModelReference.AddElement LineElem
    RaiseEvent LineAdded(LineElem)
End Sub
```

若要触发一个事件时，使用 RaiseEvent 语句。把 LineElem 变量作为参数，这样当使用该类模块的代码捕获这一事件时，可得到在 MicroStation 中绘制的线元素。

既然在类模块中已经声明并发出了一个事件，那下一步是什么呢？我们怎样编写事件中的代码使得有线加入时做某件事情？好问题！要使用刚建立的事件，需要在另一个类模块或一个窗体的“WithEvents”中声明一个 **clsLineElem** 类的变量。用窗体来演示该事件的使用较容易一些。

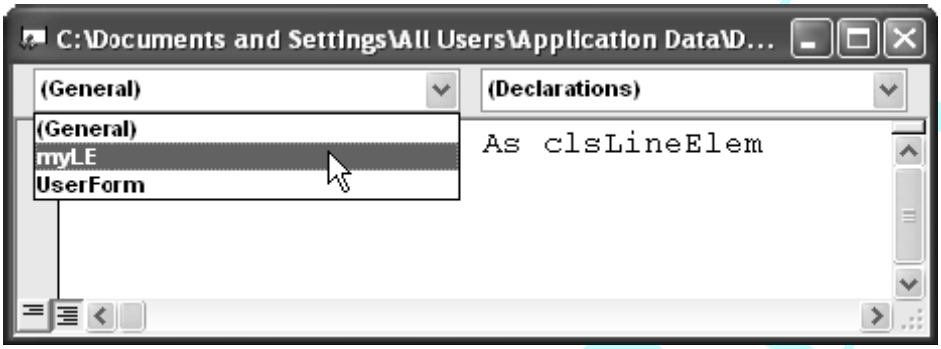
在 VBA 工程中插入一个新窗体。在窗体代码的通用声明区中声明一个“WithEvents”的变量为 **clsLineElem** 对象。



以这种方式声明一个变量，则该变量的事件就像一个命令按钮事件一样有

效。

对象 **myLE**(现在使用变量名而不是类名)现在在窗体代码的对象复合框中也能找到了。



在对象复合框中选择 **myLE** 后，能够在过程复合框中看到 **clsLineElem** 的事件。

现在开始使用在 **clsLineElem** 类中建立的事件。

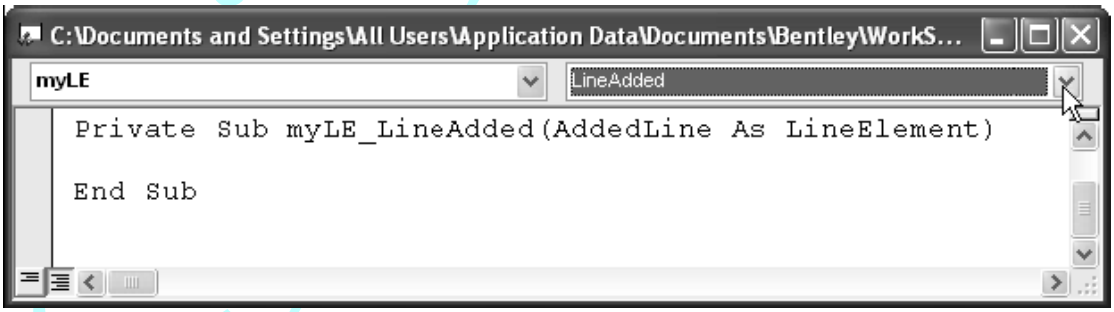
让我们回顾一下使用事件所需要的四个成分。

1、在类模块中用“Public Event”声明事件；

2、在类模块的属性或方法中用“RaiseEvent”发出事件；

3、在另一个类或窗体的通用声明区声明一个“WithEvents”变量作为该类；

4、在事件中输入代码。



在集合中使用类模块

我们用类模块包装功能，并通过使用属性、方法和事件来建立新对象。您也能在自定义的集合中使用类。

集合是保存对象的容器。自定义 VBA 集合能包含任何类型的对象。让我们看一个建立新集合的例子，该集合保存 MicroStation 层(Level)对象。

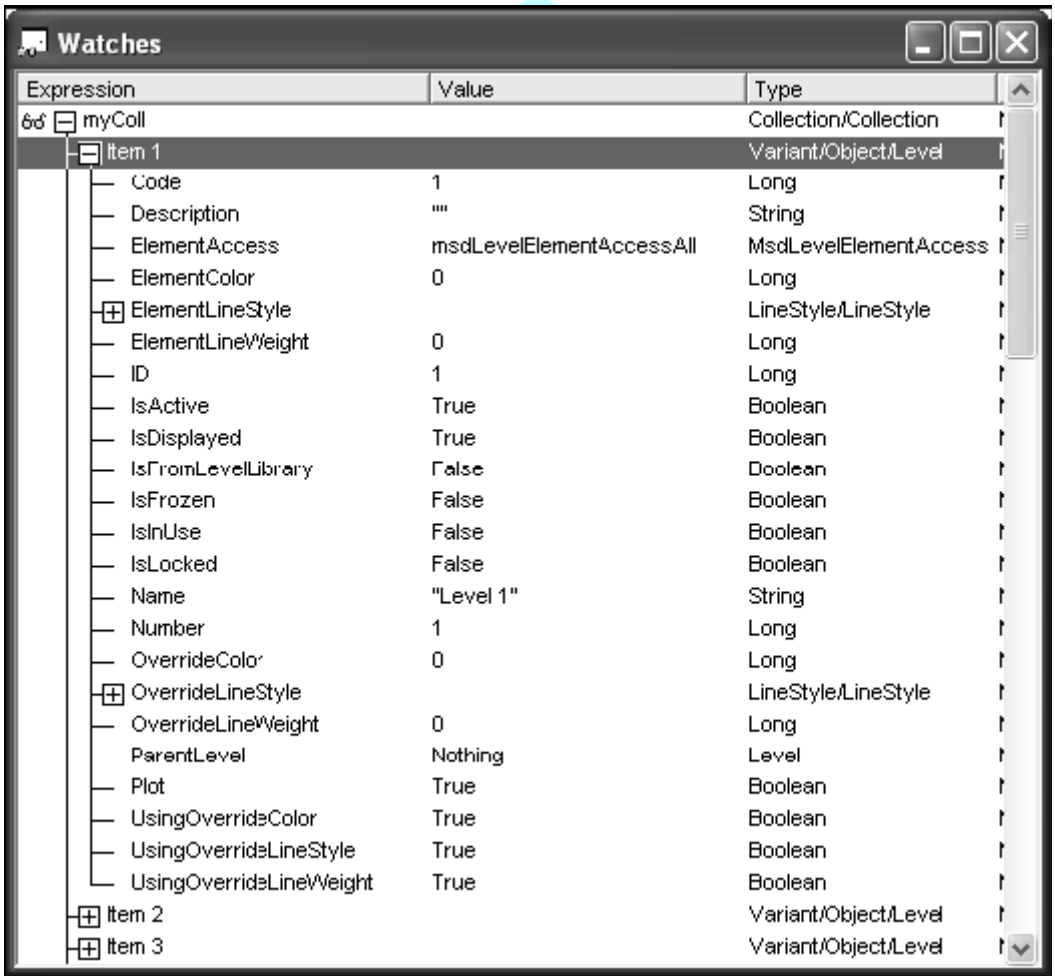
```
Sub TestCollectionA()
    Dim myColl As New Collection
    Dim myLevel As Level
    For Each myLevel In ActiveModelReference.Levels
        myColl.Add myLevel
    Next
End Sub
```

集合的 Add 方法声明是：

Sub Add(Item, [Key], [Before], [After])

在第一个例子中，通过向 Add 方法的 Item 参数提供项目(即 Item，在该情况下是一个层)将层加入了一个集合中。接下来要看一下 Key(键值)、Before(之前)和 After(之后)参数。

对 **myColl** 集合添加监视，然后按**<F8>**键进入 TestCollectionA 过程单步跟踪。



添加的每个对象都以集合中的一项来显示。能在 Type(类型)列看出对象的类型，所有对象的属性都显示有它们的值。

存取集合中的对象

对象加入集合后，有不同的方式来存取这些对象。

```
Sub TestCollectionB()
    Dim myColl As New Collection
    Dim myLevel As Level
    For Each myLevel In ActiveModelReference.Levels
        myColl.Add myLevel
    Next
    'Now that the collection is populated,
    'access the objects with For Each . . Next
    Dim myLevel2 As Level
    For Each myLevel2 In myColl
        Debug.Print myLevel2.Name
    Next
End Sub
```

一种方式是使用 For Each ... Next 语句。上例用当前模型中的层赋值一个集合，随后用 For Each ... Next 存取集合中的每个对象。

对象加入集合后，对象的属性、方法和事件依然存在，这就是说，它们不是保存值的静态变量。存取对象时要注意：不要不经意间修改了对象的属性。

使用 For Each ... Next 能轻易地存取集合中的每个对象。下面是另一种存取集合中对象的方式：

```
Sub TestCollectionC()
    Dim myColl As New Collection
    Dim myLevel As Level
    For Each myLevel In ActiveModelReference.Levels
        myColl.Add myLevel
    Next
```


China BDN

```
        'Now that the collection is populated,
        'access the objects using the item Index.
        Dim myLevel2 As Level
        Dim I As Long
        For I = 1 To myColl.Count
            Set myLevel2 = myColl(I)
            Debug.Print myLevel2.Name
        Next
    End Sub
```

我们仍然循环到了集合中的每个项，但现在是通过集合中项的索引来存取每个项。

```
    Sub TestCollectionD()
        Dim myColl As New Collection
        Dim myLevel As Level
        For Each myLevel In ActiveModelReference.Levels
            myColl.Add myLevel
        Next
        'Now that the collection is populated,
        'access the objects using the item Index.
        Dim myLevel2 As Level
        Dim I As Long
        For I = myColl.Count To 1 Step -1
            Set myLevel2 = myColl(I)
            Debug.Print myLevel2.Name
        Next
    End Sub
```

现在，不是自上而下定位集合中的项，而是使用了带“Step -1”的 For ... Next 语句自底向上定位项的。

让我们再看一眼集合的 Add 方法的声明：

```
    Sub Add(Item, [Key], [Before], [After])
```


China BDN

如果向集合中添加对象时提供一个键值(集合中唯一的字符串)，就能借助这个键值来定位对象。下面的例子向集合 **myColl** 添加层并用层名作为集合中项的键值。

```
Sub TestCollectionF()
    Dim myColl As New Collection
    Dim myLevel As Level
    For Each myLevel In ActiveModelReference.Levels
        myColl.Add myLevel, myLevel.Name
    Next
    'Now that the collection is populated,
    'address a Level by it's Key.
    Dim myLevel2 As Level
    Set myLevel2 = myColl("Default")
    MsgBox myLevel2.Number
End Sub
```

我们知道，每个 MicroStation 的 DGN 文件都有一个叫做“Default”的层。因为可能不知道该层在集合中的索引是多少，所以，通过对象的键值来存取集合中的层。

键值必须是唯一的字符串。换句话说，一个集合中不能有两个对象具有相同的键值。键值不区分大小写，所以在同一个集合中不能有键值“test”和键值“TEST”同时存在。

从集合中移除对象

可以用键值或索引来移除集合中的对象。用索引移除时，比被移除对象索引大的那些对象的索引值减一。因为这个原因，清除整个集合时总是用“Step -1”语句从最高索引开始处理到最低索引(总是索引 1)。

下面是一个用层赋值集合，然后用 Remove 方法移除所有项的例子。

```
Sub TestCollectionE()
    Dim myColl As New Collection
    Dim myLevel As Level
```

```
    For Each myLevel In ActiveModelReference.Levels
        myColl.Add myLevel
    Next
    'Now that the collection is populated,
    'remove the objects using the item Index.
    Dim I As Long
    For I = myColl.Count To 1 Step -1
        myColl.Remove I
    Next
End Sub
```

使用自定义类模块

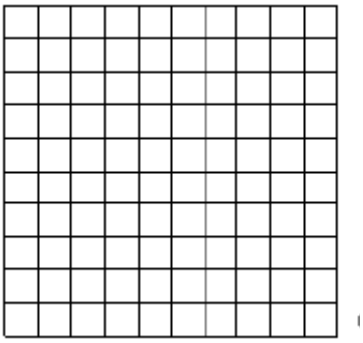
现在来使用 **clsLineElem** 类。

```
Sub TestCollectionG()
    Dim myLE As clsLineElem
    Dim XYA As Double
    Dim XYMin As Double
    Dim XYMax As Double
    Dim myColl As New Collection
    XYMin = 0
    XYMax = 10
    For XYA = XYMin To XYMax
        Set myLE = New clsLineElem
        myLE.StartPoint = Point3dFromXY(XYA, XYMin)
        myLE.EndPoint = Point3dFromXY(XYA, XYMax)
        myColl.Add myLE
        Set myLE = Nothing
        Set myLE = New clsLineElem
        myLE.StartPoint = Point3dFromXY(XYMin, XYA)
        myLE.EndPoint = Point3dFromXY(XYMax, XYA)
```

China BDN

```
        myColl.Add myLE
        Set myLE = Nothing
    Next XYA
    For Each myLE In myColl
        myLE.DrawLine
    Next
End Sub
```

本例建立了多个 clsLineElem 对象并把它们加入到一个集合中。然后用 For Each ... Next 语句存取集合中的每个 **clsLineElem** 对象并使用每个对象的 DrawLine 方法。



这个宏运行后，应该看到如右图所示的一个网格。

注意，如果不使用每个 clsLineElem 类中的 DrawLine 方法，将不会绘制出任何图形。

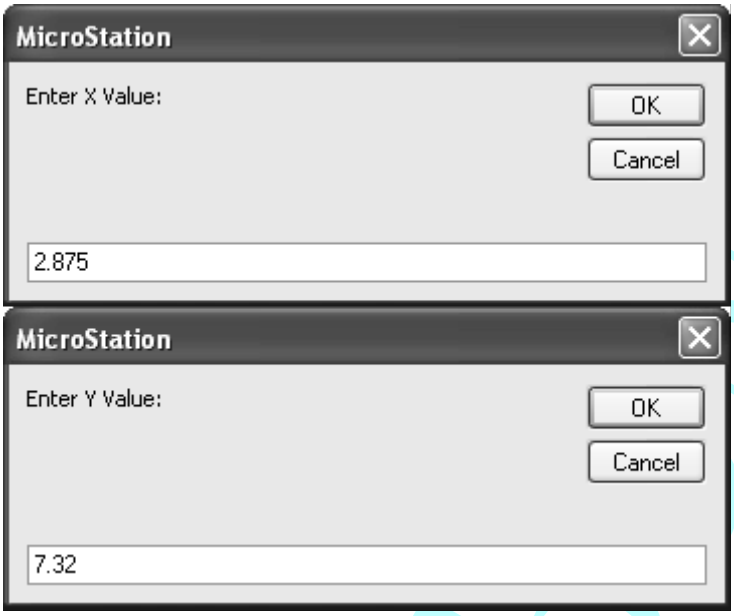
让我们看另一个在集合中使用 clsLineElem 类的例子。

```
Sub TestCollectionH()
    Dim myLE As clsLineElem
    Dim XYA As Double
    Dim XYMin As Double
    Dim XYMax As Double
    Dim xClose As Double
    Dim yClose As Double
    Dim myColl As New Collection
    XYMin = 0
    XYMax = 10
    For XYA = XYMin To XYMax
        Set myLE = New clsLineElem
        myLE.StartPoint = Point3dFromXY(XYA, XYMin)
        myLE.EndPoint = Point3dFromXY(XYA, XYMax)
```

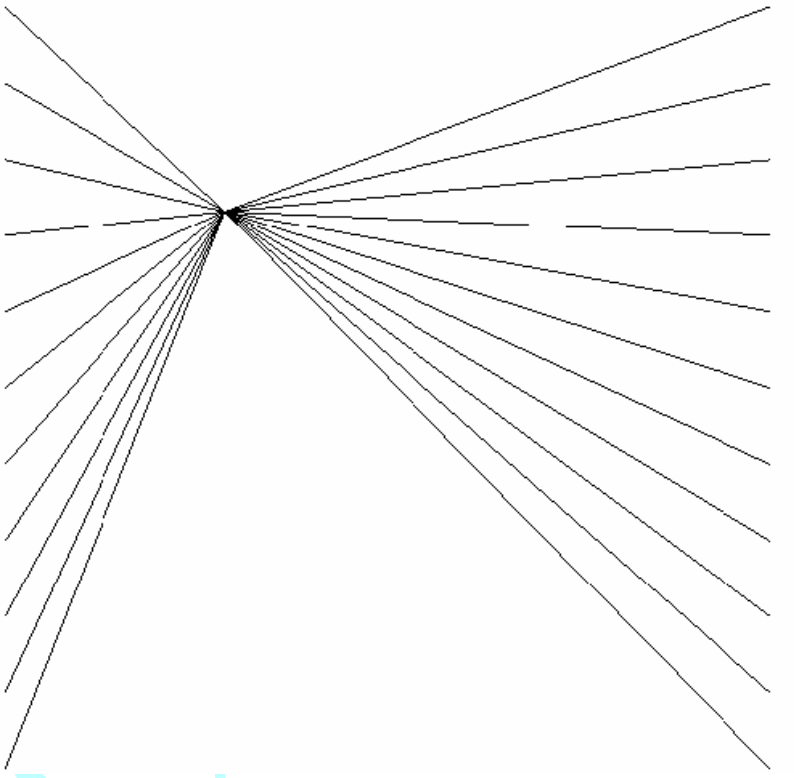
```
            myColl.Add myLE
            Set myLE = Nothing
            Set myLE = New clsLineElem
            myLE.StartPoint = Point3dFromXY(XYMin, XYA)
            myLE.EndPoint = Point3dFromXY(XYMax, XYA)
            myColl.Add myLE
            Set myLE = Nothing
        Next XYA
        xClose = CDbl(InputBox("Enter X Value:"))
        yClose = CDbl(InputBox("Enter Y Value:"))
        For Each myLE In myColl
            If myLE.StartPoint.Y = myLE.EndPoint.Y Then
                Set myLineElem = CreateLineElement2(Nothing, _
                        myLE.StartPoint, Point3dFromXY(xClose, yClose))
                ActiveModelReference.AddElement myLineElem
                Set myLineElem = CreateLineElement2(Nothing, _
                        myLE.EndPoint, Point3dFromXY(xClose, yClose))
                ActiveModelReference.AddElement myLineElem
            End If
        Next
    End Sub
```

在这个过程中，我们建立了一个 clsLineElem 对象的集合，这一点与上个例子生成网格的代码是相同的。然后考察集合中的每个 clsLineElem 并考察起点和终点的 Y 分量，如果它们相等，我们知道正在处理一条水平线。对于每个水平的 myLineElem 对象，我们绘制从输入的 X 和 Y 值分别到 myLineElem 对象的起点和终点间的两条线。

Qing BDN



运行这段代码后，用户被提示输入 X 和 Y 值，然后绘制从输入的 X、Y 坐标点分别到集合中每个 **clsLineElem** 对象的起点和终点的线。



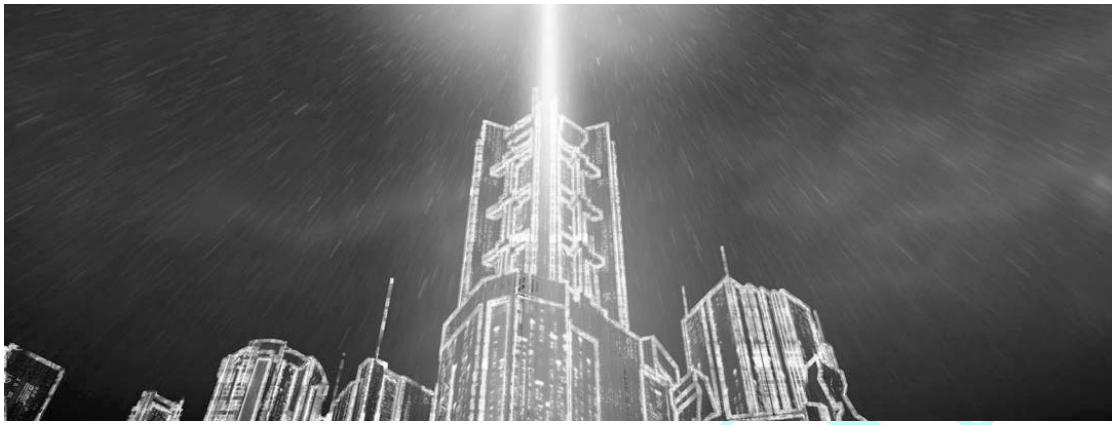
这个例子演示了如何在一个集合中放入多个对象，然后用集合来确定集合中对象的值。

本章回顾

类能封装相似的功能；建立具有独特属性、方法和事件的对象；把具有不同

目的的相似对象加入到集合中。在您的程序中实现的类越多，您就越追随于面向对象的编程思想。

China BDN



21. CAD 管理员使用的 VBA

VBA 不仅仅是为程序员所使用，也不仅仅是为 MicroStation 用户所使用 — 对 CAD 管理员来说，它也是一个强有力的工具。

本章包括：

- 用 VBA 来维护标准
- 用 VBA 简化跨公司标准
- 用 VBA 跟踪图形中的时间
- 自动装载和自动运行宏
- 保护工程
- 分发 VBA 工程
- 在高安全模式下工作

用 VBA 来维护标准

由于公司与公司间标准的不同，VBA 能够作为每个公司定制和维护标准的重要部分。让我们探讨一些用 VBA 查看 DGN 文件并报告问题的方法。

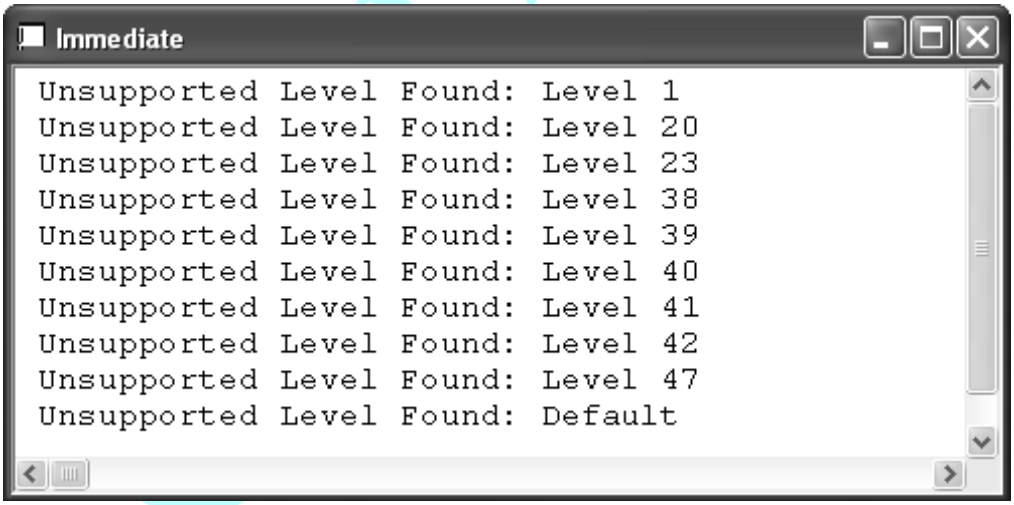
> **注意：**关于维护标准的更多信息，请看标准检查器界面(Standards Cheker Interface)，它提供了带有内置报告能力的强大功能。

要建立的第一个过程的功能是查找不支持的层。

```
Sub FindUnsupportedLevels()
    Dim GoodLevels(0 To 4) As String
```

```
    Dim GoodLevelsJ As String
    Dim myLevel As Level
    GoodLevels(0) = "ROADWAY"
    GoodLevels(1) = "SIDEWALK"
    GoodLevels(2) = "PAINT"
    GoodLevels(3) = "ELECTRIC"
    GoodLevels(4) = "GAS"
    GoodLevelsJ = UCase("~" & Join(GoodLevels, "~") & "~")
    For Each myLevel In ActiveDesignFile.Levels
        If InStr(1, GoodLevelsJ, "~" & UCase(myLevel.Name) & "~") = 0
    Then
        Debug.Print "Unsupported Level Found: " & myLevel.Name
        End If
    Next
End Sub
```

指定了五个支持的层。用波浪字符(~)作分隔合并层名数组，然后用 InStr 函数在合并的字符串中查找。如果未在支持的层名字符串中找到某个层，就把该层名打印输出到立即窗口中。



这段代码查找不支持的层做得很好。一个文件假设有某些层但这些层可能不存在，此时怎么办呢？下个例子就是用于查找丢失的层。

```
Sub FindMissingLevels()
    Dim GoodLevels(0 To 4) As String
```

```
    Dim LevelFound(0 To 4) As Boolean
    Dim myLevel As Level
    Dim I As Long
    GoodLevels(0) = "ROADWAY"
    GoodLevels(1) = "SIDEWALK"
    GoodLevels(2) = "PAINT"
    GoodLevels(3) = "ELECTRIC"
    GoodLevels(4) = "GAS"
    GoodLevelsJ = UCase("~" & Join(GoodLevels, "~") & "~")
    For Each myLevel In ActiveDesignFile.Levels
        For I = LBound(GoodLevels) To UBound(GoodLevels)
            If StrComp(GoodLevels(I), myLevel.Name, vbTextCompare) = 0 Then
                LevelFound(I) = True
            End If
        Next I
    Next
    For I = LBound(GoodLevels) To UBound(GoodLevels)
        If LevelFound(I) = False Then
            Debug.Print "MISSING LEVEL: " & GoodLevels(I)
        End If
    Next I
End Sub
```

上面的代码将能把丢失的层报告在立即窗口中。

总不能让 CAD 管理员通过对 VBA 中的立即窗口进行屏幕拷贝生成一个报告吧。让我们复制并粘贴以上两个过程并增加一些代码把报告写入 ASCII 文本文件中。

```
Sub FindUnsupportedLevelsB()
    Dim GoodLevels(0 To 4) As String
    Dim GoodLevelsJ As String
    Dim myLevel As Level
```

```
    Dim FFile As Long
    GoodLevels(0) = "ROADWAY"
    GoodLevels(1) = "SIDEWALK"
    GoodLevels(2) = "PAINT"
    GoodLevels(3) = "ELECTRIC"
    GoodLevels(4) = "GAS"
    GoodLevelsJ = UCase("~" & Join(GoodLevels, "~") & "~")
    FFile = FreeFile
    Open "C:\MicroStation VBA\LevelsUnSupported.txt" For Output As #FFile
    For Each myLevel In ActiveDesignFile.Levels
        If InStr(1, GoodLevelsJ, "~" & UCase(myLevel.Name) & "~") = 0 Then
            Print #FFile, ActiveDesignFile.Path & ActiveDesignFile.Name _
                & vbTab & "Unsupported Level Found: " & myLevel.Name
        End If
    Next
    Close #FFile
End Sub

Sub FindMissingLevelsB()
    Dim GoodLevels(0 To 4) As String
    Dim LevelFound(0 To 4) As Boolean
    Dim myLevel As Level
    Dim I As Long
    Dim FFile As Long
    GoodLevels(0) = "ROADWAY"
    GoodLevels(1) = "SIDEWALK"
    GoodLevels(2) = "PAINT"
    GoodLevels(3) = "ELECTRIC"
    GoodLevels(4) = "GAS"
    GoodLevelsJ = UCase("~" & Join(GoodLevels, "~") & "~")
```


China BDN

```
    FFile = FreeFile
    Open "C:\MicroStation VBA\LevelsMissing.txt" For Output As #FFile
    For Each myLevel In ActiveDesignFile.Levels
        For I = LBound(GoodLevels) To UBound(GoodLevels)
            If StrComp(GoodLevels(I), myLevel.Name, vbTextCompare) = 0 Then
                LevelFound(I) = True
            End If
        Next I
    Next
    For I = LBound(GoodLevels) To UBound(GoodLevels)
        If LevelFound(I) = False Then
            Print #FFile, ActiveDesignFile.Path & ActiveDesignFile.Name _
                    & vbTab & "MISSING LEVEL: " & GoodLevels(I)
        End If
    Next I
    Close #FFile
End Sub
```

我们把结果写入一个 ASCII 文件中，在文件路径/文件名与层之间放一个制表符使得这个 ASCII 文件以制表符为分隔符。为什么要以制表符为分隔符呢？因为这样做使得该文件能轻易地导入到 Microsoft Excel 和其它程序中，这些程序是具有读取以制表符为分隔符的 ASCII 文件的程序。

接下来干什么？考虑打开数百个 DGN 文件并一个个运行该宏所带来的痛苦吧。VBA 正是用于解决此类问题并使工作轻松和更加愉快的。

对过程做个小改动，就是以附加方式打开 ASCII 文件。当以输出方式打开文件时，现有文件被覆盖；当以附加方式打开文件时，向现有文件(如果存在的话)中添加内容，如果文件事先不存在就创建它。

```
Sub FindUnsupportedLevelsC(FileToQuery As DesignFile)
    Dim GoodLevels(0 To 4) As String
    Dim GoodLevelsJ As String
    Dim myLevel As Level
```

```
    Dim FFile As Long
    GoodLevels(0) = "ROADWAY"
    GoodLevels(1) = "SIDEWALK"
    GoodLevels(2) = "PAINT"
    GoodLevels(3) = "ELECTRIC"
    GoodLevels(4) = "GAS"
    GoodLevelsJ = UCase("~" & Join(GoodLevels, "~") & "~")
    FFile = FreeFile
    Open "C:\MicroStation VBA\LevelsUnSupported.txt" For Append _
                                                    As #FFile
    For Each myLevel In FileToQuery.Levels
        If InStr(1, GoodLevelsJ, "~" & UCase(myLevel.Name) & "~") = 0 Then
            Print #FFile, FileToQuery.Path & FileToQuery.Name & _
                    vbTab & "Unsupported Level Found: " & myLevel.Name
        End If
    Next
    Close #FFile
End Sub

Sub FindMissingLevelsC(FileToQuery As DesignFile)
    Dim GoodLevels(0 To 4) As String
    Dim LevelFound(0 To 4) As Boolean
    Dim myLevel As Level
    Dim I As Long
    Dim FFile As Long
    GoodLevels(0) = "ROADWAY"
    GoodLevels(1) = "SIDEWALK"
    GoodLevels(2) = "PAINT"
    GoodLevels(3) = "ELECTRIC"
    GoodLevels(4) = "GAS"
```

China BDN

```
    GoodLevelsJ = UCase("~" & Join(GoodLevels, "~") & "~")
    FFile = FreeFile
    Open "C:\MicroStation VBA\LevelsMissing.txt" For Append As #FFile
    For Each myLevel In FileToQuery.Levels
        For I = LBound(GoodLevels) To UBound(GoodLevels)
            If StrComp(GoodLevels(I), myLevel.Name, vbTextCompare) = 0 Then
                LevelFound(I) = True
            End If
        Next I
    Next
    For I = LBound(GoodLevels) To UBound(GoodLevels)
        If LevelFound(I) = False Then
            Print #FFile, FileToQuery.Path & FileToQuery.Name & _
                          vbTab & "MISSING LEVEL: " & GoodLevels(I)
        End If
    Next I
    Close #FFile
End Sub
```

除了以附加方式打开文件外，我们还给这些过程提供了一个参数。这能让我们指定一个用于查询层的 DesignFile 对象。

怎样打开一个目录中的数百个文件？在继续编写程序之前，向 VBA 工程中添加一个引用：从 VBA 菜单中选择 **Tools > References(工具 > 引用)**并从按字母排序的列表中选择“Microsoft Scripting Runtime”。一旦引入了 Microsoft Scripting Runtime(微软脚本运行时)，我们就能使用文件系统对象遍历一个文件夹中的文件。

```
Sub DoFilesInFolder()
    Dim MyFSO As New FileSystemObject
    Dim myFolder As Folder
    Dim myFile As File
    Set myFolder = MyFSO.GetFolder("C:\Documents and Settings\" & _
```


China BDN

```
        "All Users\Application Data\Documents\Bentley\WorkSpace\" & _
        "Projects\Examples\Architectural\Dgn")
    For Each myFile In myFolder.Files
        Select Case UCase(Right(myFile.Name, 3))
            Case "DGN"
                Dim myDGNFile As DesignFile
                Set myDGNFile = OpenDesignFileForProgram(myFile.Path)
                FindUnsupportedLevelsC myDGNFile
                FindMissingLevelsC myDGNFile
                myDGNFile.Close
        End Select
    Next
End Sub
```

过程 DoFilesInFolder 取一个硬编码的文件路径，并以 “ForProgram” 方式打开路径中的每个 DGN 文件。这就意味着不在 MicroStation 编辑窗口中打开这些文件也能用 VBA 来操作它们。

上面所示例子打开安装 MicroStation 时添加到硬盘上的文件。在不到一秒钟内打开了七个文件，识别了层并生成了文本文件。

使用上述例子能激发你的创造力。除层以外，还有其它的元素能用来维护标准。实际上，讨论验证所有准则的例子需要一整本书。由于这一点不是本书的重点，就把其它的功能留给您的想象吧。

有关信息的更多说明参见三十章“批处理”。

跨公司的标准

两家公司需要相互使用对方的文件，但一个公司有一个叫作“STREET”的层，而另一家有一个叫作“Level 20”层。VBA 如何帮助两家公司使用不同的标准呢？

过程 LevelSpecA 把层名从一个标准翻译成另一个标准。当该过程发现“Level 20”时，就把名字改成“STREET”。其他层名的映射也都很容易明白。

```
Sub LevelSpecA()
    Dim myLevel As Level
    Dim I As Long
    For I = 1 To ActiveDesignFile.Levels.Count
        Set myLevel = ActiveDesignFile.Levels(I)
        Select Case myLevel.Name
            Case "Level 20"
                myLevel.Name = "STREET"
                ActiveDesignFile.Levels.Rewrite
            Case "Level 21"
                myLevel.Name = "SIDEWALK"
                ActiveDesignFile.Levels.Rewrite
            Case "Level 23"
                myLevel.Name = "GUTTER"
                ActiveDesignFile.Levels.Rewrite
            Case "Level 38"
                myLevel.Name = "STRIPING"
                ActiveDesignFile.Levels.Rewrite
            Case "Level 39"
                myLevel.Name = "SEWER"
                ActiveDesignFile.Levels.Rewrite
            Case "Level 40"
                myLevel.Name = "PHONE"
                ActiveDesignFile.Levels.Rewrite
            Case "Level 41"
                myLevel.Name = "ELECTRIC"
                ActiveDesignFile.Levels.Rewrite
            Case "Level 42"
                myLevel.Name = "NATGAS"
                ActiveDesignFile.Levels.Rewrite
```


China BDN

```
            Case "Level 47"
                myLevel.Name = "FIBER"
                ActiveDesignFile.Levels.Rewrite
        End Select
    Next I
End Sub
```

这一过程运行后，设计文件就满足了公司对层名的标准。我们能用自己开发的 VBA 工具来影响文件。我们的设计师和绘图员不需要参照一个“对照单”来记忆层与层的对应关系。他们看到的层仍然是他们习惯的那些层，这样极大地提高了生产力。

在将文件返回给原公司之前，把层名设置回原来的样子，这样做很礼貌。

```
Sub LevelSpecB()
    Dim myLevel As Level
    Dim I As Long
    For I = 1 To ActiveDesignFile.Levels.Count
        Set myLevel = ActiveDesignFile.Levels(I)
        Select Case myLevel.Name
            Case "STREET"
                myLevel.Name = "Level 20"
                ActiveDesignFile.Levels.Rewrite
            Case "SIDEWALK"
                X myLevel.Name = "Level 21"
                ActiveDesignFile.Levels.Rewrite
            Case "GUTTER"
                myLevel.Name = "Level 23"
                ActiveDesignFile.Levels.Rewrite
            Case "STRIPING"
                myLevel.Name = "Level 38"
                ActiveDesignFile.Levels.Rewrite
            Case "SEWER"
```

```
                myLevel.Name = "Level 39"
                ActiveDesignFile.Levels.Rewrite
            Case "PHONE"
                myLevel.Name = "Level 40"
                ActiveDesignFile.Levels.Rewrite
            Case "ELECTRIC"
                myLevel.Name = "Level 41"
                ActiveDesignFile.Levels.Rewrite
            Case "NATGAS"
                myLevel.Name = "Level 42"
                ActiveDesignFile.Levels.Rewrite
            Case "FIBER"
                myLevel.Name = "Level 47"
                ActiveDesignFile.Levels.Rewrite
        End Select
    Next I
End Sub
```

这段代码能运行吗？当然能！它运行良好！速度很快！令人惊奇！但它也有许多硬编码的映射难于维护！

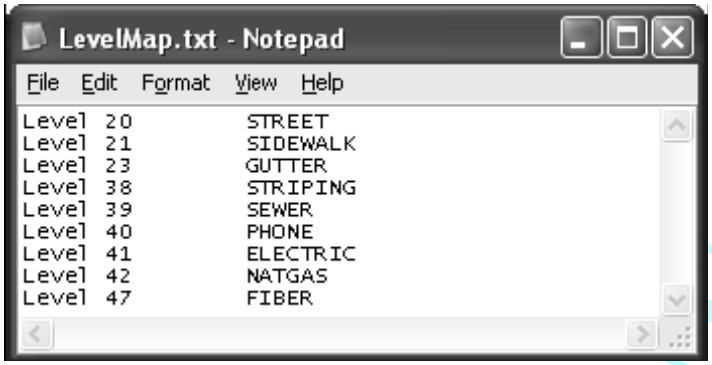
当需要翻译另一个层名时将会发生什么？您不得不修改代码并把这个 VBA 工程重新分发给每个使用它的人。这样做对吗？可能不对。

想一想如果建立一个层映射文件，该文件中保存要找的层与新层名的对应关系，这种情况下我们该如何来做。

如果本章讨论数据库，就可以在数据库中完成这种映射；如果本章讨论 Microsoft Excel，就可以在 Excel 中完成这种映射。但由于这些主题都将在后续章节中讨论，所以我们目前从一个简单的 ASCII 文本文件中读取。

现在来看层映射文件的格式。文件中的每一行代表一个映射，每行含有两个以制表符分隔的字段。第一个字段是旧层名，第二个字段是新层名。

China BDN



让我们看一看利用层映射文件的代码。

```
Sub LevelSpecFromFile()
    Dim myFile As String
    Dim OldLevel() As String
    Dim NewLevel() As String
    Dim FFile As Long
    Dim txtIn As String
    Dim xSplit() As String
    ReDim OldLevel(0)
    ReDim NewLevel(0)
    Dim myLevel As Level
    Dim I As Long
    Dim J As Long
    myFile = "C:\MicroStation VBA\LevelMap.txt"
    FFile = FreeFile
    Open myFile For Input As #FFile
    While EOF(FFile) = False
        Line Input #FFile, txtIn
        If InStr(1, txtIn, vbTab) > 0 Then
            xSplit = Split(txtIn, vbTab)
                OldLevel(UBound(OldLevel)) = xSplit(0)
                NewLevel(UBound(NewLevel)) = xSplit(1)
                ReDim Preserve OldLevel(UBound(OldLevel) + 1)
                ReDim Preserve NewLevel(UBound(NewLevel) + 1)
```

```
        End If
    Wend
    ReDim Preserve OldLevel(UBound(OldLevel) - 1)
    ReDim Preserve NewLevel(UBound(NewLevel) - 1)
    Close #FFile
    For I = 1 To ActiveDesignFile.Levels.Count
        Set myLevel = ActiveDesignFile.Levels(I)
        For J = LBound(OldLevel) To UBound(OldLevel)
            If StrComp(OldLevel(J), myLevel.Name) = 0 Then
                myLevel.Name = NewLevel(J)
                ActiveDesignFile.Levels.Rewrite
                Exit For
            End If
        Next J
    Next I
End Sub
```

以上就是代码。不管文本文件中有多少层，可以是 5 层或者 5000 层，这段代码都不需要改变。

跟踪时间

时间就是金钱，对吗？这种说法我们听过多少次？或许正因为它是真理才使得我们听了这么多遍。学习 VBA 的一个好处是用 VBA 做许多次事情会比不用 VBA 要快得多。

每个人对其在一张图纸中或一个工程上所花费时间的关心度是不同的，这往往由我们和图纸的关系来确定。

绘图员

一个绘图员可能把其在图纸中花费的时间看作其工作所获报酬的多少。“画图时间越多，薪水就会越高”。

另一个绘图员可能把其在图纸中花费的时间看作生产力的体现。“我比别的

China BDN

绘图员工作效率高一倍”。

经理

一位经理可能把在图纸中花费的时间量看作是绘图员或设计师需要额外培训的标志，或者是一个绘图员/设计师需要教公司中其他人提高工作效率的标志。

另一位经理可能会把在图纸中花费的时间量看作工程进度的依据。

会计师

一位会计师可能会把在图纸中所花费的时间看作要付给客户多少钱的依据。

另一位会计师可能会把在图纸中所花费的时间作为提高和设置薪水标准的参考。

无论我们在公司中扮演着什么角色，跟踪时间这个基本点都是相同的。如果我们努力工作(直到背痛和肌肉抽筋)，精确的时间报告总是需要的。

那么，跟踪时间的基本点是什么？是谁在什么时候做了什么。每当打开 MicroStation 时，您能知道是谁登录了计算机；每当想捕获数据时，您能获取当前的日期和时间。所以唯一需要回答的问题是“做了什么？”。我们需要捕获什么事件呢？

我们以前讨论过用接口捕获用户输入、元素选择等。您能记录用户启动的每个命令，从某种角度来看，这样做有助于研究 MicroStation 的可用性，但对于我们这里要实现的功能来说可能是小题大做了。

您能捕获文件打开和文件关闭事件。这对计算工作时间有帮助，但不足够或者会引起误解，特别是用于记账时就更是如此。假如某个人在星期五的下午 4:59 打开了一个文件然后整个周末一直让它开着会怎么样呢？

最好是捕获某种事件，它在文件打开时能知道文件是否在使用。让我们试着监视一下试图更新事件。该事件没有被经常触发，记录它的信息不会造成性能问题。

要捕获事件就需要一个新的类模块，把这个类模块命名为 **clsTimeTrack**。下面是类模块中的代码：

```
Implements IViewUpdateEvents
Dim WithEvents MSApp As Application
```

```
Private Sub Class_Initialize()
    Dim FFile As Long
    FFile = FreeFile
    Open "C:\MicroStation VBA\TimeTrack.txt" For Append As #FFile
        Print #FFile, "INIT" & vbTab & Now
    Close #FFile
    Set MSApp = Application
End Sub
Private Sub Class_Terminate()
    Dim FFile As Long
    FFile = FreeFile
    Open "C:\MicroStation VBA\TimeTrack.txt" For Append As #FFile
        Print #FFile, "TERM" & vbTab & Now
    Close #FFile
End Sub
Private Sub IViewUpdateEvents_AfterRedraw(TheViews() As View, _
                                          TheModels() As ModelReference, _
                                          ByVal DrawMode As MsdDrawingMode)
    If DrawMode = msdDrawingModeNormal Then
        Dim FFile As Long
        FFile = FreeFile
        Open "C:\MicroStation VBA\TimeTrack.txt" For Append As #FFile
        Print #FFile, "REDR" & vbTab & Now & vbTab & _
                      Application.UserName & vbTab & _
                      ActiveDesignFile.Name
        Close #FFile
    End If
End Sub
Private Sub IViewUpdateEvents_BeforeRedraw(TheViews() As View, _
                                           TheModels() As ModelReference, _
```

```
                                    ByVal DrawMode As MsdDrawingMode)
End Sub
Private Sub MSApp_OnDesignFileClosed(ByVal DesignFileName As String)
    Dim FFile As Long
    FFile = FreeFile
    Dim XSplit() As String
    XSplit = Split(DesignFileName, "\")
    Open "C:\MicroStation VBA\TimeTrack.txt" For Append As #FFile
    Print #FFile, "CLOS" & vbTab & Now & vbTab & _
                  Application.UserName & vbTab & XSplit(UBound(XSplit))
    Close #FFile
End Sub
Private Sub MSApp_OnDesignFileOpened(ByVal DesignFileName As String)
    Dim FFile As Long
    Dim XSplit() As String
    XSplit = Split(DesignFileName, "\")
    FFile = FreeFile
    Open "C:\MicroStation VBA\TimeTrack.txt" For Append As #FFile
    Print #FFile, "OPEN" & vbTab & Now & vbTab & _
                Application.UserName & vbTab & XSplit(UBound(XSplit))
    Close #FFile
End Sub
```

让我们看一下所捕获的事件：

Class_Initialize	第一次打开类时触发。
Class_Terminate	类被终止时触发。
IViewUpdateEvents_AfterRedraw	IViewUpdateEvents 接口的一个成员。该类作为一个事件处理器，当重绘视图时触发该事件。
OnDesignFileClosed	MicroStation 应用对象的一个成员。当关

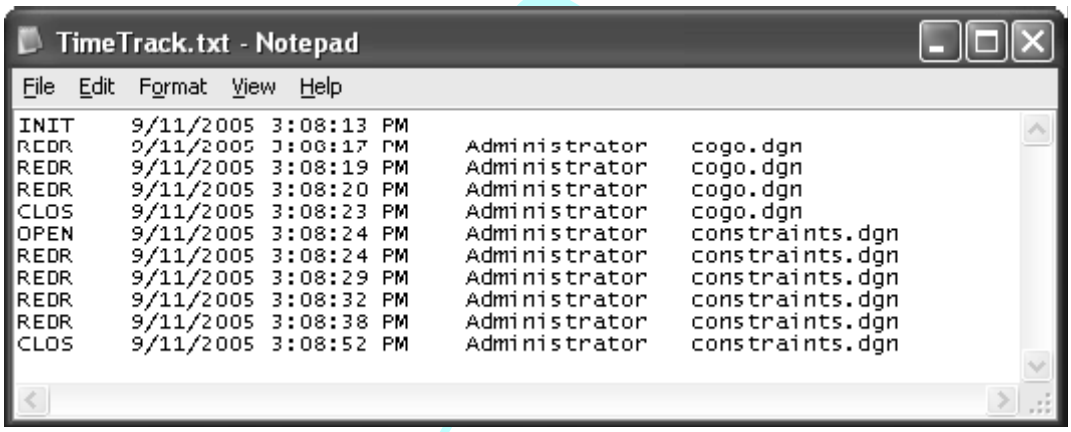
	闭一个文件时，给出了文件名。
OnDesignFileOpened	MicroStation 应用对象的一个成员。当打开一个文件时，给出了文件名。

正如代码所表达的那样，我们把事件记录数据写入到一个 ASCII 文本文件中并给每个捕获的事件赋以四字母缩写，如“INIT”、“TERM”、“REDR”、CLOS”和“OPEN”。这些缩写告诉我们在其指定的日期和事件发生了什么。文本文件中的每个字段以一个制表符分隔。

因为代码是在一个类模块中编写的，所以需要在代码模块中用一段代码来调用这个类，这样这个类才能捕获事件。

```
Sub TestTimeTrack()
    Set myME = New clsTimeTrack
    AddModelActivateEventsHandler myME
End Sub
```

下图是这个类的输出结果：



对记录 DGN 文件全路径做个小改动。在打印语句中用参数“DesignFileName” 取代分割文件名。

自动装载和自动运行

从计算机用户的角度来看，没有几件事情比告诉他们说正有软件在跟踪其计算机的使用情况更可怕的了。如果测试以上代码，会看到它运行正常，但不希望由用户来启动这个宏。不需用户的任何干涉就能装载和执行会更有效率。



自动装载工程很容易。在 VBA 工程管理器(Utilities > Macro > Project Manager)中，点击 Auto Load(自动装载)列能设置或取消任何工程的 AutoLoad 状态。MicroStation 下次打开时，任何标记为 AutoLoad 的 VBA 工程都将被装载。

既然工程被自动装载，那么如何自动运行一个过程呢？

```
Sub OnProjectLoad()
    TestTimeTrack
End Sub
```

MicroStation VBA 中的 OnProjectLoad 特性很容易使用。输入过程名“OnProjectLoad()”，然后是过程中的执行部分，这部分内容将在 MicroStation 启动后执行。

虽然 OnProjectLoad 是个极好的功能，但要有限制地使用它。VBA 不管 OnProjectLoad 中的代码做什么事情，它只是开始执行这些代码。如果我们的代码启动数据处理、打开并关闭文件然后又处理另外的事请，这将出现问题。其结果是，当用户启动 MicroStation 后就开始执行自动装载工程中的 OnProjectLoad 过程里的代码，但看起来好像 MicroStation 被锁定了一样。

保存工程后退出 MicroStation，再次打开 MicroStation 将会使 INIT 事件登记到您的记录文件中。

目前，这段代码运行于您开发的机器上。马上把这个新工程放到一个共享驱动器上让每个人今天就开始使用，这件事想起来多么令人激动。您从一台机器走到另一台机器，装载这个工程并设置为自动装载状态，您不告诉用户为什么要这么做而让他们关闭 MicroStation 再重新进入。您一台台机器这么设置，但一件事情突然发生了。

Windows 允许共享文件夹和文件，但当多于 20 个人试图同时打开同一个文件时，Windows 就开始不干了。那么，如何支持 100 个用户呢？把 VBA 工程放到每个用户的机器上？不，请不要这么做。

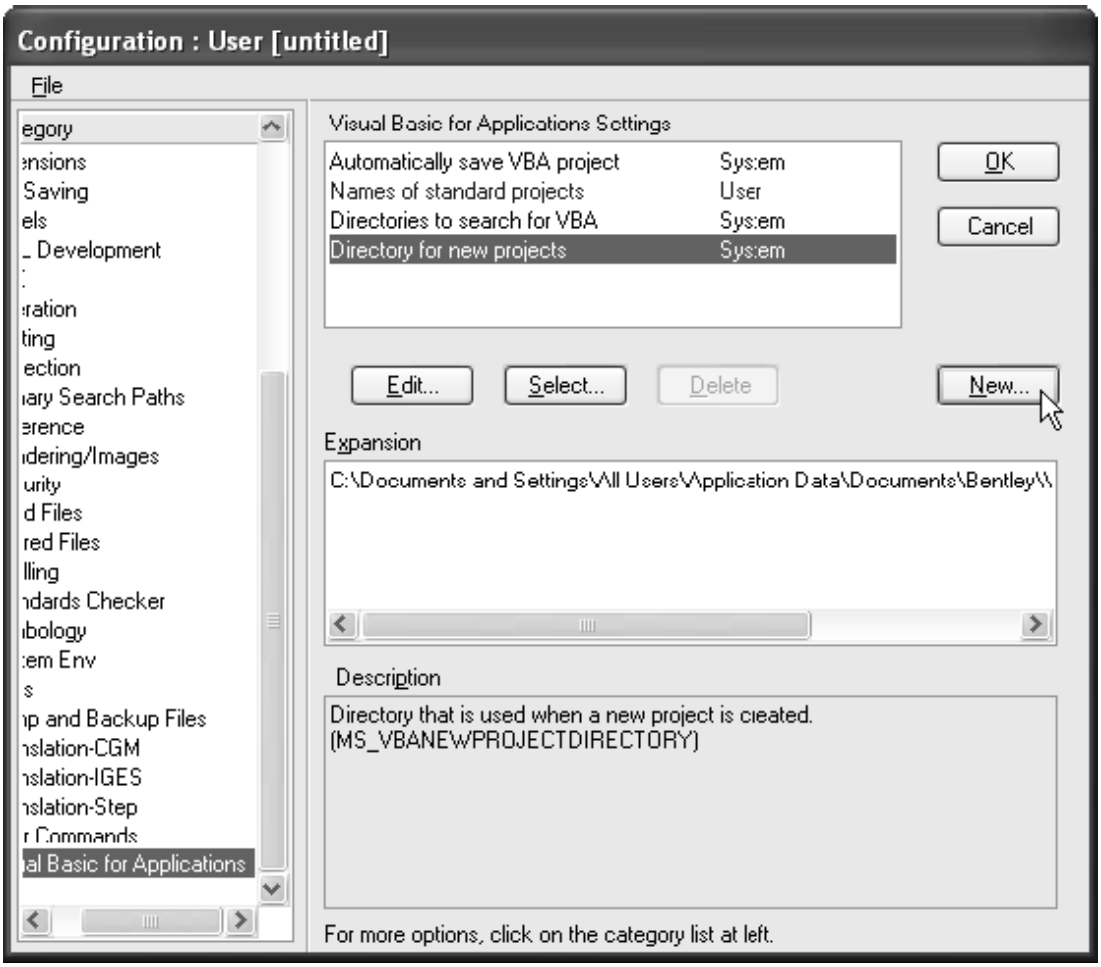
MS_VBA_OPEN_IN_MEMORY

MS_VBA_OPEN_IN_MEMORY 配置变量能让我们指定 VBA 工程在什么时候应该在内存中打开而不是维护一个工程文件句柄。缺省情况下，该变量未被定义，因为从磁盘打开文件通常不成为问题。

让我们讨论一下该变量可接受的值，定义它并设置其值。

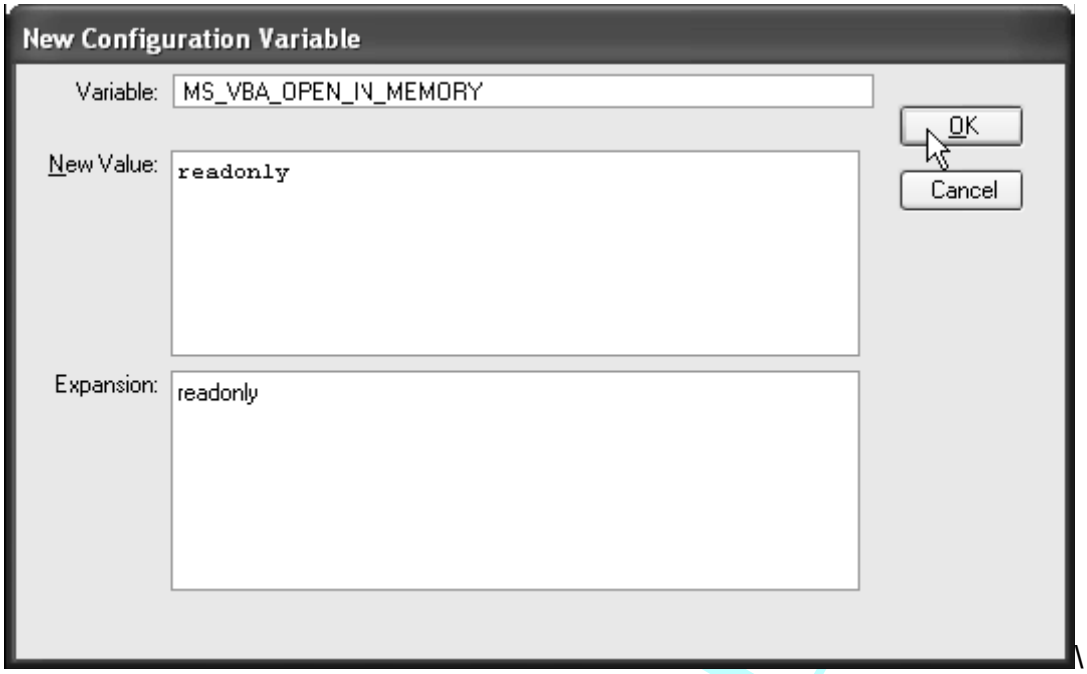
all	在内存中打开所有的 VBA 工程。该项消除了对 20 用户的限制并阻止用户编辑现有的 VBA 工程，因为这些 VBA 工程仅存在于内存中。
readonly	在内存中打开只读的 VBA 工程。可读写工程从文件中打开。
none	从不在内存中打开 VBA 工程。所有工程都从文件打开。

要添加 MS_VBA_OPEN_IN_MEMORY 变量，需要转到 MicroStation 的 **Workspace > Configuration(工作空间 > 配置)**菜单项，然后滚动到 Visual Basic for Application 类别。

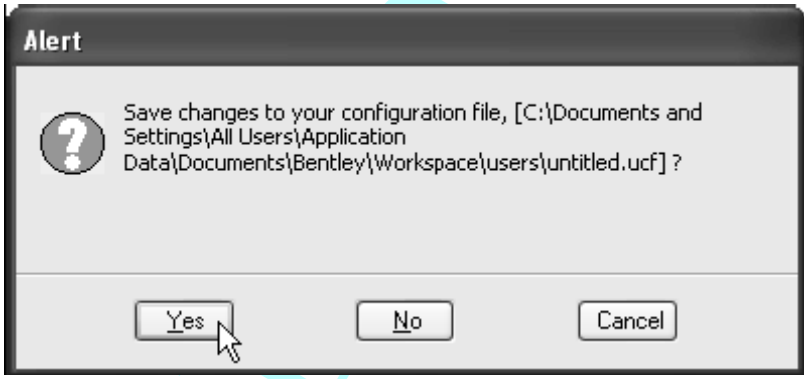


从上图中您能看到已经在用的属于 VBA 的 MicroStation 配置变量。

点击 New(新建)按钮添加新的配置变量。

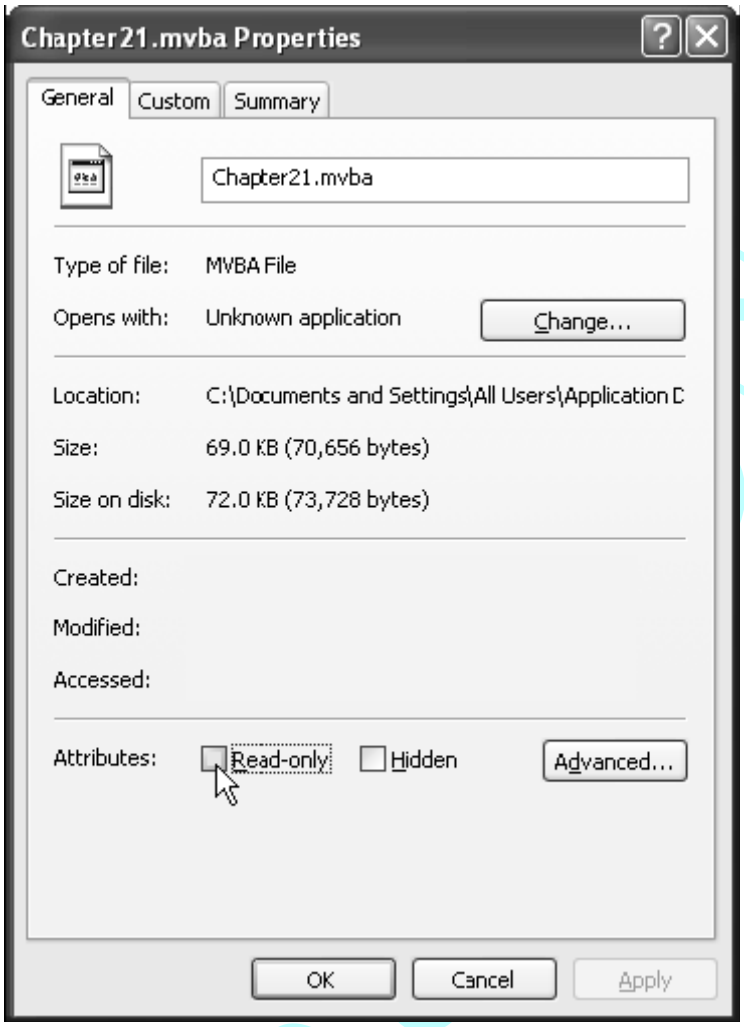


变量名是 MS_VBA_OPEN_IN_MEMORY。我们讨论过它的值：这里输入“readonly”以管理 VBA 文件在内存中打开。点击 OK 按钮返回配置对话框，在配置对话框中点击 OK 保存配置。



点击 Yes 保存配置文件。被标记为“只读”的文件将在内存中被打开。

标记文件为只读很简单。先在 VBA 工程管理器中卸载它，文件被关闭后在 Windows 资源管理器中浏览到它，右击该文件并选择“属性”。



只读复选框可选择也可取消选择。现在不要选择只读，因为对该工程还要作更多的修改。当完成一个 VBA 工程后把它设置为“只读”是个好主意。点击 OK(确定)或 Cancel(取消)按钮退出文件属性对话框。

用 VBA 工程管理器再次打开该文件继续工作。

保护工程

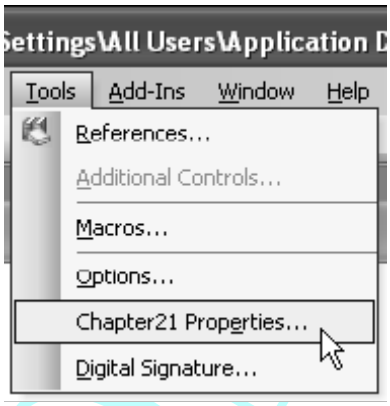
通常，我们不赞成把口令写在纸上。为什么呢？因为我们不相让别人发现这张纸并知道口令。虽然保护口令的概念是正确的，但忘记被保护工程的口令将会... 将会... 是破坏性的。

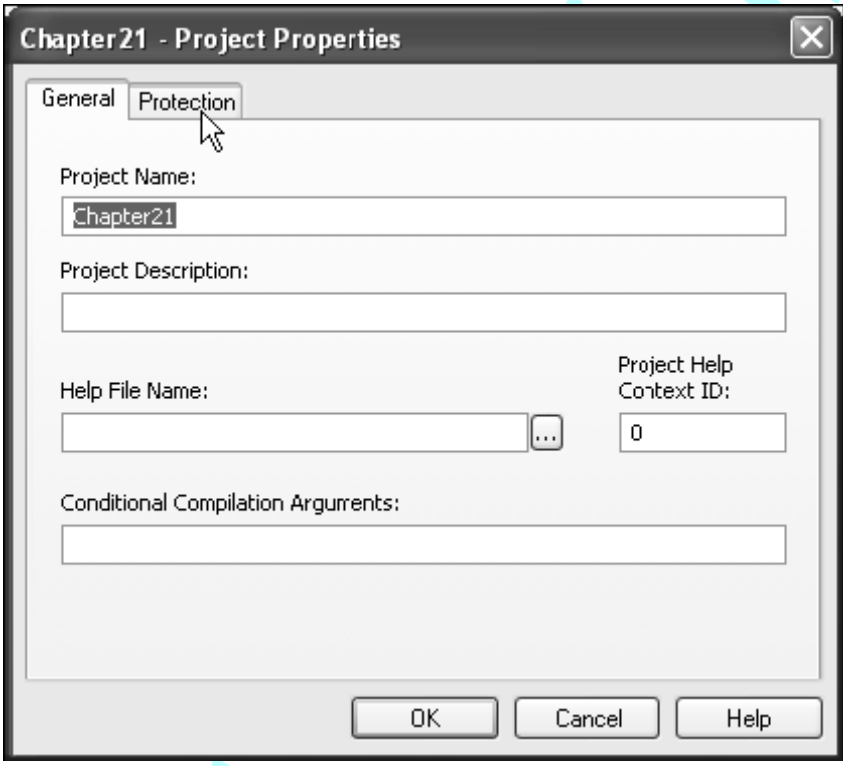
用口令保护一个完成了的工程是个好主意。不管是经理还是市场人员，我们都不想让辛苦得到的代码被别人得到。

1、在 VBA 菜单中选择 **Tools > Project Properties(工具 > 工程属性)**。

属性菜单项随工程的不同而不同；

2、点击属性菜单项显示 VBA 工程属性窗口；





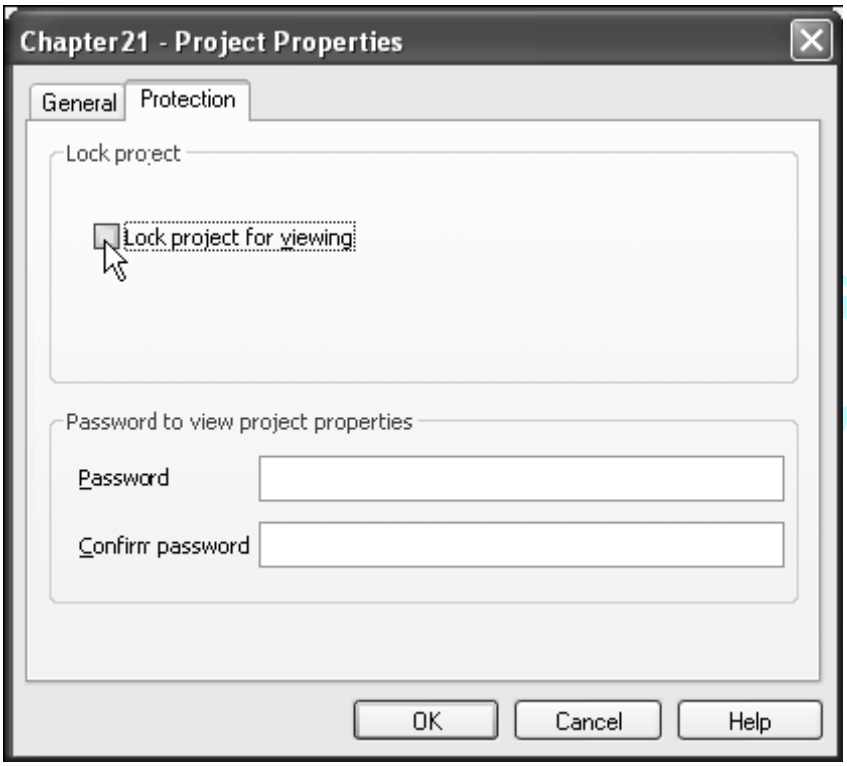
以上对话框中出现了两个选项卡。第一个选项卡，General(通用)，给出工程名和工程描述。帮助和编译区超出了本书的范围，不在此讨论它们。

使用 Protection(保护)选项卡来设置工程的口令。

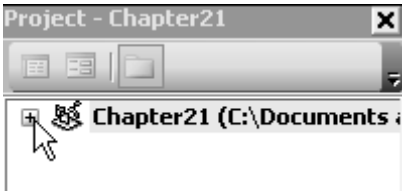
把工程设置为“Locked for viewing(查看锁定)”使工程能被装载，其中的过程也能从 VBA 工程管理器中执行但不能在 VBA 环境中打开修改，除非提供了正确的口令。

3、选择复选框来锁定对当前工程的查看并输入一个口令，使用超强口令“dorami”。输入一个口令并点击 OK 按钮来锁定对工程的查看；

4、保存工程后卸载它，然后再用 VBA 工程管理器重新装载；



装载工程后回到 VBA 工程管理器，您在工程列表中看到工程的工程树是合起来的。





5、在加号上点击以展开工程时会要求您输入口令。现在的口令是什么？写在某个地方吗？啊哈，在这儿呢：“dorami”。

成功输入口令后您又能存取工程中的代码了。

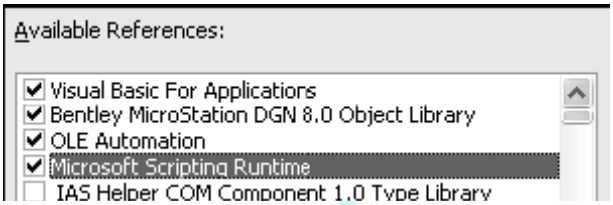
记住，您不能修改以只读方式打开的工程，也不能修改在内存中打开的工程。

分发 VBA 工程

VBA 工程包含在单个.mvba 文件中，这使得分发很容易。只要把一个文件电邮给世界上任何具有相同版本 MicroStation 的人，他们就能使用您的程序。或许没有这么简单。

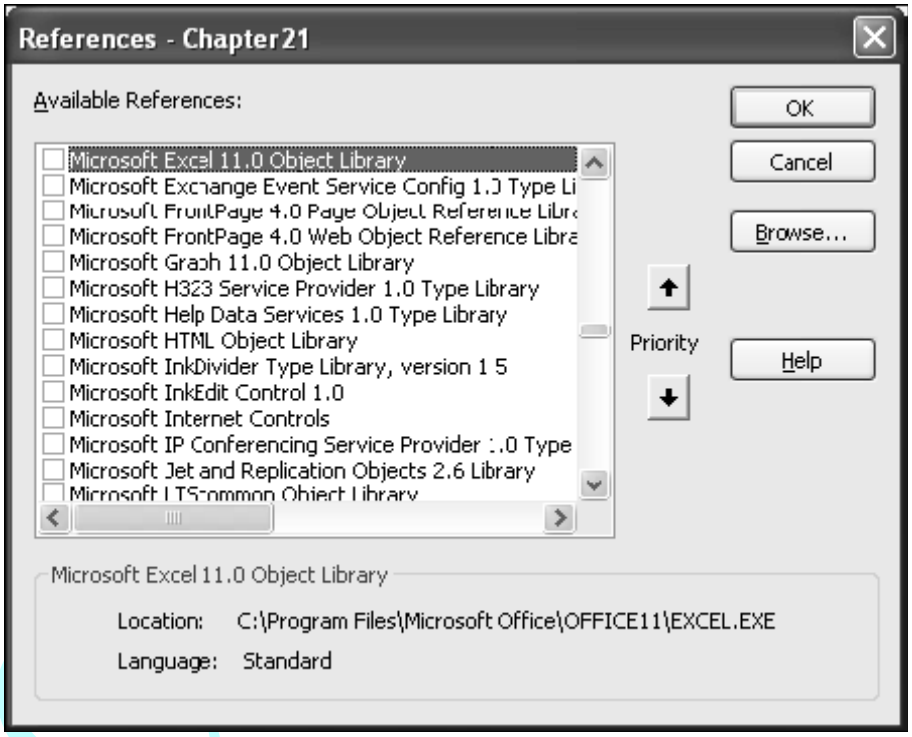
从 VBA 菜单选择 **Tools > References(工具 > 引用)**显示引用对话框。

引用对话框的一部分：



每个选取的引用都指向一个.dll 或.exe 文件，这些文件具有我们的程序中使用的功能。最顶上的三项出现在每个 MicroStation VBA 工程中。我们在本章早些时候添加了 Microsoft Scripting Runtime 库。在分发工程前要看一下添加到工程中的引用，因为其他人的计算机上缺少这些引用文件将会造成问题。

例如，当使用 Microsoft Excel 时，我们添加引用“Microsoft Excel 11.0 Object Library”，这个引用能帮助我们在使用 Excel 时能更快更准确地进行开发。这个程序在开发的机器上运行得很好，但把它放到一台没有安装 Microsoft Excel 的计算机上时会发生一些奇怪的事情(并不仅仅是关于 Excel 的代码)。像 UCase、LCase、Trim 等函数都显示出错误信息，告诉我们未装载它们的库。



所以，分发 VBA 应用时应明确知道选择了哪些引用，让用户知道为了使您的程序运行正常他们需要安装哪些程序。

在高安全模式下工作

以高安全模式启动 MicroStation 很有好处。下面就是两个好处：

1、点击 **Utilities > Macro > Visual Basic Editor(实用程序 > 宏 > Visual Basic 编辑器)**试着进入 VBA 编辑区时将出现下面的错误：



2、试图打开一个未签名的工程时出现下面的错误：



为什么是这些好处？因为我们不能运行我们自己的代码。

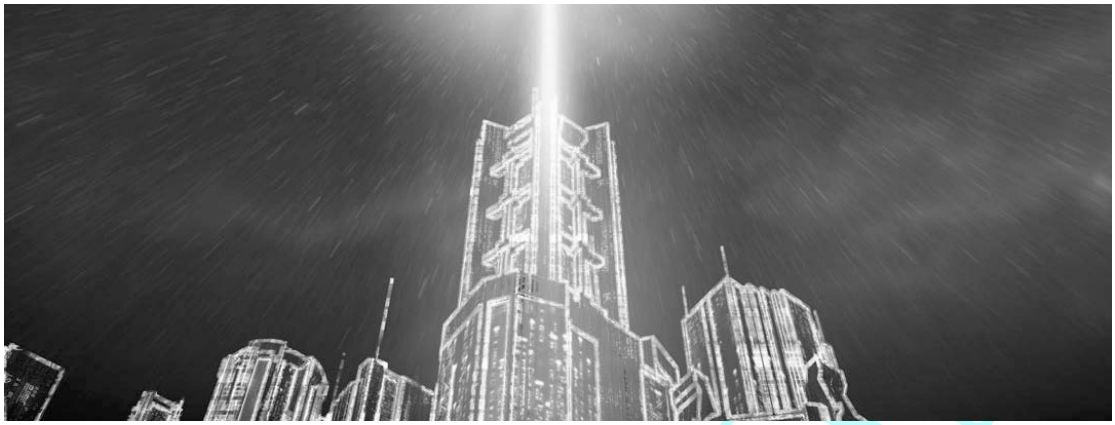
由于编写破坏 MicroStation DGN 文件和 Windows 系统的代码很容易，所以高安全模式不允许装载和执行未签名的工程。那下一个问题当然会是“怎样对我的工程进行签名？”这是个好问题。然而答案不在此提供，因为在本书中解释如何对一个 VBA 工程签名会使每个 VBA 工程都能被签名，这将破坏高安全模式的目的，也达不到签名 VBA 工程的目的。

在心里要记住的主要事情是，高安全意味着高度的安全性 — 我们建立的 VBA 工程不能运行在高安全模式下的 MicroStation 中。

本章回顾

从一位 CAD 管理者的角度看，VBA 能做许多有助于维护 CAD 标准的事情。维持生产力是重要的，有时测量生产力也是很重要的。保护 VBA 工程的意思是从 VBA 工程中获得生产力并且不让不怀好意者执行您的代码。自动装载和自动执行代码功能强大且易于实现。最难的事情是知道什么时候能把这些强大的特性应用于您最优势的地方。

China BIM



22. MicroStation 中基于文件的事件

每当发生一个特定事件时都能执行代码，例如每当打开文件时把文件名写入一个 ASCII 文本文件。本章采用一些非常强大的编程技术来处理 MicroStation 中基于文件的事件。

我们主要分析 14 章中曾提及的这些事件中的两个，但这里讨论得更详细。

本章包括：

- OnDesignFileOpened 事件
- OnDesignFileClosed 事件
- ISaveAsEvents_BeforeRemap 事件
- ISaveAsEvents_AfterRemap 事件
- ISaveAsEvents_AfterSaveAs 事件

OnDesignFileOpened

OnDesignFileOpened 事件是 MicroStation 应用对象的一部分。每当打开一个设计文件时，OnDesignFileOpened 事件就被触发。让我们给该事件添加一些非常简单的代码。

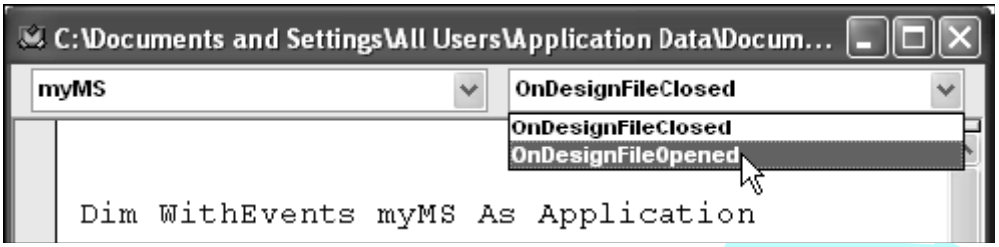
开始之前，该事件必须用“WithEvents”关键字声明一个“Application”对象类型的变量。

建立一个新的类模块，起名为 **clsSaveAs**。

```
Dim WithEvents myMS As Application
```

这个声明在什么地方？在类模块 **clsSaveAs** 的通用声明区。

声明完 **myMS** 变量后，该对象的事件就在类模块的过程复选框中显示。



在复选框中选择 OnDesignFileOpened 将自动声明 OnDesignFileOpened 事件。

```
Private Sub myMS_OnDesignFileOpened(ByVal _
                                    DesignFileName As String)
    Dim ffile As Long
    ffile = FreeFile
    Open "C:\MicroStation VBA\FileOpen.txt" For Append As #ffile
     Print #ffile, Now & vbTab & DesignFileName
    Close #ffile
End Sub
```

这里就是 OnDesignFileOpened 事件的代码。我们将日期/时间和文件名写入位于 C:\MicroStation VBA\FileOpen.txt 的 ASCII 文本文件中。

编写代码很简单但事件中的代码不能像代码模块中的代码那样直接执行。本章后续部分将讨论如何让一个类“站起来跑”。(译者注：英文中跑和运行都是 run)

OnDesignFileClosed

该事件在文件被关闭时触发。您的代码可以像 OnDesignFileOpened 事件那样写 ASCII 文本文件，但有更强大和有用的功能可做。

让我们多想想 OnDesignFileClosed 事件都能干些什么？用户在 MicroStation 中关闭了一个文件。它是今天工作的结束吗？是本周最后一次打开这个文件吗？文件打开后做了什么改变？关闭文件时谁正在使用它？什么时间关闭的？今天早些时候打开过这个文件吗？

用 OnDesignFileClosed 事件能编写出 100 个不同的应用，但让我们先挑选一个：当文件关闭时捕获它。您当然能复制文件到一个文件夹中，但这将比专门致力于文件压缩占用更多的硬盘空间。如果压缩文件(压缩成一个.zip 文件)将节

省一些磁盘空间。把文件放入一个现有的压缩文件夹或文件中更有用。

那么如何将多个 MicroStation DGN 文件压缩成单个 zip 文件呢？VBA 不支持这一强大的功能，对吗？喔，我们可以花钱买第三方的 DLL 或 ActiveX 控件，但是当不需要它时你会花钱买吗？

Windows XP 引入了压缩(zipped)文件夹的概念，其本质是 Windows 将一个 zip 文件(.zip 扩展名)看作一个文件夹。可以用标准的 Windows 功能复制和粘贴文件进出一个压缩的文件夹。让我们在 VBA 中用 Windows XP 的新功能来压缩文件吧。

编写代码前，在 VBA(Tools > Reference)中添加引用“Microsoft Shell Controls and Automation”。该引用是微软为开发者(当前情况下，我们是开发者)开发的强大功能。Shell32.dll 对象具有许多功能，本章讨论的几个事件仅仅是揭开了她的一点面纱。

正如本章稍后将要看到的那样，往现有 zip 文件(压缩文件夹)中添加文件很容易；但如果该文件不存在，就需要建立它。

我们每天使用的几乎所有的文件都具有两个识别特征。第一个是文件扩展名，例如，我们本能地把.dgn 扩展名的文件作为 MicroStation 文件，但也能强行将一个.txt 文件的扩展名改为.dgn。所以虽然文件扩展名是文件类型的一个好的指示，但并不能确保文件的类型。第二个识别特征是文件头，文件头由文件开始部分特定的几个字节组成，这几个字节帮助程序来确认文件类型。

zip 文件(或压缩的 zip 文件夹)头由 24 个字节组成。怎么知道这一点的？在 Windows 中建立一个新的压缩文件夹，然后在一个十六进制编辑器中打开这个文件查看其中每个字节的值。zip 文件中的前四个字节值是 80、75、5 和 6，后 20 个字节都是 0(零)。所以我们可以写 Chr(80)、Chr(75)、Chr(5)、Chr(6)和 20 个 Chr(0)值来建立一个空的 zip 文件。这样做就能写一个 zip 文件头到一个文件中，该文件能被 Windows XP 按扩展名和文件头来识别。其它的 zip 文件读写器也能把该文件认作合法的 zip 文件。当然，一个空的 zip 文件是没有什么用的。

zip 文件建立后(如果它不存在的话)，就能用 Shell 库复制文件到这个“文件/文件夹”中了。

把过程 CopyFileToZipFile 放入类模块中而不是代码模块。

```
Sub CopyFileToZipFile(ZipFile As String, FileToCopy As String, _
                                                CopyFileAs As String)
        Dim ffile As Long
        Dim myShell As New Shell
        Dim zipFolder As Shell32.Folder3
        ffile = FreeFile
        If Dir(ZipFile) = "" Then
            Open ZipFile For Output As #ffile
            Print #ffile, Chr(80) & Chr(75) & Chr(5) & Chr(6) & _
                Chr(0) & Chr(0) & Chr(0) & Chr(0) & _
                Chr(0) & Chr(0) & Chr(0) & Chr(0) & _
                Chr(0) & Chr(0) & Chr(0) & Chr(0) & _
                Chr(0) & Chr(0) & Chr(0) & Chr(0) & _
                Chr(0) & Chr(0) & Chr(0) & Chr(0)
            Close #ffile
        End If
    Set zipFolder = myShell.Namespace(ZipFile)
    If StrComp(FileToCopy, CopyFileAs) <> 0 Then
        FileCopy FileToCopy, CopyFileAs
        zipFolder.CopyHere CopyFileAs
         Kill CopyFileAs
    Else
        zipFolder.CopyHere CopyFileAs
    End If
End Sub
```

第一件事是检查.zip 文件是否存在。如果不存在就建立该文件并用标准的文件 I/O(输入/输出)命令向该文件中打印，这些命令在本书中已经使用过多次。

函数 CopyFileToZipFile 让我们指定要复制的文件以及在 zip 文件内的文件名，随后您就能看到指定.zip 文件内的文件名的重要性。变量 **myShell** 被声明

为“New Shell”对象，这样就公开了该引用对象的方法、属性和事件。我们通过 Shell 对象的 Namespace 方法用 zip 文件名取得一个文件夹。如果 **FileToCopy** 变量不同于 **CopyFileAs** 变量，我们就复制 **FileToCopy** 到 **CopyFileAs**，然后再复制 **CopyFileAs** 文件到 zip 文件，最后杀掉(删除)这个中间文件。如果 **FileToCopy** 变量和 **CopyFileAs** 变量相同，只需简单地复制这个设计文件到 zip 文件而不需要其它的复制和“杀掉”(这个函数听起来如此的暴力，不是吗？)。

> **注意**：http://www.codeproject.com/csharp/DecompressWinShellAPICS.asp 网页描述了用Shell对象处理Windows XP的压缩文件夹的问题。

CopyFileToZipFile 过程现在能被 OnDesignFileClosed 事件使用了。

```
Private Sub myMS_OnDesignFileClosed(ByVal _
                                    DesignFileName As String)
    CopyFileToZipFile "C:\MicroStation VBA\FileClosed.zip", _
                      DesignFileName, DesignFileName & "." & CLng(Timer * 1000)
End Sub
```

使用 CopyFileToZipFile 过程时要指定设计文件复制到的 zip 文件、作为源文件和目的文件的文件名(全路径)。源文件和目的文件的文件名可以相同，那么为何想要一个不同的 zip 文件名呢？

我们提供不同文件名的目的是让设计文件(如 asks.dgn)能在同一个 zip 文件中有多份，通过在文件名后附加一个数字实现这一点。什么数字？本例中用 Timer 函数取得自午夜至今的秒数，再把这个秒数乘以 1000 得到自午夜至今的毫秒数。这样能保证不会把相同的文件以相同的数字后缀复制到相同的 zip 文件中。

ISaveAsEvents 接口

ISaveAsEvents 接口包括 ISaveAsEvents_BeforeRemap 、ISaveAsEvents_AfterRemap 和 ISaveAsEvents_AfterSaveAs。

ISaveAsEvents_BeforeRemap 事件是 ISaveAsEvents 接口的成员。要使用接口的成员需要在类模块中声明该接口。下面是要放入类模块中的声明：

```
Implements ISaveAsEvents
```

正如前面章节讨论接口时说的那样，在往接口的任一事件中写代码前需要声明接口的所有事件(方法)。

```
Private Sub ISaveAsEvents_BeforeRemap( _
        ByVal TheDesignFile As DesignFile, _
        ByVal SavedFormat As MsdDesignFileFormat, _
        ByVal DestinationFilename As String)
End Sub
Private Sub ISaveAsEvents_AfterRemap( _
        ByVal TheDesignFile As DesignFile, _
        ByVal SavedFormat As MsdDesignFileFormat, _
        ByVal DestinationFilename As String)
End Sub
Private Sub ISaveAsEvents_AfterSaveAs()
End Sub
```

以上属于 ISaveAsEvents 接口的三个事件以它们被执行的顺序显示出来。

由于 AfterSaveAs 事件是最后执行的事件，所以它正是我们放置代码的地方。但在这样做之前让我们先分析一下这个事件。

在 BeforeRemap 和 AfterRemap 事件中，我们能知道 DesignFile(设计文件)、SavedFormat(保存格式)和 DestinationFilename(目的文件名)的参数值。由于 AfterSaveAs 事件中没有传递这些信息，所以需要在知道这些信息时就捕获它们。为此，在类模块的通用声明区声明三个变量来保存 AfterRemap 事件中给的值，这些值将在 AfterSaveAs 事件中使用。

```
Dim FileSaved As String
Dim FileFormat As Long
Dim FilePrevious As String
```

既然声明了这些变量，它们就能如下所示在 AfterRemap 事件中使用了：

```
Private Sub ISaveAsEvents_AfterRemap( _
        ByVal TheDesignFile As DesignFile, _
        ByVal SavedFormat As MsdDesignFileFormat, _
```

```
                ByVal DestinationFilename As String)
        FileSaved = DestinationFilename
        FileFormat = SavedFormat
        FilePrevious = TheDesignFile.FullName
    End Sub
```

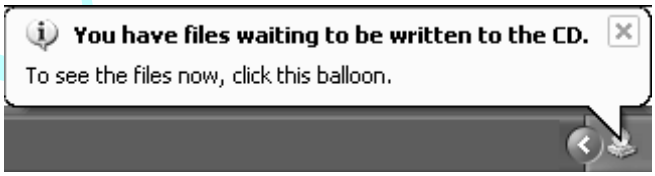
好了，我们捕获了“另存为”命令所用的设计文件(DesignFile)，还有文件将被保存的文件格式以及目的文件名(DestinationFilename)。用这些能做什么呢？

在前一个例子中，设计文件关闭时我们把设计文件添加到一个 zip 文件中。那是一个轻松搞定的功能强大的例子，仅用了几行 VBA 代码就实现了非常强大的功能(当然了，客观地说也夹杂进了一些其它方面的知识)。我们能在 AfterSaveAs 事件中完成那个功能吗？

又到了开动脑筋的时候了，因为可能性是无止境的。我们讨论的函数将在“TheDesignFile”参数或“DestinationFilename”参数上执行。我们能把文件绕半个地球 FTP 到服务器上；能把文件根据其所属工程通过电子邮件发送给不同的人；能打开文件、提取信息并把它放入数据库中。以上每个功能都很有用，且都很容易实现。让我们做点别的工作。

当用户执行“另存为”时，正打开的文件被给以一个新的文件名，且可选地另存为一个不同的文件类型。在当前情形下，当“另存为”一个文件为 AutoCAD 的.dwg 文件时，我们取得原始的设计文件(它也能是一个 AutoCAD 的.dwg 文件)并将它写入一个 CD 上。

我们再次使用 Windows XP 引入的新功能，它让我们复制并粘贴文件到一个可写的 CD 或 DVD 驱动器上。这样做时，文件被放入一个临时存储位置直到决定实际写入 CD 时为止。当“发送”一个文件到 CD 刻录机时，出现一个消息告诉我们有文件正在等待写入 CD：



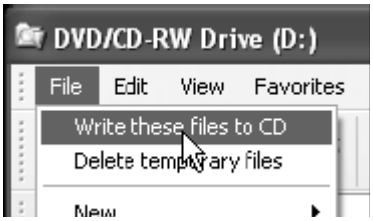
那么如何将文件放入这个临时存储位置呢？这个位置在哪儿呢？让我们在本例中使用 Shell 对象，这个对象刚刚在上个例子中使用过。

我们需要在上例用过的类模块中建立一个新的过程，该过程叫做 CopyFileToCD，只有一个参数就是要复制到 CD 的文件。下面是代码，讨论如下。

```
Sub CopyFileToCD(FileToCopy As String)
    Dim myShell As New Shell
    Dim cdFolder As Shell32.Folder3
    Set cdFolder = myShell.Namespace(59)
    If Not cdFolder Is Nothing Then
        cdFolder.CopyHere FileToCopy, 0
    End If
End Sub
```

第一件事是找到登录用户所用的 CD 文件的临时存储位置。我们现在是怎么做的呢？给 Shell 对象的 Namespace 方法提供数字 59 作为参数。如果这样做后返回的不是“Nothing”(换句话说，找到了该文件夹)，就可以复制文件到 **cdFolder** 文件夹了。

该过程的最大一个好处是，我们不需要购买 CD 刻录插件。只要用户用的是 Windows XP 就可以使用该过程。



请记住，CopyFileToCD 过程并不实际执行 CD 刻录 — 它只把文件复制到刻录 CD 的存储区。当准备刻录 CD 时，在 CD 刻录机中放入一张 CD-R 或 CD-RW，在 Windows 资源管理器中选择 CD 驱动器，在资源管理器菜单中执行 **File > Write these files to CD(文件 > 将这些文件写入 CD)**。

通过这些菜单选择就开始了把临时 CD 文件夹中的文件写入 CD 的过程。

那我们现在就有了准备把文件写入 CD 的代码。怎么使用呢？在 ISaveAsEvents_AfterSaveAs 事件中用它。

```
Private Sub ISaveAsEvents_AfterSaveAs()
    Select Case FileFormat
        Case msdDesignFileFormatCurrent
        Case msdDesignFileFormatDWG
            CopyFileToCD   FilePrevious
```

```
            Case msdDesignFileFormatDXF
            Case msdDesignFileFormatUnknown
            Case msdDesignFileFormatV7
            Case msdDesignFileFormatV8
        End Select
    End Sub
```

AfterSaveAs 事件是用户执行“另存为”时发生的最后一个事件，但该事件本身没有告诉我们源文件名、目标文件名或格式。我们在 AfterRemap 事件中收集这些信息，在 AfterSaveAs 事件中使用。

上例演示了如何使用建立的 CopyFileToCD 过程，但它仅当“另存为”命令用于保存一个 AutoCAD 的.dwg 文件时才执行。我们能在其它的“Case”语句下放上相同的代码行以适应别的文件格式，或者干脆不用 Select Case 结构，只要有“另存为”发生就使用 CopyFileToCD。

我们在几节中都提及新类模块 **clsSaveAs** 的功能，现在从头至尾看一看整个类模块。请记住，变量声明出现在类模块的通用声明区。特别注意变量 **FileSaved**、**FileFormat** 和 **FilePrevious** 的使用。

下面就是从头至尾的类模块代码：

```
Dim WithEvents myMS As Application
Implements ISaveAsEvents
Dim FileSaved As String
Dim FileFormat As Long
Dim FilePrevious As String
Private Sub Class_Initialize()
    Set myMS = Application
End Sub
Private Sub ISaveAsEvents_AfterRemap( _
                ByVal TheDesignFile As DesignFile, _
                ByVal SavedFormat As MsdDesignFileFormat, _
                ByVal DestinationFilename As String)
    FileSaved = DestinationFilename
```

```
    FileFormat = SavedFormat
    FilePrevious = TheDesignFile.FullName
End Sub
Private Sub ISaveAsEvents_AfterSaveAs()
    Select Case FileFormat
        Case msdDesignFileFormatCurrent
        Case msdDesignFileFormatDWG
            CopyFileToCD FilePrevious
        Case msdDesignFileFormatDXF
        Case msdDesignFileFormatUnknown
        Case msdDesignFileFormatV7
        Case msdDesignFileFormatV8
    End Select
End Sub
Private Sub ISaveAsEvents_BeforeRemap( _
            ByVal TheDesignFile As DesignFile, _
            ByVal SavedFormat As MsdDesignFileFormat, _
            ByVal DestinationFilename As String)
End Sub
Private Sub myMS_OnDesignFileClosed(ByVal _
                    DesignFileName As String)
    CopyFileToZipFile "C:\MicroStation VBA\FileClosed.zip", _
        DesignFileName, DesignFileName & "." & CLng(Timer * 1000)
End Sub
Private Sub myMS_OnDesignFileOpened(ByVal _
                                DesignFileName As String)
    Dim ffile As Long
    ffile = FreeFile
    Open "C:\MicroStation VBA\FileOpen.txt" For Append As #ffile
    Print #ffile, Now & vbTab & DesignFileName
```


China BDN

```
    Close #ffile
End Sub
Sub CopyFileToCD(FileToCopy As String)
    Dim myShell As New Shell
    Dim cdFolder As Shell32.Folder3
    Set cdFolder = myShell.Namespace(59)
    If Not cdFolder Is Nothing Then
        cdFolder.CopyHere FileToCopy, 0
    End If
End Sub
Sub CopyFileToZipFile(ZipFile As String, FileToCopy As String, _
                                          CopyFileAs As String)
    Dim ffile As Long
    Dim myShell As New Shell
    Dim zipFolder As Shell32.Folder3
    ffile = FreeFile
    If Dir(ZipFile) = "" Then
        Open ZipFile For Output As #ffile
        Print #ffile, Chr(80) & Chr(75) & Chr(5) & Chr(6) & _
            Chr(0) & Chr(0) & Chr(0) & Chr(0) & _
            Chr(0) & Chr(0) & Chr(0) & Chr(0) & _
            Chr(0) & Chr(0) & Chr(0) & Chr(0) & _
            Chr(0) & Chr(0) & Chr(0) & Chr(0) & _
            Chr(0) & Chr(0) & Chr(0) & Chr(0)
        Close #ffile
    End If
    Set zipFolder = myShell.Namespace(ZipFile)
    If StrComp(FileToCopy, CopyFileAs) <> 0 Then
        FileCopy FileToCopy, CopyFileAs
        zipFolder.CopyHere CopyFileAs
```


China BDN

```
            Kill CopyFileAs
        Else
            zipFolder.CopyHere CopyFileAs
        End If
    End Sub
```

到目前为止，已经编写了太多的代码，但仍不能运行它因为这些代码包含在类模块中，所以让我们讨论一下如何使用类模块代码。

您必须在代码模块或用户窗体中调用类模块中的代码。本例中我们使用代码模块。

```
    Sub TestSaveAs()
        Dim mySaveAs As New clsSaveAs
        AddSaveAsEventsHandler mySaveAs
    End Sub
```

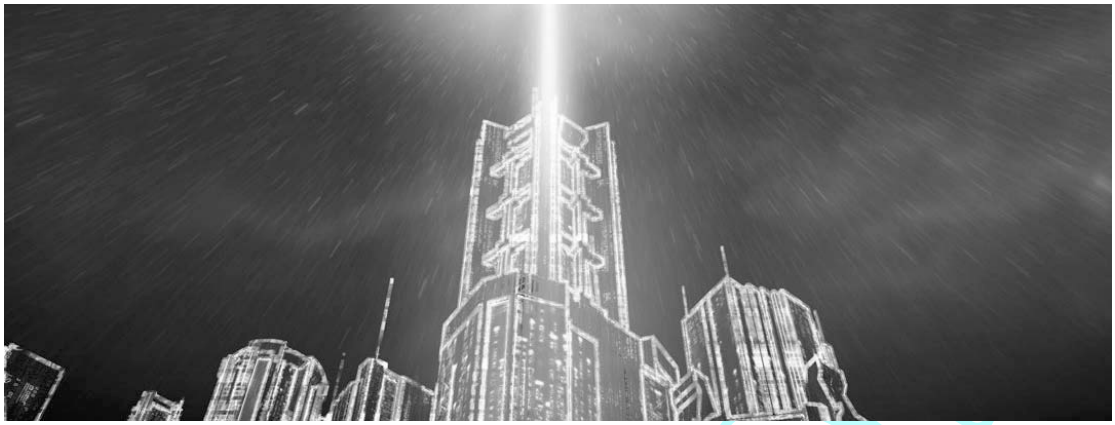
这段代码非常简单。声明一个 **clsSaveAs**(刚完成的类模块的名字)类型的变量并把它加入到 SaveAsEventsHandler。然后 **clsSaveAs** 对象被装入内存并开始响应 SaveAs(另存为)事件，也包括 OnDesignFileOpened 和 OnDesignFileClosed 事件。

本章回顾

在这一章中，使用 Windows XP 压缩文件并准备刻录到 CD 上。在其它操作系统上也能实现相同的功能但编程方法会不同。现在您知道了当用户执行“另存为”、打开文件或关闭文件时能执行一段代码，不管是什么操作系统或想实现什么功能。

在启动任何项目前都要花时间想一想您在该项目中要实现的功能，考虑最可能的编程方式。不要作茧自缚，因为你无法确定如何完成一个、二个、三个或更多个任务。即使最老的狗也能学习新的技巧 — 对于现在来说就是 VBA 技巧。

China BDN



23. 响应 MicroStation 附件事件

从本章开始的四章继续讨论对 MicroStation 中事件的响应，先从 IAttachmentEvents 接口开始。总共讨论了五个事件。

本章包括：

- IAttachmentEvents 接口
- AfterAttach
- AfterDetach
- AttachmentModified
- BeforeAttach
- BeforeDetach

IAttachmentEvents 接口

让我们建立一个新的类模块 **clsAttachmentEvents**。必须声明 IAttachmentEvents 接口实现的每个事件。在每个事件中添加简单的 Debug.Print 语句来帮助我们理解事件被触发的顺序。

```
Private Sub IAttachmentEvents_AfterAttach(ByVal _
                                          TheAttachment As Attachment)
    Debug.Print "AfterAttach"
End Sub
Private Sub IAttachmentEvents_AfterDetach(ByVal _
                                          TheAttachment As Attachment)
```

```
    Debug.Print "AfterDetach"
End Sub
Private Sub IAttachmentEvents_AttachmentModified(ByVal _
                                        TheAttachment As Attachment)
    Debug.Print "AttachmentModified"
End Sub
Private Sub IAttachmentEvents_BeforeAttach(FileName As String, _
                                        AllowAttachment As Boolean)
    Debug.Print "BeforeAttach"
End Sub
Private Sub IAttachmentEvents_BeforeDetach(ByVal _
                                        TheAttachment As Attachment)
    Debug.Print "BeforeDetach"
End Sub
```

两个事件与连接参考文件相关，两个事件与卸载参考文件相关。

AfterAttach

```
Private Sub IAttachmentEvents_AfterAttach(ByVal _
                                        TheAttachment As Attachment)
End Sub
```

一个 DesignFile(设计文件)连接到当前设计文件后，AfterAttach 事件被触发。该事件发生时我们能确知文件被实际连接并能进行处理。事件中提供了参数“TheAttachment”，所以能立即开始使用附件。

“TheAttachment”参数被声明为 Attachment 类型的对象。对这一对象理解得深入一些有助于我们知道用这个事件能做什么事情。该对象的属性和方法列表有好几页长，所以这里只看几个。

- Sub AddElement(Element As Element)
- Sub AddElements(Elements() As _Element)
- Property AttachName As String {read-only}
- Property Description As String

- Property DesignFile As DesignFile {read-only}
- Property DisplayFlag As Boolean
- Property Is3D As Boolean {read-only}
- Property IsActive As Boolean {read-only}
- Property IsReadOnly As Boolean {read-only}
- Property IsTrueScale As Boolean {read-only}
- Property Level As Level
- Property Levels As Levels {read-only}
- Property LogicalDescription As String
- Property LogicalName As String
- Property MasterOrigin As Point3d {read-only}
- Sub Move(Offset As Point3d, ApplyToClipElement As Boolean)
- Property Name As String
- Function Reattach(FileName As String, ModelName As _
  String) As Attachment
- Sub Redraw([DrawMode As MsdDrawingMode = _
  msdDrawingModeNormal])
- Sub RemoveElement(Element As Element)
- Sub ReplaceElement(OldElement As Element, _
  NewElement As Element)
- Property RevisionNumber As String
- Sub Rewrite()
- Sub Rotate(Pivot As Point3d, AboutX As Double, AboutY _
  As Double, AboutZ As Double, ViewSpecifier As Variant)
- Property Rotation As Matrix3d {read-only}
- Property ScaleFactor As Double
- Function Scan([ScanCriteria As ElementScanCriteria]) _
  As ElementEnumerator
- Sub SelectElement(Element As Element, [DisplayAsSelected _
  As Boolean = True])

➢ Property Transparency As Double

在看这些属性和事件的同时，考虑一下附件被连接后如何使用它们。

先看一下 DesignFile 属性。当附件连接后，可得到附件的 DesignFile 以及 DesignFile 的路径。让我们给当前模型添加带有附件文件名的日期/时间戳。

```
Private Sub IAttachmentEvents_AfterAttach(ByVal _
                                TheAttachment As Attachment)
    Dim TxtPt As Point3d
    TxtPt.X = 0
    TxtPt.Y = -100
    Dim RotMatrix As Matrix3d
    Dim myText As TextElement
    Set myText = CreateTextElement1(Nothing, Now & "..." & _
                                TheAttachment.DesignFile.Path, TxtPt, RotMatrix)
    ActiveModelReference.AddElement myText
End Sub
```

每当连接了一个附件时，我们就把附件的 DesignFile 路径作为文本写到 ActiveModelReference(当前模型参考)中。正如现在的代码所表达的那样，文本都被添加到了相同的地方，无论是第一个附件还是第一百个附件。我们能增加代码来改变文本插入点的位置等内容，但在此不深入讨论这个问题，而是转向另一个使用 AfterAttach 事件的例子。

```
Private Sub IAttachmentEvents_AfterAttach(ByVal _
                                TheAttachment As Attachment)
    Dim FFile As Long
    FFile = FreeFile
    Open "C:\MicroStation VBA\AttachmentLog.txt" For Append As #FFile
    Print #FFile, Now & vbTab & ActiveDesignFile.Path & vbTab & _
                                TheAttachment.DesignFile.Path
    Close #FFile
End Sub
```

China BDN

分析以上代码能够看出，我们把 ActiveDesignFile 的路径和附件的路径写入到一个叫做 AttachmentLog.txt 的 ASCII 文本文件中。您可用该巨鹿文件来跟踪文件的相互依赖性。

另一个例子演示了更多附件属性的使用：

```
Private Sub IAttachmentEvents_AfterAttach(ByVal _
                                    TheAttachment As Attachment)
    Dim PtA As Point3d
    Dim PtB As Point3d
    Dim myRange As Range3d
    Dim MyRec As LineElement
    Dim LinePts(0 To 4) As Point3d
    myRange = TheAttachment.Range(True)
    PtA = myRange.High
    PtB = myRange.Low
    LinePts(0).X = PtA.X: LinePts(0).Y = PtA.Y
    LinePts(1).X = PtB.X: LinePts(1).Y = PtA.Y
    LinePts(2).X = PtB.X: LinePts(2).Y = PtB.Y
    LinePts(3).X = PtA.X: LinePts(3).Y = PtB.Y
    LinePts(4).X = PtA.X: LinePts(4).Y = PtA.Y
    Set MyRec = Application.CreateLineElement1(Nothing, LinePts)
    ActiveModelReference.AddElement MyRec
End Sub
```

该例子在附件的外围矩形边框绘制一个矩形。

我们看了三个使用 AfterAttach 事件的例子。这三个例子不能同时存在与类中。保留您想看其运行的那个事件代码，把其它的 AfterAttach 例子注释掉。

和其它接口一样，需要给事件处理器添加一个类。我们工作用的类名是**clsAttachmentEvents**。下面是把类添加到事件处理器的代码：

```
Sub TestAttachmentsA()
    Dim myAE As New clsAttachmentEvents
    AddAttachmentEventsHandler myAE
```

```
End Sub
```

AfterDetach

两个事件与卸载附件有关：BeforeDetach 和 AfterDetach。AfterDetach 事件是对附件进行处理的最后机会。让我们尝试一些代码，这些代码通知人们附件已被卸载。

```
Private Sub IAttachmentEvents_AfterDetach(ByVal _
                                          TheAttachment As Attachment)
    Shell "c:\Program Files\Internet Explorer\iexplore.exe " & _
        """http://www.trackmydgndrawings.com/log.asp?filename=" & _
        ActiveDesignFile.Name & "&attachment=" & _
        TheAttachment.DesignFile.Name & """", vbMaximizedFocus
End Sub
```

Shell 函数是一个标准的 VBA 函数。本例用该函数执行 Internet Explorer 并转向一个假想的网站，提供参数 ActiveDesignFile.Name 和 TheAttachment.DesignFile.Name。如果 **log.asp** 文件能在 HTTP 请求中读取这些参数，我们就有了一个附件被卸载时跟踪文件的 Web 服务器。

AttachmentModified 事件

AttachmentModified 事件提供与 AfterAttach 和 AfterDetach 事件相同的信息，提供给我们的还是参数“TheAttachment”。该事件有助于跟踪对一个附件所作的修改，但不能帮助我们了解修改的种类，还需要另外的事件来取得更详细的信息。

```
Private Sub IAttachmentEvents_AttachmentModified( _
                                    ByVal TheAttachment As Attachment)
    Debug.Print "AttachmentModified"
End Sub
```

BeforeAttach 事件

```
Private Sub IAttachmentEvents_BeforeAttach(FileName As String, _
                                    AllowAttachment As Boolean)
```

BeforeAttach 事件在附件连接前触发。从事件中能得到 FileName，通过设置 AllowAttachment 参数为 False 还能有机会取消对附件的连接。

```
Private Sub IAttachmentEvents_BeforeAttach(FileName As String, _
                                           AllowAttachment As Boolean)
    If GetAttr(FileName) And vbReadOnly Then
        MsgBox "Attaching Read-Only Files is Prohibited."
        AllowAttachment = False
    End If
End Sub
```

在这个例子中，如果被连接的文件设置为只读，我们显示一个消息框指明不允许连接只读文件。消息框显示完后，设置 AllowAttachment 参数为 False，这意味着连接对话框不会呈现给用户。如果被连接的文件不是只读的，我们就什么也不做，连接文件的过程会继续进行。

BeforeDetach 事件

```
Private Sub IAttachmentEvents_BeforeDetach(ByVal _
                                           TheAttachment As Attachment)
```

我们需要讨论的最后一个事件是 BeforeDetach 事件。同样是一个附件对象作为事件的参数提供给我们。该事件仅在 AfterDetach 事件之前被触发。

我们不能阻止卸载连接的发生。所以能做的一件事情就是记录卸载发生这个实事。

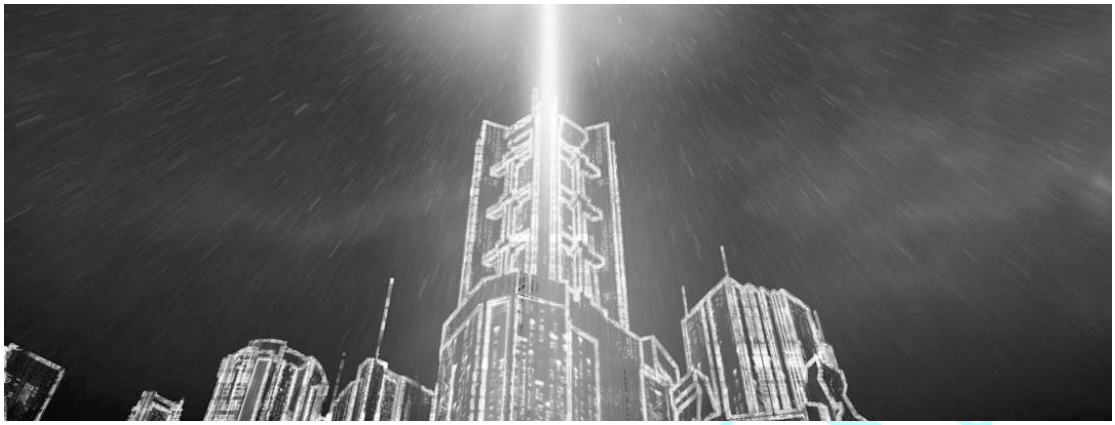
```
Private Sub IAttachmentEvents_BeforeDetach(ByVal _
                                           TheAttachment As Attachment)
    Dim FFile As Long
    FFile = FreeFile
    Open "C:\MicroStation VBA\DetachmentLog.txt" For Append As #FFile
    Print #FFile, Now & vbTab & Application.UserName & _
                  vbTab & TheAttachment.DesignFile.FullName
    Close #FFile
End Sub
```

本例捕获日期和时间、卸载文件的用户以及被卸载文件的全名，把这些信息写入到了一个记录文件。

本章回顾

连接一个文件到现有的设计文件上这个功能很强大，它能让我们设计得更快且出错更少。准确性得以提高，因为同一时间内能看到整个设计，如在结构基础上绘制墙线。用 IAttachmentEvents 接口能截取事件，这样我们就能跟踪哪个文件被连接、哪个被卸载以及哪个被修改。

China BDI



24. 模型事件

上一章处理附件事件，本章处理模型事件。两个独立的接口公开了与模型相关的事件：IModelActivateEvents 和 IModelChangeEvents。我们将在同一个类模块 **clsModelEvents** 中实现这两个接口。

本章包括：

- AfterActivate 事件
- BeforeActivate 事件
- ModelChange 事件

在同一个类模块中实现每个接口使得显示这些事件被触发的顺序较为容易。下面是类模块 **clsModelEvents** 的代码：

```
Implements IModelActivateEvents
Implements IModelChangeEvents
Private Sub IModelActivateEvents_AfterActivate(ByVal TheModel _
                                               As ModelReference)
    Debug.Print "AfterActivate: " & vbTab & vbTab & _
                TheModel.DesignFile.Name & vbTab & TheModel.Name
End Sub
Private Sub IModelActivateEvents_BeforeActivate(ByVal TheModel As _
                                                ModelReference)
    Debug.Print "BeforeActivate: " & vbTab & _
                TheModel.DesignFile.Name & vbTab & TheModel.Name
```

```
End Sub
Private Sub IModelChangeEvents_ModelChange(ByVal TheModel As _
                ModelReference, ByVal Change As MsdModelChangeType)
    Debug.Print "Change: " & vbTab & vbTab & vbTab & _
        TheModel.DesignFile.Name & vbTab & TheModel.Name & " - " & _
        ModelChange(Change)
End Sub
Function ModelChange(ChangeIn As MsdModelChangeType) As String
    Select Case ChangeIn
        'Active
        Case MsdModelChangeType.mdlModelChangeBeforeActive
            ModelChange = "BeforeActive"
        Case MsdModelChangeType.mdlModelChangeActive
            ModelChange = "Active"
        'Create
        Case MsdModelChangeType.mdlModelChangeBeforeCreate
            ModelChange = "BeforeCreate"
        Case MsdModelChangeType.mdlModelChangeCreate
            ModelChange = "Create"
        'Delete
        Case MsdModelChangeType.mdlModelChangeBeforeDelete
            ModelChange = "BeforeDelete"
        Case MsdModelChangeType.mdlModelChangeDelete
            ModelChange = "Delete"
        'Name
        Case MsdModelChangeType.mdlModelChangeBeforeName
            ModelChange = "BeforeName"
        Case MsdModelChangeType.mdlModelChangeName
            ModelChange = "Name"
        'Properties
```

```
        Case MsdModelChangeType.mdlModelChangeBeforeProperties
            ModelChange = "BeforeProperties"
        Case MsdModelChangeType.mdlModelChangeProperties
            ModelChange = "Properties"
        'Settings
        Case MsdModelChangeType.mdlModelChangeBeforeSettings
            ModelChange = "BeforeSettings"
        Case MsdModelChangeType.mdlModelChangeSettings
            ModelChange = "Settings"
        'UnCreate
        Case MsdModelChangeType.mdlModelChangeBeforeUnCreate
            ModelChange = "BeforeUnCreate"
        Case MsdModelChangeType.mdlModelChangeUnCreate
            ModelChange = "UnCreate"
        'UnDelete
        Case MsdModelChangeType.mdlModelChangeBeforeUnDelete
            ModelChange = "BeforeUnDelete"
        Case MsdModelChangeType.mdlModelChangeUnDelete
            ModelChange = "UnDelete"
        'PropagateAnnotationScale
        Case MsdModelChangeType.mdlModelChangePropagateAnnotationScale
            ModelChange = "PropagateAnnotationScale"
    End Select
End Function
```

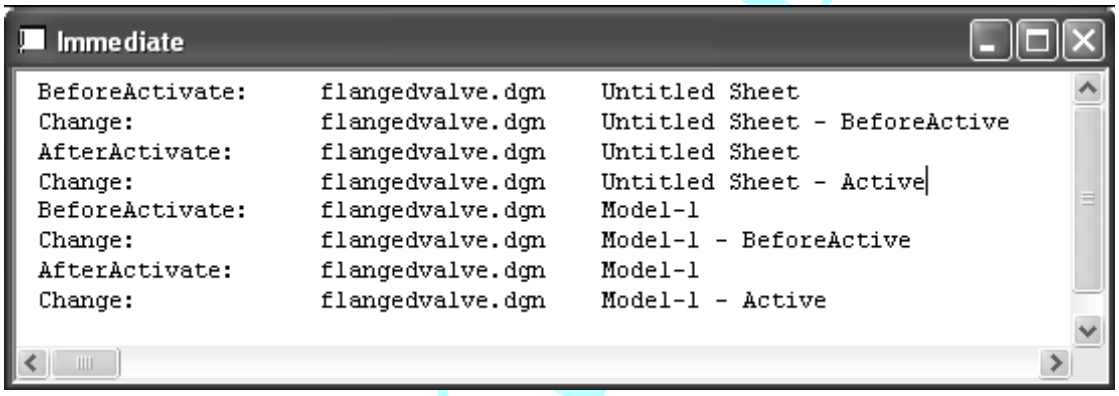
函数 ModelChange 接收到一个常数值并把它转换为文本，这能方便人们看出发生了什么。正如其它接口一样，类模块必须在代码模块中被实例化：

```
Private myME As clsModelEvents
Sub AddEvents()
    RemoveEvents
    Set myME = New clsModelEvents
```


China BDN

```
        AddModelActivateEventsHandler myME
        AddModelChangeEventsHandler myME
    End Sub
    Sub RemoveEvents()
        If myME Is Nothing = False Then
            RemoveModelActivateEventsHandler myME
            RemoveModelChangeEventsHandler myME
            Set myME = Nothing
        End If
    End Sub
```

在设计文件中切换模型时，从立即窗口中能看到如下结果：



事件的顺序是：

IModelActivateEvents_BeforeActivate

IModelChangeEvents_ModelChange (With BeforeActive Event)

IModelActivateEvents_AfterActivate

IModelChangeEvents_ModelChange (With Active Event)

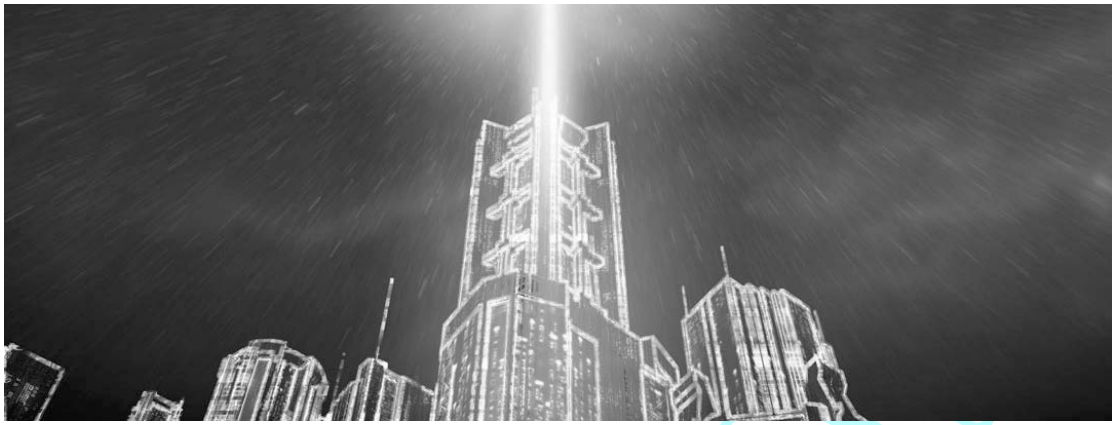
每个事件都提供有一个 ModelReference 参数，所以能知道哪个模型要被激活。改变当前模型时，ModelChange 事件被触发了两次：第一次带有 BeforeActivate 常数，第二次带有 Active 常数。

本章回顾

模型事件也和其它事件一样是通过接口实现的。添加简单的 Debug.Print 语句就能帮助我们弄明白事件是如何、何时和为何被触发的。一小点创造性再加上

一些 VBA 的技巧就能生成强大的解决方法。

China BDN



25. 层事件

层对于设计文件的组织来说是如此的重要，以至于为层事件专门设计了一个接口。

ILevelChangeEvents 接口仅公开了一个事件，该事件处理下面十二种类型的改变：

- AfterChangeActive
- AfterCreate
- AfterDelete
- BeforeChangeActive
- BeforeDelete
- ChangeAttribute
- ChangeCode
- ChangeDisplay
- ChangeName
- ChangeParent
- TableRedo
- TableUndo

虽然改变类型名是自解释的，但在继续前最好看一下实际的事件。

```
Private Sub ILevelChangeEvents_LevelChanged( _
        ByVal ChangeType As MsdLevelChangeType, _
        ByVal TheLevel As Level, ByVal TheModel As ModelReference)
```

China BIM

End Sub

在 LevelChanged 事件中传递给我们了三个参数。第一个是改变类型，以枚举常数给出。ChangeType 参数的值如下：

MsdLevelChangeType.msdLevelChangeAfterChangeActive = 9

MsdLevelChangeType.msdLevelChangeAfterCreate = 2

MsdLevelChangeType.msdLevelChangeAfterDelete = 3

MsdLevelChangeType.msdLevelChangeBeforeChangeActive = 17

MsdLevelChangeType.msdLevelChangeBeforeDelete = 18

MsdLevelChangeType.msdLevelChangeChangeAttribute = 8

MsdLevelChangeType.msdLevelChangeChangeCode = 5

MsdLevelChangeType.msdLevelChangeChangeDisplay = 7

MsdLevelChangeType.msdLevelChangeChangeName = 4

MsdLevelChangeType.msdLevelChangeChangeParent = 6

MsdLevelChangeType.msdLevelChangeTableRedo = 15

MsdLevelChangeType.msdLevelChangeTableUndo = 14

先从一个函数开始，该函数将以上常数转换为一个字符串并用简单的 Debug.Print 语句打印输出。这能让我们实现接口并查看层事件何时、如何被处理。

```
Implements ILevelChangeEvents
'before change active shows the old level name
Private Sub ILevelChangeEvents_LevelChanged( _
            ByVal ChangeType As MsdLevelChangeType, _
            ByVal TheLevel As Level, ByVal TheModel As ModelReference)
    Debug.Print TheLevel.Name & vbTab & TheModel.DesignFile.Name & _
        vbTab & TheModel.Name & vbTab & GetChangeType(ChangeType)
End Sub
Function GetChangeType(ChangeIn As MsdLevelChangeType) As String
    Select Case ChangeIn
        Case MsdLevelChangeType.msdLevelChangeAfterChangeActive
            GetChangeType = "AfterChangeActive"
```

```
        Case MsdLevelChangeType.msdLevelChangeAfterCreate
            GetChangeType = "AfterCreate"
        Case MsdLevelChangeType.msdLevelChangeAfterDelete
            GetChangeType = "AfterDelete"
        Case MsdLevelChangeType.msdLevelChangeBeforeChangeActive
            GetChangeType = "BeforeChangeActive"
        Case MsdLevelChangeType.msdLevelChangeBeforeDelete
            GetChangeType = "BeforeDelete"
        Case MsdLevelChangeType.msdLevelChangeChangeAttribute
            GetChangeType = "ChangeAttribute"
        Case MsdLevelChangeType.msdLevelChangeChangeCode
            GetChangeType = "ChangeCode"
        Case MsdLevelChangeType.msdLevelChangeChangeDisplay
            GetChangeType = "ChangeDisplay"
        Case MsdLevelChangeType.msdLevelChangeChangeName
            GetChangeType = "ChangeName"
        Case MsdLevelChangeType.msdLevelChangeChangeParent
            GetChangeType = "ChangeParent"
        Case MsdLevelChangeType.msdLevelChangeTableRedo
            GetChangeType = "ChangeTableRedo"
        Case MsdLevelChangeType.msdLevelChangeTableUndo
            GetChangeType = "ChangeTableUndo"
    End Select
End Function
```

下列代码放在一个代码模块中，它用于调用类“clsLevelEvents”。

```
Private myLC As clsLevelEvents
Sub AddEvents()
    Set myLC = New clsLevelEvents
    AddLevelChangeEventsHandler myLC
End Sub
```

```
    Sub RemoveEvents()
        If myLC Is Nothing = False Then
            RemoveLevelChangeEventsHandler myLC
            Set myLC = Nothing
        End If
    End Sub
```

运行 AddEvents 将类 **clsLevelEvents** 加到 LevelChangeEventsHandler。如果我们现在对 MicroStation 设计文件中的层作出修改的话，在立即窗口中将能看到几行反映事件的信息。

因为在一个类模块中仅能有一个 LevelChanged 事件，所以实现该事件的最佳途径是使用 Select Case 语句。

```
Private Sub ILevelChangeEvents_LevelChanged( _
        ByVal ChangeType As MsdLevelChangeType, _
        ByVal TheLevel As Level, ByVal TheModel As ModelReference)
    Select Case ChangeType
        Case MsdLevelChangeType.msdLevelChangeAfterChangeActive
        Case MsdLevelChangeType.msdLevelChangeAfterCreate
        Case MsdLevelChangeType.msdLevelChangeAfterDelete
        Case MsdLevelChangeType.msdLevelChangeBeforeChangeActive
        Case MsdLevelChangeType.msdLevelChangeBeforeDelete
        Case MsdLevelChangeType.msdLevelChangeChangeAttribute
        Case MsdLevelChangeType.msdLevelChangeChangeCode
        Case MsdLevelChangeType.msdLevelChangeChangeDisplay
        Case MsdLevelChangeType.msdLevelChangeChangeName
        Case MsdLevelChangeType.msdLevelChangeChangeParent
        Case MsdLevelChangeType.msdLevelChangeTableRedo
        Case MsdLevelChangeType.msdLevelChangeTableUndo
    End Select
End Sub
```

现在做好了用 Select Case 语句处理层改变的准备。我们在该接口中不能禁止层的改变，只能对改变事件作出反应。

让我们看几个本接口中实用的事件：

Active 事件

```
Case MsdLevelChangeType.msdLevelChangeAfterChangeActive
    Dim FFile As Long
    FFile = FreeFile
    Open "c:\levelactivated.txt" For Append As #FFile
    Print #FFile, Now & vbTab & TheLevel.Name & vbTab & _
            TheModel.Name & vbTab & TheModel.DesignFile.FullName
    Close #FFile
```

这段代码生成一个简单的记录文件，当前层被改变时记录日期/时间、层名、模型名和设计文件的全名。

AfterCreate 事件

```
Case MsdLevelChangeType.msdLevelChangeAfterCreate
    Dim FFile2 As Long
    FFile2 = FreeFile
    Open "c:\levelcreated.txt" For Append As #FFile2
    Print #FFile2, Now & vbTab & Application.UserName & vbTab & _
            TheLevel.Name & vbTab & TheModel.Name & vbTab & _
            TheModel.DesignFile.FullName
    Close #FFile2
```

知道建立了一个层很有用，知道是谁建立了这个层可能有些要求苛刻。该事件在层被建立后触发，生成了一个记录文件，该文件中记录有日期/时间戳、建立该层的用户和其它一些信息。

AfterDelete 事件

```
Case MsdLevelChangeType.msdLevelChangeAfterDelete
        MsgBox "Level " & TheLevel.Name & " has been deleted."
```

该事件被触发时对层做不了太多事情，但仍能取得层名。

BeforeChangeActive 事件

```
Case MsdLevelChangeType.msdLevelChangeBeforeChangeActive
    Debug.Print "Level """ & TheLevel.Name & _
                """ is about to be deactivated."
```

该事件的名字好像是要告诉我们哪个层将要成为当前层。其实不是这样的— 它告诉我们哪个层将要不成为当前层。在 AfterChangeActive 事件之前我们不知道哪个层将要成为当前层。

BeforeDelete 事件

```
Case MsdLevelChangeType.msdLevelChangeBeforeDelete
    Debug.Print "Before Delete: " & TheLevel.Name & vbCr & _
                vbTab & TheLevel.Description
```

在层被删除前，我们能收集一些信息。层被删除后就只能得到 Name 属性了。本例取得层的描述和名字。

ChangeAttribute 事件

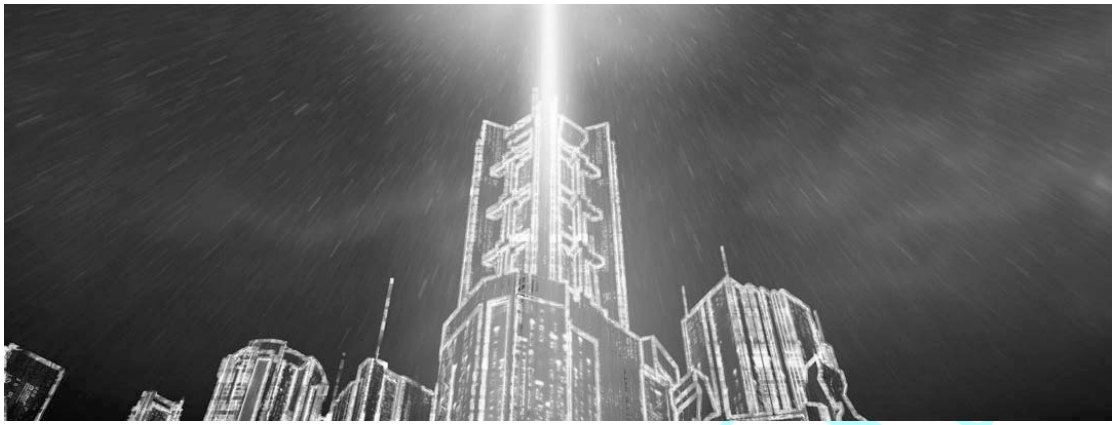
```
Case MsdLevelChangeType.msdLevelChangeChangeAttribute
    Debug.Print "Change Attribute: " & TheLevel.Name & vbCr & _
                vbTab & TheLevel.Description & vbCr & _
                vbTab & TheLevel.IsActive & vbCr & _
                vbTab & TheLevel.ElementColor
```

层有非常多的属性，本例仅查看了其中的四个属性。

本章回顾

每个模型都使用层，因为层对组织设计文件和模型来说至关重要。知道层何时被修改很有用，特别是在处理标准时更是如此。

China BDN



26. 改变跟踪事件

我们在本书中要考察的最后一个公开 MicroStation 事件的接口就是“改变跟踪事件(Change Track Events)”接口，它公开了四个事件：BeginUndoRedo、ElementChanged、FinishUndoRedo 和 Mark。我们将考察其中的两个。

本章包括：

- BeginUndoRedo 事件
- ElementChanged 事件

BeginUndoRedo 事件

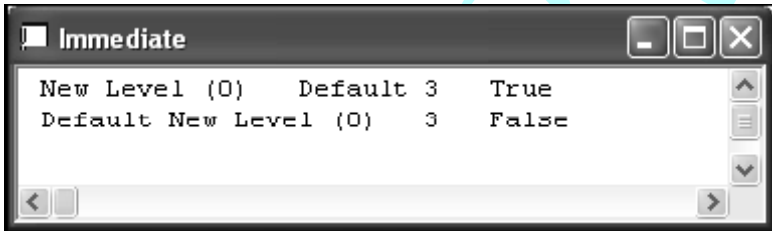
```
Private Sub IChangeTrackEvents_BeginUndoRedo( _
        ByVal AfterUndoRedo As Element, _
        ByVal BeforeUndoRedo As Element, _
        ByVal Action As MsdChangeTrackAction, _
        ByVal IsUndo As Boolean)
End Sub
```

BeginUndoRedo 事件在任何撤销(undo)或重做(redo)之前触发。AfterUndoRedo 参数指向撤销或重做动作之后的元素，BeforeUndoRedo 参数指向那些动作之前的元素，Action 参数表明发生了哪种动作，IsUndo 参数让我们知道动作是一个撤销或是重做。接下来让我们看一些该事件中的代码：

```
Private Sub IChangeTrackEvents_BeginUndoRedo( _
                ByVal AfterUndoRedo As Element, _
```

```
                                ByVal BeforeUndoRedo As Element, _
                                ByVal Action As MsdChangeTrackAction, _
                                ByVal IsUndo As Boolean)
        Debug.Print AfterUndoRedo.Level.Name & vbTab & _
            BeforeUndoRedo.Level.Name & vbTab & Action & vbTab & IsUndo
    End Sub
```

这个例子将被修改元素的层名、Action(动作)和 IsUndo 参数写入立即窗口。例如，我们改变了一个元素的层、发出撤销命令然后又发出重做命令，在立即窗口中看到如下行的内容：



从以上窗口中能看到改变前后的层名、动作类型(3)以及第一个动作是撤销第二个动作是重做(根据 True 和 False 来判定的)。层名和撤销/重做值的含义都很明确，但数字 3 是什么意思呢？它是事件的类型。对吗？该值指向 MsdChangeTrackAction 枚举类型中的一项。

msdChangeTrackActionAdd = 2

msdChangeTrackActionAppData = 8

msdChangeTrackActionDelete = 1

msdChangeTrackActionDrop = 6

msdChangeTrackActionMark = 7

msdChangeTrackActionModelAdd = 9

msdChangeTrackActionModelDelete = 10

msdChangeTrackActionModify = 3

msdChangeTrackActionModifyFence = 5

msdChangeTrackActionNewFilePositionAndModify = 4

我们看到撤销/重做动作对一个元素的修改。请记住，该枚举中的常数不仅用在 BeginUndoRedo 事件中，也用在其它改变跟踪事件接口的地方。

ElementChanged 事件

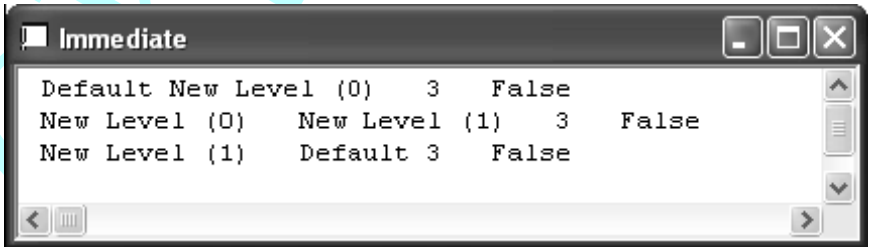
```
Private Sub IChangeTrackEvents_ElementChanged( _
                ByVal AfterChange As Element, _
                ByVal BeforeChange As Element, _
                ByVal Action As MsdChangeTrackAction, _
                CantBeUndone As Boolean)
End Sub
```

每当元素改变时就触发 ElementChanged 事件。该事件所提供的参数与 BeginUndoRedo 事件非常相似，那就是元素改变前后的引用、动作类型和该动作是否能被撤销。

让我们看几个 ElementChanged 事件的实现：

```
Private Sub IChangeTrackEvents_ElementChanged( _
                ByVal AfterChange As Element, _
                ByVal BeforeChange As Element, _
                ByVal Action As MsdChangeTrackAction, _
                CantBeUndone As Boolean)
    Debug.Print BeforeChange.Level.Name & vbTab & _
                AfterChange.Level.Name & vbTab & _
                Action & vbTab & CantBeUndone
End Sub
```

代码既简单又直接。当元素被修改时，给出修改前后的元素、动作类型以及该动作是否可被撤销。



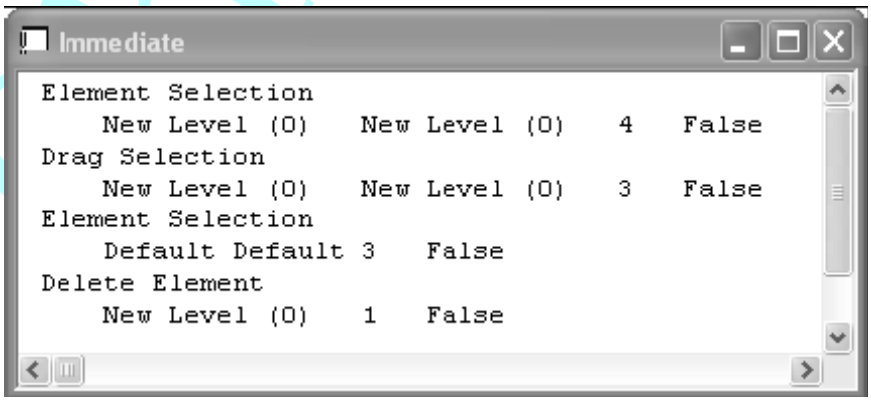
从上图能看出，一个元素的层由“Default”改变成了“New Level (0)”，又改变成了“New Level (1)”，再改回到“Default”。每个动作类型都是 3 且动作可被撤销，因为传递给我们的“CantBeUndone”参数是 False 值。

改变类型是 3。通过查看 MsdChangeTrackAction 枚举常数可知，我们正处理“msdChangeTrackActionModify”动作。

让我们给事件多添加一些代码：

```
Private Sub IChangeTrackEvents_ElementChanged( _
                                    ByVal AfterChange As Element, _
                                    ByVal BeforeChange As Element, _
                                    ByVal Action As MsdChangeTrackAction, _
                                    CantBeUndone As Boolean)
    Select Case Application.CommandState.CommandName
        Case "Delete Element"
            Debug.Print "Delete Element"
            Debug.Print vbTab & BeforeChange.Level.Name & vbTab & _
                        Action & vbTab & CantBeUndone
        Case Else
            Debug.Print Application.CommandState.CommandName
            Debug.Print vbTab & BeforeChange.Level.Name & vbTab & _
                        AfterChange.Level.Name & vbTab & _
                        Action & vbTab & CantBeUndone
    End Select
End Sub
```

查看一下 CommandState.CommandName 属性能告知我们就在元素改变前发生了什么。



从上图可看出我们遇到了“msdChangeTrackActionNewFilePosition_

AndModify ” (4) 、“ msdChangeTrackActionModify ” (3) 和“msdChangeTrackActionDelete”(1)，这三个动作是基于 CommandState 对象的 CommandName 属性得到的。在“Delete Element”命令名后仅显示了“BeforeChange”对象的层，因为元素被删除后“AfterChange”对象被设置为“Nothing”且对象本身也被删除。

把响应基于 CommandName 事件在某些情况下能正常工作，但把响应基于“动作”将产生更可靠的结果。

```
Private Sub IChangeTrackEvents_ElementChanged( _
                                ByVal AfterChange As Element, _
                                ByVal BeforeChange As Element, _
                                ByVal Action As MsdChangeTrackAction, _
                                CantBeUndone As Boolean)
    Select Case Action
        Case msdChangeTrackActionAdd
        Case msdChangeTrackActionAppData
        Case msdChangeTrackActionDelete
        Case msdChangeTrackActionDrop
        Case msdChangeTrackActionMark
        Case msdChangeTrackActionModelAdd
        Case msdChangeTrackActionModelDelete
        Case msdChangeTrackActionModify
        Case msdChangeTrackActionModifyFence
        Case msdChangeTrackActionNewFilePositionAndModify
    End Select
End Sub
```

我们现在就能搭建这一框架。请看几个例子。

例 1

```
Private Sub IChangeTrackEvents_ElementChanged( _
                                ByVal AfterChange As Element, _
```

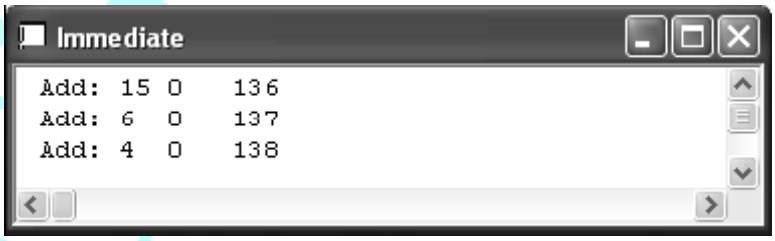
```
                                ByVal BeforeChange As Element, _
                                ByVal Action As MsdChangeTrackAction, _
                                CantBeUndone As Boolean)
    Select Case Action
        Case msdChangeTrackActionAdd
        Case msdChangeTrackActionAppData
        Case msdChangeTrackActionDelete
        Case msdChangeTrackActionDrop
        Case msdChangeTrackActionMark
        Case msdChangeTrackActionModelAdd
        Case msdChangeTrackActionModelDelete
        Case msdChangeTrackActionModify
            Select Case AfterChange.Type
                Case MsdElementType.msdElementTypeTextNode, _
                     MsdElementType.msdElementTypeText
                    If AfterChange.Level.Name <> "TEXT" Then
                        AfterChange.Level = _
                                ActiveDesignFile.Levels("TEXT")
                        AfterChange.Rewrite
                    End If
            End Select
        Case msdChangeTrackActionModifyFence
        Case msdChangeTrackActionNewFilePositionAndModify
    End Select
End Sub
```

当一个元素被修改时，我们检查它是否为文本节点或文本元素。如果是这两种类型之一，再进一步检查其所在的层。如果它不在“TEXT”层上，就将其放置到“TEXT”层上并重写之，这样改变才能保存到文件中。

China BDN

例 2

```
Private Sub IChangeTrackEvents_ElementChanged( _
                                ByVal AfterChange As Element, _
                                ByVal BeforeChange As Element, _
                                ByVal Action As MsdChangeTrackAction, _
                                CantBeUndone As Boolean)
    Select Case Action
        Case msdChangeTrackActionAdd
            Debug.Print "Add: " & AfterChange.Type & vbTab & _
                    AfterChange.ID.High & vbTab & AfterChange.ID.Low
        Case msdChangeTrackActionAppData
        Case msdChangeTrackActionDelete
        Case msdChangeTrackActionDrop
        Case msdChangeTrackActionMark
        Case msdChangeTrackActionModelAdd
        Case msdChangeTrackActionModelDelete
        Case msdChangeTrackActionModify
        Case msdChangeTrackActionModifyFence
        Case msdChangeTrackActionNewFilePositionAndModify
    End Select
End Sub
```



知道添加了什么元素类型很有用。再添加一个函数用于将元素类型数字转换为描述。我们将用这个新的函数修改事件。

```
Private Sub IChangeTrackEvents_ElementChanged( _
                                ByVal AfterChange As Element, _
```

```
                                ByVal BeforeChange As Element, _
                                ByVal Action As MsdChangeTrackAction, _
                                CantBeUndone As Boolean)
    Select Case Action
        Case msdChangeTrackActionAdd
            Debug.Print "Add: " & GetType(AfterChange.Type) & vbTab & _
                    AfterChange.ID.High & vbTab & AfterChange.ID.Low
        Case msdChangeTrackActionAppData
        Case msdChangeTrackActionDelete
        Case msdChangeTrackActionDrop
        Case msdChangeTrackActionMark
        Case msdChangeTrackActionModelAdd
        Case msdChangeTrackActionModelDelete
        Case msdChangeTrackActionModify
        Case msdChangeTrackActionModifyFence
        Case msdChangeTrackActionNewFilePositionAndModify
    End Select
End Sub
Function GetType(TypeIn As MsdElementType) As String
    Select Case TypeIn
        Case msdElementTypeArc
            GetType = "Arc"
        Case msdElementTypeBsplineBoundary
            GetType = "BSplineBoundary"
        Case msdElementTypeBsplineCurve
            GetType = "BSplineCurve"
        Case msdElementTypeBsplineKnot
            GetType = "BSplineKnot"
        Case msdElementTypeBsplinePole
            GetType = "BSplinePole"
```


China BDN

```
Case msdElementTypeBsplineSurface
    GetType = "BSplineSurface"
Case msdElementTypeBsplineWeight
    GetType = "BSplineWeight"
Case msdElementTypeCellHeader
    GetType = "CellHeader"
Case msdElementTypeCellLibraryHeader
    GetType = "CellLibraryHeader"
Case msdElementTypeComplexShape
    GetType = "ComplexShape"
Case msdElementTypeComplexString
    GetType = "ComplexString"
Case msdElementTypeCone
    GetType = "Cone"
Case msdElementTypeConic
    GetType = "Conic"
Case msdElementTypeCurve
    GetType = "Curve"
Case msdElementTypeDesignFileHeader
    GetType = "DesignFileHeader"
Case msdElementTypeDgnStoreComponent
    GetType = "DgnStoreComponent"
Case msdElementTypeDgnStoreHeader
    GetType = "DgnStoreHeader"
Case msdElementTypeDigSetData
    GetType = "DigSetData"
Case msdElementTypeDimension
    GetType = "Dimension"
Case msdElementTypeEllipse
    GetType = "Ellipse"
```

China BDN

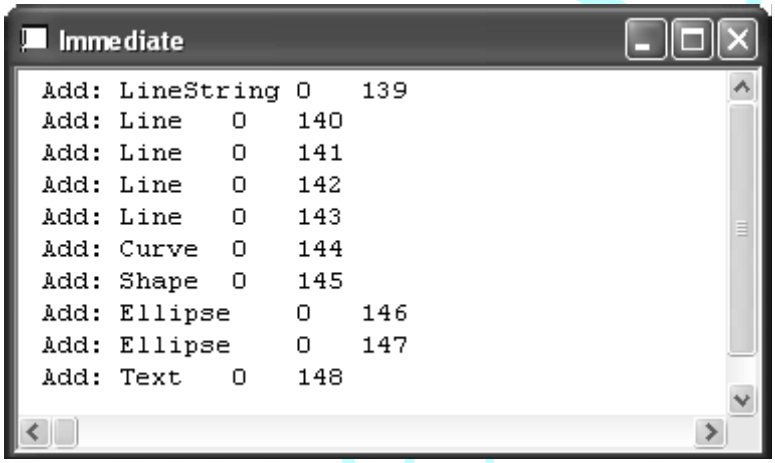
```
Case msdElementTypeGroupData
    GetType = "GroupData"
Case msdElementTypeLevelMask
    GetType = "LevelMask"
Case msdElementTypeLevelSymbology
    GetType = "LevelSymbology"
Case msdElementTypeLine
    GetType = "Line"
Case msdElementTypeLineString
    GetType = "LineString"
Case msdElementTypeMatrixDoubleData
    GetType = "MatrixDoubleData"
Case msdElementTypeMatrixHeader
    GetType = "MatrixHeader"
Case msdElementTypeMatrixIntegerData
    GetType = "MatrixIntegerData"
Case msdElementTypeMeshHeader
    GetType = "MeshHeader"
Case msdElementTypeMicroStation
    GetType = "MicroStation"
Case msdElementTypeMultiLine
    GetType = "MultiLine"
Case msdElementTypeNamedGroupComponent
    GetType = "NamedGroupComponent"
Case msdElementTypeNamedGroupHeader
    GetType = "NamedGroupHeader"
Case msdElementTypePointString
    GetType = "PointString"
Case msdElementTypeRasterComponent
    GetType = "RasterComponent"
```


Qing BDN

```
Case msdElementTypeRasterFrame
    GetType = "RasterFrame"
Case msdElementTypeRasterHeader
    GetType = "RasterHeader"
Case msdElementTypeRasterReference
    GetType = "RasterReference"
Case msdElementTypeRasterReferenceComponent
    GetType = "RasterReferenceComponent"
Case msdElementTypeReferenceAttachment
    GetType = "ReferenceAttachment"
Case msdElementTypeReferenceOverride
    GetType = "ReferenceOverride"
Case msdElementTypeShape
    GetType = "Shape"
Case msdElementTypeSharedCell
    GetType = "SharedCell"
Case msdElementTypeSharedCellDefinition
    GetType = "SharedCellDefinition"
Case msdElementTypeSolid
    GetType = "Solid"
Case msdElementTypeSurface
    GetType = "Surface"
Case msdElementTypeTable
    GetType = "Table"
Case msdElementTypeTableEntry
    GetType = "TableEntry"
Case msdElementTypeTag
    GetType = "Tag"
Case msdElementTypeText
    GetType = "Text"
```

China BDN

```
        Case msdElementTypeTextNode
            GetType = "TextNode"
        Case msdElementTypeView
            GetType = "View"
        Case msdElementTypeViewGroup
            GetType = "ViewGroup"
    End Select
End Function
```

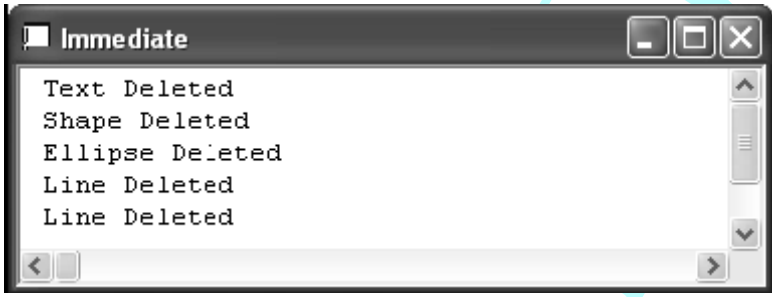


例 3

当删除一个元素时，记录的动作是 msdChangeTrackActionDelete。

```
Private Sub IChangeTrackEvents_ElementChanged( _
                        ByVal AfterChange As Element, _
                        ByVal BeforeChange As Element, _
                        ByVal Action As MsdChangeTrackAction, _
                        CantBeUndone As Boolean)
    Select Case Action
        Case msdChangeTrackActionAdd
        Case msdChangeTrackActionAppData
        Case msdChangeTrackActionDelete
            Debug.Print GetType(BeforeChange.Type) & " Deleted"
        Case msdChangeTrackActionDrop
        Case msdChangeTrackActionMark
```

```
        Case msdChangeTrackActionModelAdd
        Case msdChangeTrackActionModelDelete
        Case msdChangeTrackActionModify
        Case msdChangeTrackActionModifyFence
        Case msdChangeTrackActionNewFilePositionAndModify
    End Select
End Sub
```



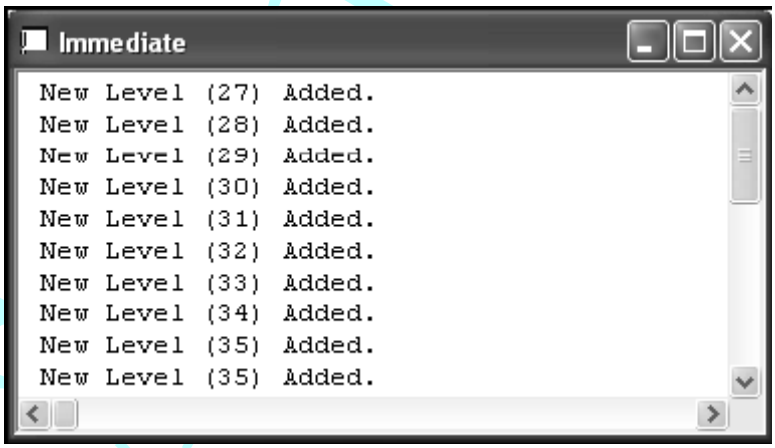
元素被删除时，我们只存取“BeforeChange”元素。

例 4

已经有几个例子用“Action(动作)”参数代替对“CommandName”的依靠。下面的例子使用了这两种技术。

```
Private Sub IChangeTrackEvents_ElementChanged( _
                        ByVal AfterChange As Element, _
                        ByVal BeforeChange As Element, _
                        ByVal Action As MsdChangeTrackAction, _
                        CantBeUndone As Boolean)
    Dim myLevel As Level
    Dim LevelCount As Long
    Select Case Action
        Case msdChangeTrackActionAdd
        Case msdChangeTrackActionAppData
        Case msdChangeTrackActionDelete
        Case msdChangeTrackActionDrop
        Case msdChangeTrackActionMark
```

```
            Case msdChangeTrackActionModelAdd
            Case msdChangeTrackActionModelDelete
            Case msdChangeTrackActionModify
            Case msdChangeTrackActionModifyFence
            Case msdChangeTrackActionNewFilePositionAndModify
                Select Case GetType(AfterChange.Type)
                    Case "Table"
                        Select Case CommandState.CommandName
                            Case "New Level"
                                LevelCount = ActiveDesignFile.Levels.Count
                                Set myLevel = _
                                    ActiveDesignFile.Levels(LevelCount - 1)
                                Debug.Print myLevel.Name & " Added."
                        End Select
                End Select
        End Select
End Sub
```



NewFilePositionAndModify 改变类型跟踪设计文件和非元素对象的修改，如对层的修改。本例中除了 Action 参数外还使用了 CommandState 对象的 CommandName 属性，这样我们能知道层何时添加到文件中或从文件中删除。关于层修改的更多信息，请回顾一下 ILevelChangeEvents 接口。

激活 ChangeTrackEvents 接口

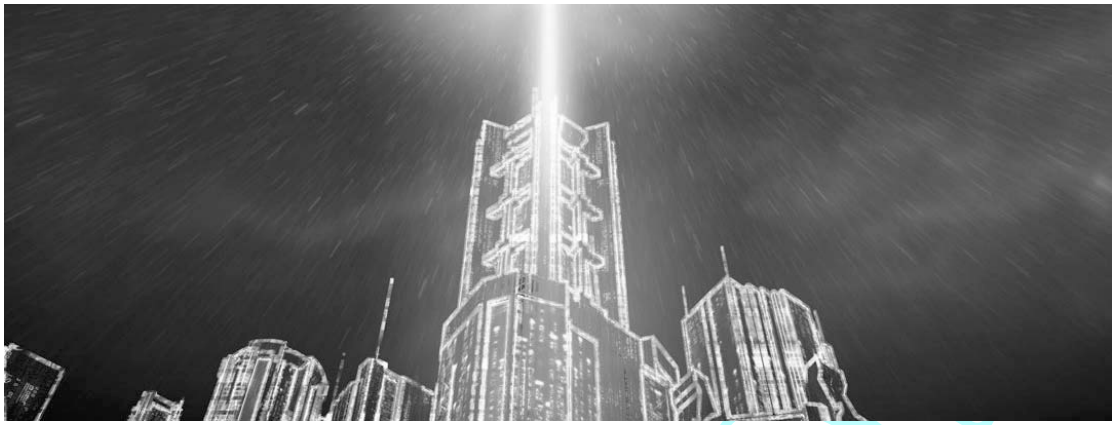
与其它接口一样，我们在类模块中实现每个接口事件然后添加事件处理器。在一个代码模块中运行添加事件处理器的代码如下：

```
Private myCTE As clsChangeTrackEvents
Sub AddEvents()
    RemoveEvents
    Set myCTE = New clsChangeTrackEvents
    AddChangeTrackEventsHandler myCTE
End Sub
Sub RemoveEvents()
    If myCTE Is Nothing = False Then
        RemoveChangeTrackEventsHandler myCTE
        Set myCTE = Nothing
    End If
End Sub
```

本章回顾

ChangeTrackEvents 接口能用于非常多的目的。很幸运，通常经过很少的试验就能实现该接口并得到您需要的结果。

China BDN



27. 非图形信息—数据库

我们能把有关设计文件元素的数据保存在数据库中而不是设计文件中。这样做使设计文件更小，且在 MicroStation 环境外也能取得元素信息。

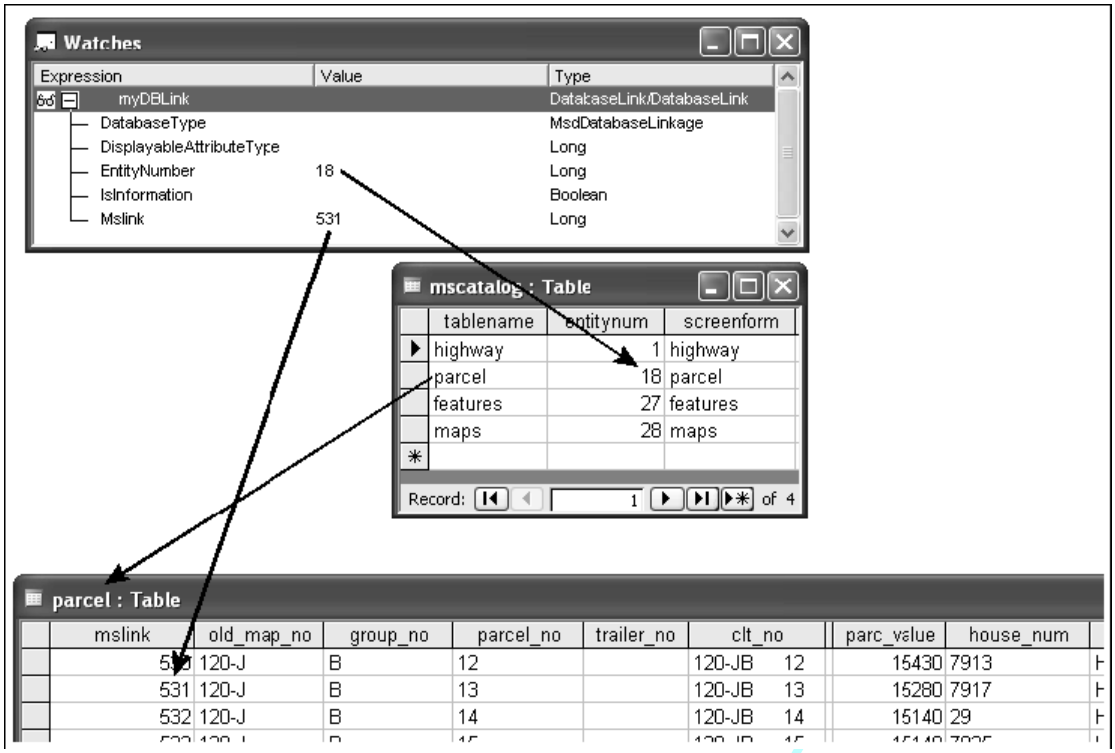
本章集中于 MicroStation 数据库链接的功能。后续的章节讨论使用数据库的另外途径。

本章包括：

- 链接MicroStation元素到数据库
- 从头建立一个链接到设计文件的Microsoft Access数据库
- 使用UDL(通用数据链接)文件
- 用VBA将MicroStation中的元素链接到数据库
- 用SQL建立数据库记录
- 建立一个查看数据库信息的用户界面

MicroStation 是如何将元素“链接”到数据库的

通过把“DatabaseLink”对象附加到元素上，MicroStation 能让我们链接元素到数据库。数据库包含表，每个表包含行(记录)，每行又包含列。DatabaseLink 对象含有 MicroStation 所需的信息，利用这些信息能找到链接到元素的表和行。存储这些信息的 DatabaseLink 属性是“EntityNumber(实体号)”和“Mslink”。



这张图的最顶部是一个 DatabaseLink 对象，该对象附加到了设计文件中的一个元素上。让我们考察一下 DatabaseLink 的 EntityNumber 属性。

DatabaseLink 的 EntityNumber 用于确定 DatabaseLink 指向的表。EntityNumber 的值是数字型的，所以它不能直接表示要查找的表，而仅仅是告诉我们如何利用它来查找到表。DatabaseLink 所使用的每个数据库都有一个“mscatalog”表，该表把 DatabaseLink. EntityNumber 属性对应到所需的数据库表上。当在 mscatalog 表中找到一个合适的 entitynum 值时，其对应的 tablename 字段就是我们所要的表名。

知道了表以后，就可以在表中查找匹配的 mslink 值。

在上面的例子中，我们发现了一个附加到 MicroStation 元素上的 DatabaseLink 对象，DatabaseLink 对象的 EntityNumber 是 18。在 mscatalog 表中，实体号 18 指向叫做 parcel 的表。在 parcel 表的 mslink 字段中能找到一个值为 531，它提供了 MicroStation 元素与数据库记录(行)间的匹配。

这就是 MicroStation 链接元素到数据库记录的工作原理。如果多个记录链接到一个元素，则多个 DatabaseLink 对象附加到了 MicroStation 的这个元素上。

我们讨论了链接 MicroStation 元素到数据库记录背后的理论。现在看个例子。

1、打开文件…\Examples\Civil\Dgn\cogo.dgn。该文件随 MicroStation 一起安装；

2、放大文件的左上角；



在文件和上图中能看到 Hayden 大道的 13 号区。

3、在 Microsoft Access 中打开数据库文件…\Examples\Database\Oledb\Examples\Access\gis.mdb，这是连接到该文件的数据库。查看 mscatalog 表能看到 MicroStation 可使用的 Access 数据库中的数据库表。“parcel”表显示了我们刚刚看过的记录。

我们已经看过了 MicroStation 是如何将设计文件中的元素链接到数据库中的记录。通过在 MicroStation 帮助文件中查找“Database”能获得更多信息。有许多与使用 MicroStation 数据库工具相关的主题，但本章将集中讨论 VBA 编程工具。

从头开始建立数据库

MicroStation 支持 Oracle、ODBC、OLEDB 和 SQL Server(通过 BUDBC 数据库服务器选择)。我们用 OLEDB 连接来操作 Microsoft Access 数据库。OLEDB 驱动程序作为 Microsoft Windows 操作系统的一部分被安装，我们利用

这些驱动程序与数据库通讯并建立新的数据库。

让我们建立一个新的 Microsoft Access 数据库。由于本书是讲编程的，所以我将演示如何建立一个新数据库，即使您的计算机上没有安装 Microsoft Access 也没有关系。先是代码后是解释。

```
Sub CreateDB()
    Dim myCat As New ADOX.Catalog
    Dim dbLocation As String
    Dim ConnString As String
    dbLocation = "C:\MicroStation VBA\DatabaseLinkTest.mdb"
    ConnString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & _
                                    dbLocation
    myCat.Create ConnString
End Sub
```

这段代码使用了“Microsoft ADO Ext. 2.8 for DDL and Security”引用，执行该过程前需要在 VBA 中添加这个引用。这段代码建立一个新的数据库，数据库是空的，其中没有表。下一步是建立表，虽然有许多向现有数据库添加表的方式，但我们仅讨论其中的三种。

向现有数据库添加表的一种方式是使用我们建库时刚用过的同一个 ADOX 引用。

```
Sub CreateDB2()
    Dim myCat As New ADOX.Catalog
    Dim dbLocation As String
    Dim ConnString As String
    dbLocation = "C:\MicroStation VBA\DatabaseLinkTest.mdb"
    ConnString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & _
                    dbLocation
    myCat.ActiveConnection = ConnString
    Dim myTable As New Table
    Dim myColumn As ADOX.Column
    myTable.Name = "Lots"
```

```
'mslink Column
myTable.Columns.Append "mslink", adInteger

'Owner Column
Set myColumn = New ADOX.Column
myColumn.Name = "Owner"
myColumn.Type = adVarWChar
myColumn.Attributes = adColNullable
myColumn.DefinedSize = 50
myTable.Columns.Append myColumn

'Sold Column
Set myColumn = New ADOX.Column
myColumn.Name = "Sold"
myColumn.Type = adBoolean
myTable.Columns.Append myColumn

'DateSold Column
Set myColumn = New ADOX.Column
myColumn.Name = "DateSold"
myColumn.Type = adDate
myColumn.Attributes = adColNullable
myTable.Columns.Append myColumn

'Acres Column
Set myColumn = New ADOX.Column
myColumn.Name = "Acres"
myColumn.Type = adDouble
myColumn.Attributes = adColNullable
myTable.Columns.Append myColumn
```

China BDN

```
        'SaleAmount Column
        Set myColumn = New ADOX.Column
        myColumn.Name = "SaleAmount"
        myColumn.Type = adCurrency
        myColumn.Attributes = adColNullable
        myTable.Columns.Append myColumn
    myCat.Tables.Append myTable
    myCat.ActiveConnection = Nothing
End Sub
```

建立“mslink”列时使用了内嵌方式(in-line)附加列集合，以内嵌方式建立列时不能对列(字段)属性进行更多的控制。例如，您可能想让一个字段(列)为必填或者不想这样，内嵌方式做不到。在非内嵌方式下，如果不想让列给必填，可指定字段的 Attributes 属性为“adColNullable”，意思是该列可以是“null”值。

建立表的第二种方式是用“ActiveX Data Objects Library”引用。在运行 CreateDB3 前需要添加一个新的引用。当打开引用对话框来添加“ActiveX Data Objects”时可能会看到有几个可选的库，您应当选择最高版本的那个。

下面的例子通过执行一个 SQL 语句来建立新表“Expenses”。代码如下：

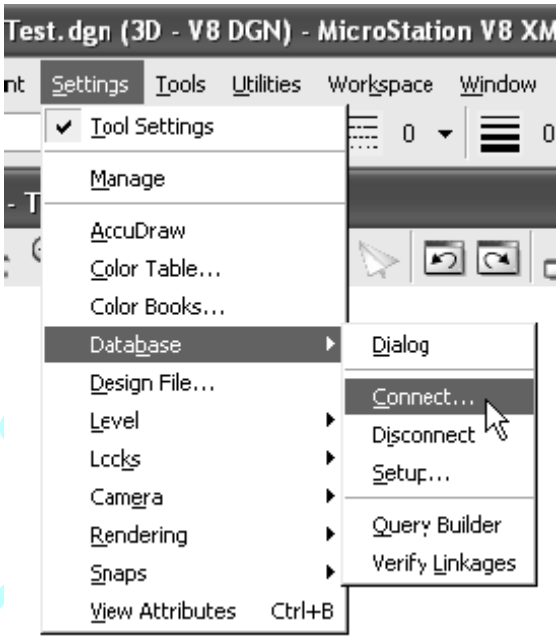
```
Sub CreateDB3()
    Dim myDB As New ADODB.Connection
    Dim dbLocation As String
    Dim ConnString As String
    dbLocation = "C:\MicroStation VBA\DatabaseLinkTest.mdb"
    ConnString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & _
                 dbLocation
    myDB.Open ConnString
    myDB.Execute "Create Table Expenses (mslink Counter, " & _
                 "LotLink Long, Description Char(100), Amount Double)"
End Sub
```

建立 mslink 字段时给了一个“Counter”类型，这能使该字段成为自动编号的字段，即每当建立一个新记录时自动给该字段赋一个数字值。该字段从一(1)开始，每当建立一条新记录时该值增一。自动编号字段类型并非在所有数据库中都可用但在 Microsoft Access 数据库中可用。

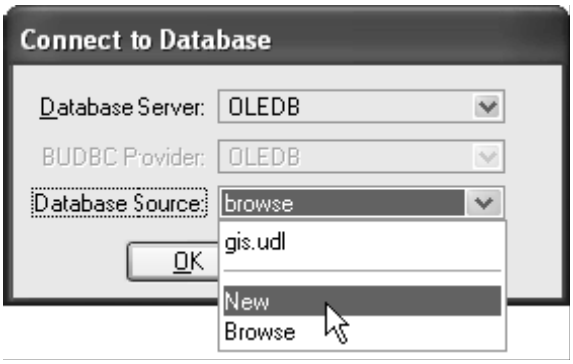
最后一种方式使用 MicroStation 可视化 SQL 查询建造器(Query Builder)。查询建造器从连接到 MicroStation 的数据库中取得信息，并在该连接的数据库上执行标准 SQL 语句。

在使用查询建造器前请连接数据库，而在 MicroStation 能连接一个数据库前需要知道数据库的类型和数据库的位置。提供这一信息的最佳方式是通过一个 UDL 文件。

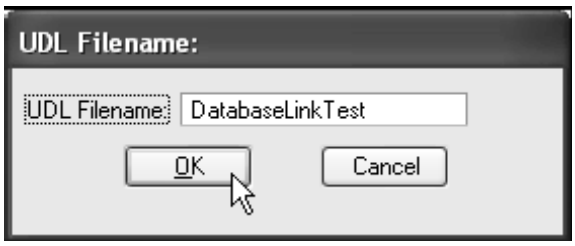
在 MicroStation 中选菜单 **Settings > Database > Connect(设置 > 数据库 > 连接)**。



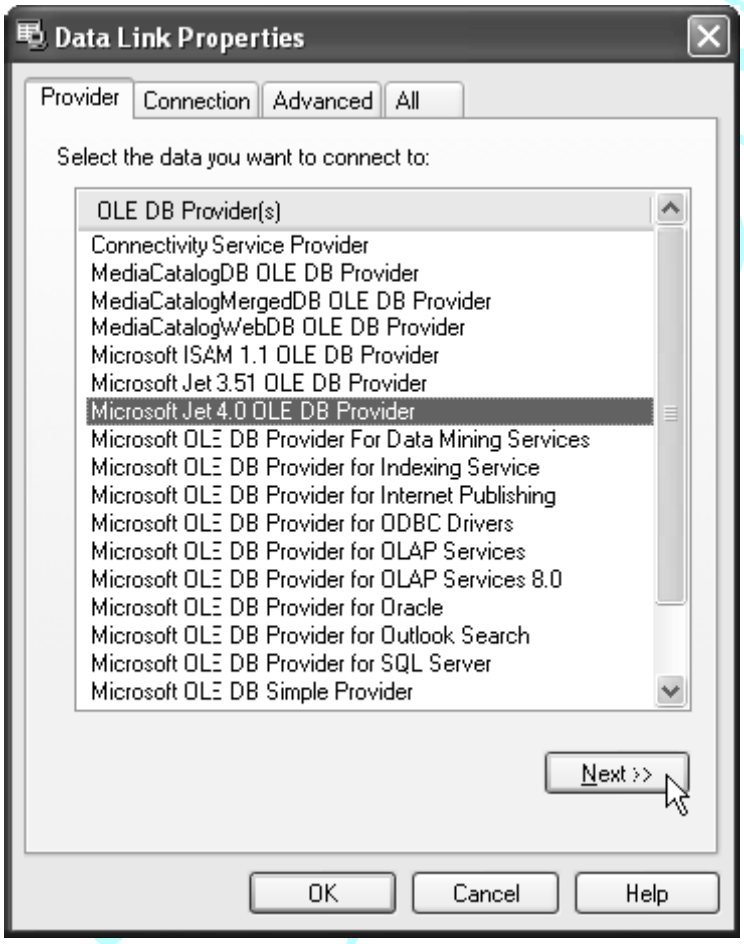
数据库连接对话框被打开：



从 Database Server(数据库服务器)中选择 OLEDB 后，再从 Database Source(数据库源)复合框中选择 New(新建)。



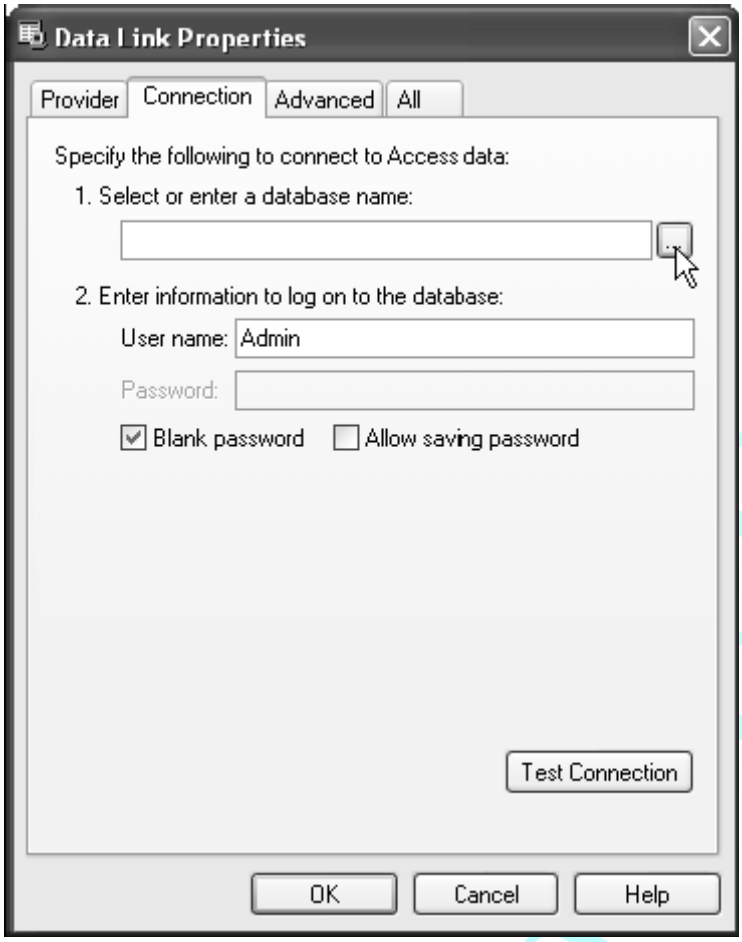
选择 New 以后，就建立了一个新的 UDL 文件，随后是输入 UDL 文件的参数。



需要向 MicroStation 提供的两个关键信息是什么？首先是数据库的类型，通过选择 OLE DB 提供程序(数据库驱动程序)来指定数据库类型。当使用 Microsoft Access 数据库时，从列表中选择“Microsoft Jet 4.0 OLE DB Provider”。点击 Next(下一步)继续。

Connection(连接)选项卡让我们提供有关要连接的数据库位置信息。Connection 选项卡的界面随 Provider(提供程序)选项卡的变化而不同。

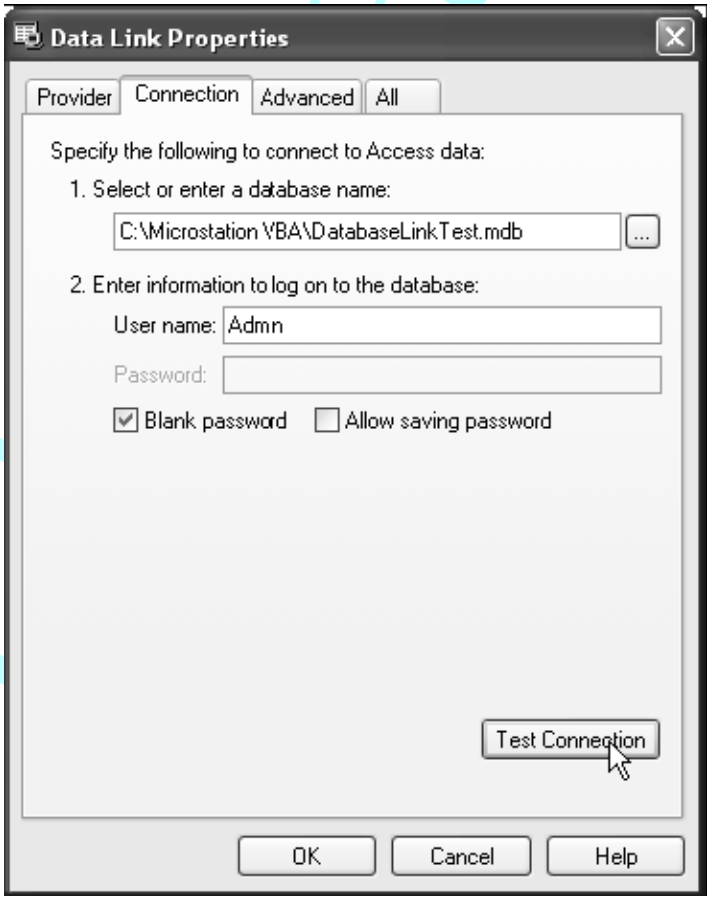
当使用 Jet(Microsoft Access)数据库时，可以键入文件名或点击如图所示的按钮来浏览文件。如果使用的是 SQL Server 数据库，将提示用户选择 SQL 服务器名(它可以是网络上任意的安装有 SQL Server 的服务器)然后再选择要连接的数据库。

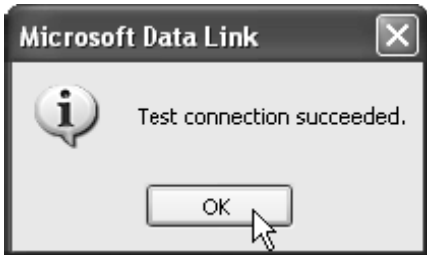


如果数据库需要，还可以输入用户名和密码参数。

点击 Browse(浏览)按钮后可以选择刚建立的 Access 数据库。选择完文件后，文件路径显示在 UDL 属性对话框中。

用 TestConnection(测试连接)按钮来查看是否提供了含有足够信息的 UDL 文件，这些信息当然是指连接所选择的数据库的信息。

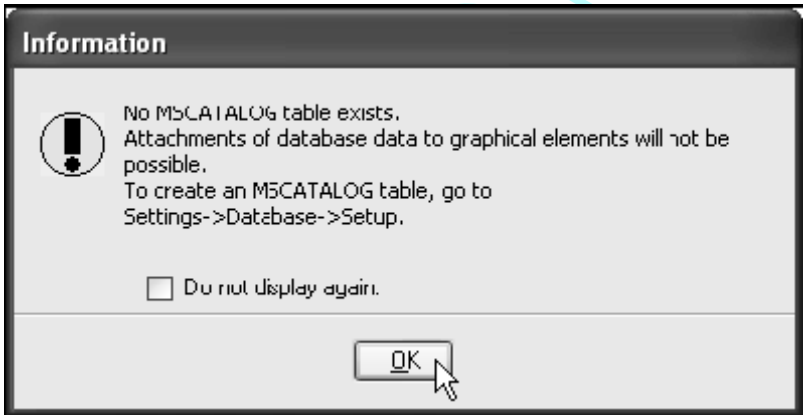




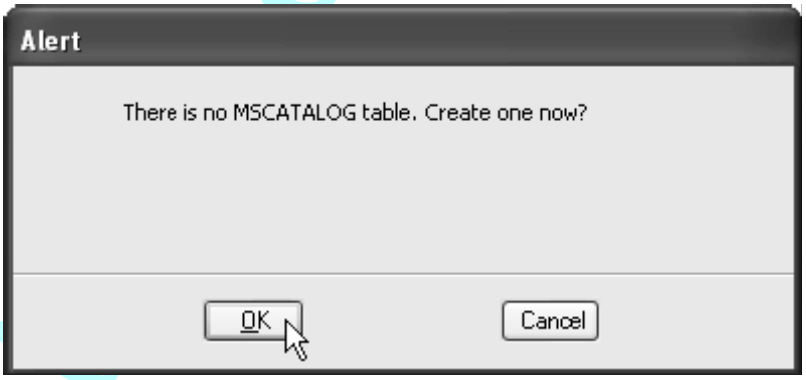
如果提供的信息充足，将得到一个“Test connection succeeded(测试连接成功)”的消息。这是我们期望看到的消息框，然而，有时会看到不同的消息框。

例如，如果对选择的数据库指定错了提供程序，将看到一个错误消息框。或者，如果您选择的数据库在别的地方被独占式打开，也将看到一个错误。

在测试连接消息框中点击 OK 后再在 UDL 对话框点击 OK。如果一切顺利，将看到右图所示消息框：



MicroStation 依赖一个叫做“MSCatalog”的表把 DatabaseLink 对象与数据库记录关联。因为我们还没有在刚连接到 MicroStation 的数据库中建立 MSCatalog 表，所以 MicroStation 通知我们需要建立这个表并给出了如何做的提示。



让我们按消息框中的指导来做吧，选择菜单 **Settings > Database > Setup (设置 > 数据库 > 设置)**将看到这个消息框：

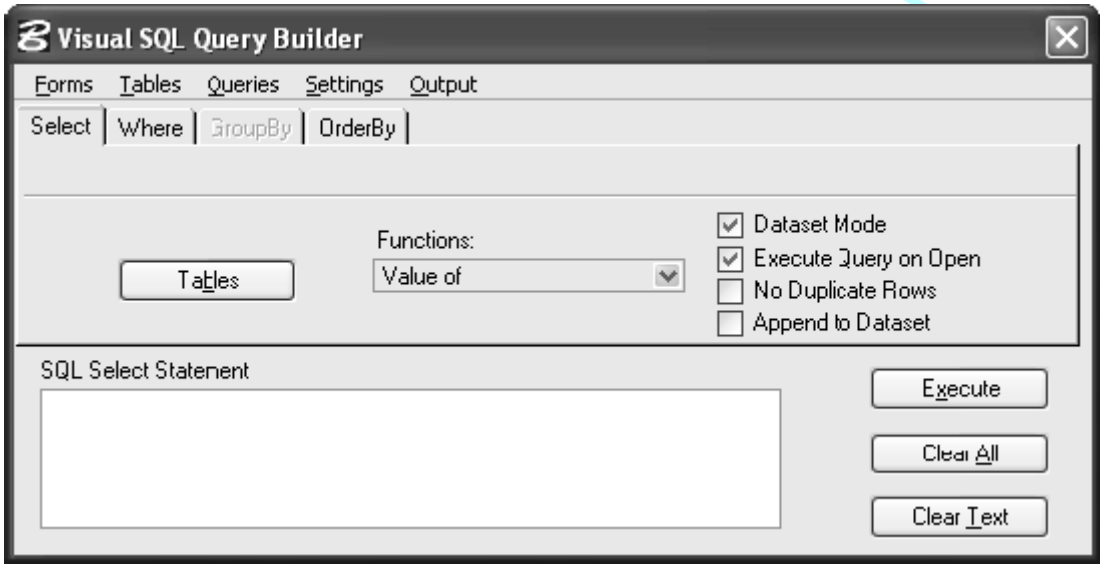
点击 OK 让 MicroStation 建立 MSCatalog 表。

接下来看到的是数据库设置对话框，我们稍后处理它。现在点击 OK 按钮关闭这个对话框。

MicroStation 建立 MSCatalog 表时可能需要卸载然后重新连接数据库，所以您可能再次看到 UDL 文件属性。在关闭 UDL 文件属性对话框前确保正确设置

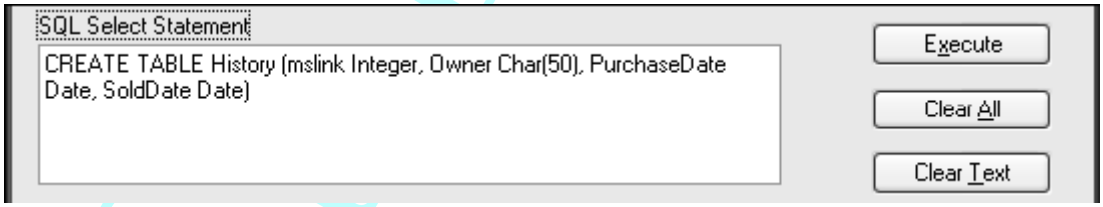
了“Provider(提供程序)”和“Connection(连接)”。

好。我们刚刚执行了一系列步骤把 DatabaseLinkTest.mdb 数据库连接上了 MicroStation。既然连接了数据库就能用 SQL 建造器(**Settings > Database > Query Builder**)来建立新表。



用可视化 SQL 查询建造器对话框能从连接的数据库中获取信息。从数据库检索信息经常用“Select”SQL 语句，所以键入框的标头是“SQL Select Statement(SQL Select 语句)”。但在该文本框中除了简单地选择数据库记录外还能做许多事情。

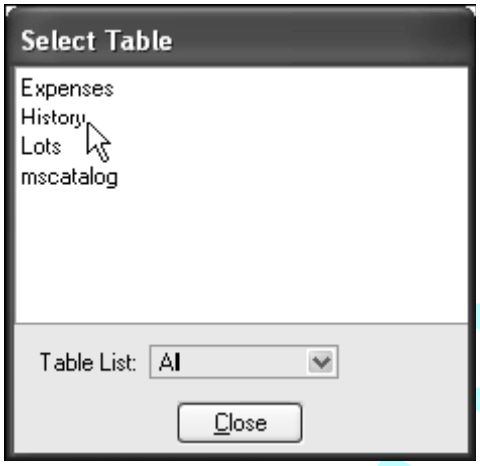
在“SQL Select Statement”文本框中键入 SQL 语句并点击 Execute(执行)按钮后，所输入的语句将被连接的数据库执行。让我们通过执行 SQL 语句来建立一个新表“History”。这个 History 表将保存过去的所有者的数据。



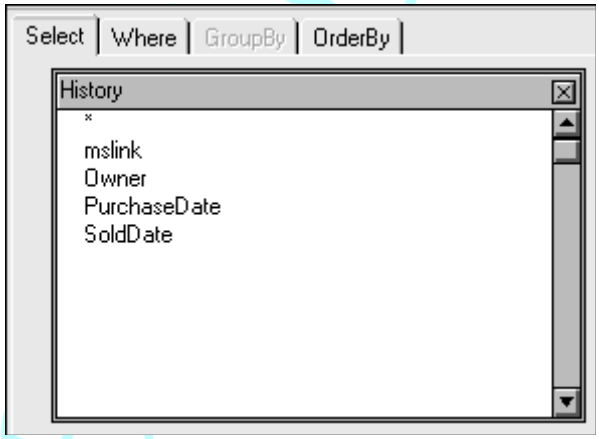
CREATE TABLE History (mslink Integer, Owner Char(50), PurchaseDate Date, SoldDate Date)

以上是建立新 History 表的语句。点击 Execute 按钮后新表被建立(如果 SQL 语句正确的话)。如何能知道它被建立了呢？请点击 Tables(表)按钮。

如果 History 表未能建立，它将不出现在“Select Table(选择表)”列表中，我们就知道所用的 SQL 语句需要进行适当地修改。



要在查询建造器对话框中显示 History 表，可在列表中的“History”上双击然后点击 Close(关闭)按钮。



这些就是 History 表中建立的字段。

使用 UDL 文件

迄今为止，我们讨论了 MicroStation 链接设计元素到数据库记录背后的工作机制，并且建立了新的 Access 数据库。在建立 Access 数据库时，我们建立了一个新的 UDL 文件，它有两个基本组件：提供程序和连接，分别指定数据库驱动程序和数据库位置。

UDL 文件最了不起的功能之一是，如果数据库位置发生改变，只需要对一个文件(UDL 文件)的一处作修改就能使所编写的程序以及建立的 MicroStation 链接正常工作。我们甚至能把数据库平台从 Microsoft Access 扩展为 SQL Server 或 Oracle，只需要对 UDL 文件作几处简单修改，我们的程序就能继续正常运行。

让我们再多看一眼以前运行过的一个过程：

```
Sub CreateDB3()
    Dim myDB As New ADODB.Connection
    Dim dbLocation As String
    Dim ConnString As String
```

```
    dbLocation = "C:\MicroStation VBA\DatabaseLinkTest.mdb"
    ConnString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & _
                 dbLocation
    myDB.Open ConnString
    myDB.Execute "Create Table Expenses (mslink Counter, " & _
                 "LotLink Long, Description Char(100), Amount Double)"
End Sub
```

在这个过程的代码中，使用了“Location”和“Connection String”来连接指定位置的数据库。这表明可以不用 UDL 文件也能操作数据库，但如果数据库的位置或文件名改变后将会怎样？

如果在程序中硬编码了数据库连接信息就必须修改代码。假如只有您一人使用程序，这还不是太大的问题，但如果多人使用您的程序，改变 UDL 文件还是比修改代码要容易得多，特别是当您锁定了 VBA 中的代码之后。您甚至能指导最初级的计算机用户来双击一个 UDL 文件并浏览不同的数据库。

在随后一章将更详细地讨论 UDL 文件的用法，现在只要记住使用 UDL 文件能提供强大和灵活的完美组合就够了。

链接 MicroStation 元素到数据库记录

链接 MicroStation 元素到数据库记录的过程很简单，这里是“CreateDatabaseLink”的声明：

```
Function CreateDatabaseLink(Mslink As Long, Entity As Long, _
        LinkType As MsdDatabaseLinkage, IsInformation As Boolean,
        DisplayableAttributeType As Long) As DatabaseLink
```

下面是使用 CreateDatabaseLink 的一个例子：

```
Sub DatabaseLinkA()
    Dim myElem As Element
    Dim myLink As DatabaseLink
    Set myElem = CommandState.GetLocatedElement(True)
    Set myLink = CreateDatabaseLink(1, 1, msdDatabaseLinkageOleDb, _
                 True, 0)
```

```
    myElem.AddDatabaseLink myLink
    myElem.Rewrite
End Sub
```

DatabaseLink 建立后，把它附加到 MicroStation 中的一个元素上。

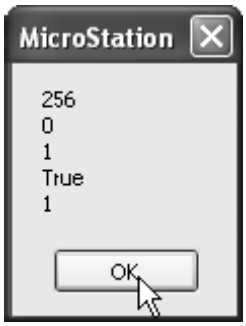
注意，可以在不连接数据库的情况下建立数据库链接并把它们附加到元素上。在上例中，我们建立了 MSLink 为 1、Entity(实体)也为 1 的数据库链接。

请记住，Entity 属性指向查找的表(通过在 MSCatalog 表中的 Entitynum 字段引用)，Mslink 属性指向表中的记录。

这段代码假设 MSCatalog 表中有一条 Entitynum 为 1 的记录，其所指向的表中有一条 MSLink 为 1 的记录。

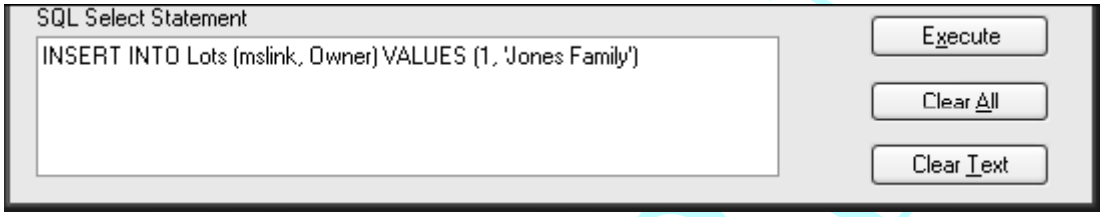
下面的代码查找 MicroStation 中选择的元素，在一个消息框中显示连接到其上的每个 DatabaseLink 的数据库链接属性。

```
Sub GetDatabaseLinksA()
    Dim myElem As Element
    Dim myLinks() As DatabaseLink
    Dim myLink As DatabaseLink
    Dim I As Long
    Set myElem = CommandState.GetLocatedElement(True)
    myLinks = myElem.GetDatabaseLinks
    For I = LBound(myLinks) To UBound(myLinks)
        Set myLink = myLinks(I)
        MsgBox myLink.DatabaseType & vbCr & _
            myLink.DisplayableAttributeType & vbCr & _
            myLink.EntityNumber & vbCr & _
            myLink.IsInformation & vbCr & _
            myLink.Mslink
    Next I
End Sub
```



用 SQL 建立数据库记录

到目前为止，我们成功地建立了新数据库、往新数据库中添加了新表，也知道了如何连接 DatabaseLink 到 MicroStation 中的元素。数据库中唯一缺少的是数据！我们需要知道如何向数据库中添加记录。可以像向数据库中添加表那样使用 SQL 查询建造器向表中添加记录。让我们向“Lots”表添加一条记录。



SQL 语句是：

INSERT INTO Lots (mslink, Owner) VALUES (1, 'Jones Family')

本例在“Lots”表中建立了一条新记录。Mslink 和 Owner 字段分别被赋以 1 和 Jones Family。可在适当位置放置它们来添加字段和值。关于 SQL 语句的更多信息，参见书后的“附加资源”。

建立一个查看数据库信息的用户界面



让我们利用目前所知的数据库链接来建立一个用户界面查看和编辑“Lot”信息。

这里是界面，文本框和复选框从“Lots”表的字段中显示相关的信息。

下面是界面背后的代码：

```
Const UDLFile As String = "C:\Documents and Settings\All Users\" & _
                          "Application Data\Documents\Bentley\WorkSpace\" & _
                          "Projects\Examples\Database\Oledb\Udl\" & _
                           "DatabaseLinkTest.udl"
```

```
Dim MSLinkID As Long
Private Sub btnClose_Click()
    Unload Me
End Sub
Private Sub btnUpdate_Click()
    Dim myDB As New ADODB.Connection
    Dim myRS As New ADODB.Recordset
    myDB.Open "File name=" & UDLFile
    myRS.Open "Select * from Lots Where MSLink = " & MSLinkID, _
                                myDB, adOpenForwardOnly, adLockOptimistic
    If myRS.EOF = False Then
        If txtOwner.Text <> "" Then
            myRS("Owner") = txtOwner.Text
        Else
            myRS("Owner") = Null
        End If
        myRS("Sold") = chkSold.Value
        If IsDate(txtDateSold.Text) Then
            myRS("DateSold") = CDate(txtDateSold.Text)
        Else
            myRS("DateSold") = Null
        End If
        If IsNumeric(txtAcres.Text) Then
            myRS("Acres") = CDbl(txtAcres.Text)
        Else
            myRS("Acres") = Null
        End If
        If IsNumeric(txtSaleAmount.Text) Then
            myRS("SaleAmount") = CDbl(txtSaleAmount.Text)
        Else
```

```
                myRS("SaleAmount") = Null
            End If
            myRS.Update
        End If
End Sub
Private Sub UserForm_Initialize()
        Dim myElem As Element
        Dim myLinks() As DatabaseLink
        Dim myLink As DatabaseLink
        Dim I As Long
        Dim myDB As New ADODB.Connection
        Dim myRS As New ADODB.Recordset
        myDB.Open "File name=" & UDLFile
        Set myElem = CommandState.GetLocatedElement(True)
        myLinks = myElem.GetDatabaseLinks
        For I = LBound(myLinks) To UBound(myLinks)
            Set myLink = myLinks(I)
            If myLink.EntityNumber = 1 Then
                myRS.Open "Select * from Lots Where mslink = " & _
                        myLink.Mslink, _
                        myDB, adOpenForwardOnly, adLockReadOnly
                If myRS.EOF = False Then
                    MSLinkID = myLink.Mslink
                    If IsNull(myRS("Owner")) Then
                        txtOwner.Text = ""
                    Else
                        txtOwner.Text = myRS("Owner")
                    End If
                    chkSold.Value = myRS("Sold")
                    If IsNull(myRS("DateSold")) Then
```


China BDN

```
                txtDateSold.Text = ""
            Else
                txtDateSold.Text = myRS("DateSold")
            End If
            If IsNull(myRS("Acres")) Then
                txtAcres.Text = ""
            Else
                txtAcres.Text = myRS("Acres")
            End If
            If IsNull(myRS("SaleAmount")) Then
                txtSaleAmount.Text = ""
            Else
                txtSaleAmount.Text = myRS("SaleAmount")
            End If
            myRS.Close
            myDB.Close
            Exit Sub
        End If
        myRS.Close
    End If
Next I
myDB.Close
End Sub
```

当窗体初始化时，我们在 MicroStation 中查找选择的元素，然后通过 DatabaseLink 对象取得“Lot”信息。然后在对话框上显示数据库中的信息并让用户改变文本框和复选框中的值。当用户点击 Update Database(更新数据库)按钮时，数据库记录基于在窗体中输入的值而更新。

本章回顾

数据库在表中存储大量的信息。数据库表含有字段(或列)，表中的每个记录

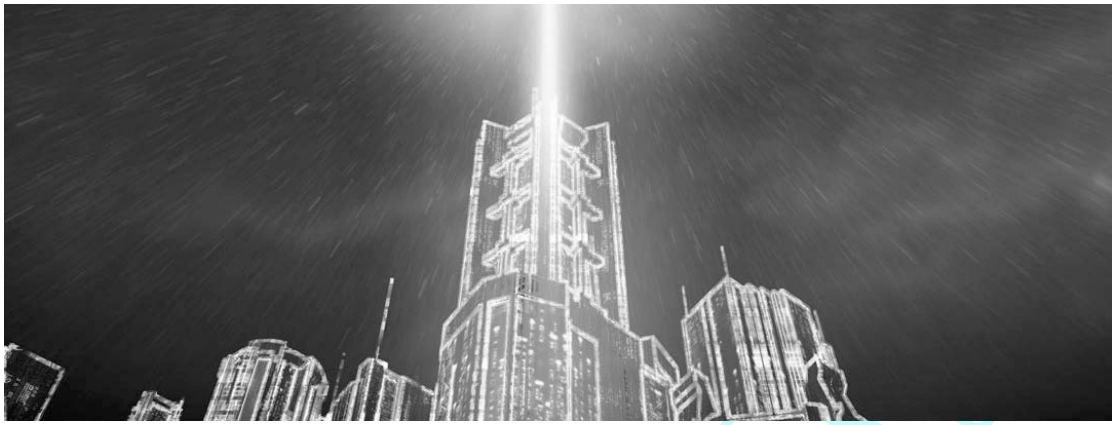
China BDN

在这些字段都能具有值。

DatabaseLink 对象把 MicroStation 元素与数据库记录相关联。要能在数据库中使用 DatabaseLink，该数据库必须有一个叫做“mscatalog”的表。DatabaseLink 对象通过 MSLink 属性与特定的表关联。

随后的一章将更详细地讨论数据库。

China BDN



28. 标签

标签存储并显示设计文件中与元素关联的信息，在文件与文件间显示相同类型的信息。存储在每个标签中的数据是不同的。例如，“Drawn By(制图人)”对每个文件来说都是有用的信息，但实际值会随文件的不同而不同。

本章使用随 MicroStation 一起安装的“Building”工程中的宏。

本章包括：

- 从基于选择集的标签中取得信息
- 取得文件中的所有标签
- 使用标签集
- 取得一个文件夹下所有文件中的所有标签
- 改变标签的值
- 改变多个文件中的多个标签值
- 把标签信息导出到一个文件中

从基于选择集的标签中取得信息

先让我们看一下标签的奇妙世界，让用户(在这种情况下，用户可能就是我们)选择一个标签然后从中取得一些基本信息。

```
Sub GetSelectedTagA()
    Dim myTag As TagElement
    Dim myElemEnum As ElementEnumerator
    Set myElemEnum = _
```

```
            Application.ActiveModelReference.GetSelectedElements
        While myElemEnum.MoveNext
            Select Case myElemEnum.Current.Type
                Case MsdElementType.msdElementTypeTag
                    Set myTag = myElemEnum.Current
                    If myTag Is Nothing = False Then
                        MsgBox myTag.Value
                    End If
            End Select
        Wend
    End Sub
```

本例建立了一个 ElmentEnumerator(元素枚举器)来查看 MicroStation 中选择的每个元素。我们查看每个选择的元素是否为标签元素，如果是，就在消息框中显示标签值。

标签值对我们来说非常重要，但还有比标签值更重要的。让我们扩展宏以包含标签名。

```
    Sub GetSelectedTagB()
        Dim myTag As TagElement
        Dim myElemEnum As ElementEnumerator
        Set myElemEnum = _
            Application.ActiveModelReference.GetSelectedElements
        While myElemEnum.MoveNext
            Select Case myElemEnum.Current.Type
            Case MsdElementType.msdElementTypeTag
                Set myTag = myElemEnum.Current
                If myTag Is Nothing = False Then
                    MsgBox myTag.TagDefinitionName & _
                        vbTab & myTag.Value
                End If
            End Select
```


China BDN

```
        Wend
    End Sub
```

通过标签名能知道标签值告诉我们的是制图人，还是校图人或是印图人。GetSelectedTagB 比前一个过程给出的信息多，但还能进一步改进。让我们取得标签的标签集名。

```
    Sub GetSelectedTagC()
        Dim myTag As TagElement
        Dim myElemEnum As ElementEnumerator
        Set myElemEnum = _
                Application.ActiveModelReference.GetSelectedElements
        While myElemEnum.MoveNext
            Select Case myElemEnum.Current.Type
                Case MsdElementType.msdElementTypeTag
                    Set myTag = myElemEnum.Current
                    If myTag Is Nothing = False Then
                        MsgBox myTag.TagSetName & vbTab & _
                        myTag.TagDefinitionName & vbTab & myTag.Value
                    End If
            End Select
        Wend
    End Sub
```

标签集(TagSet)告诉我们一个标签属于哪个组。例如，该标签属于标题框吗？它属于门的一部分吗？

至此我们得到了一些非常有用的信息。如果选择了四个标签，我们将看到四个消息框；如果选择了一个标签，将看到一个消息框。现在扩展代码以包含属于同一个标签集中的所有标签。

下一个例子显示标签信息并使用选择的标签，而且显示与选择的标签同级的所有标签。通过取得属于选择标签的基准元素(BaseElement)的所有标签来实现这一点。

```
Sub GetSelectedTagD()
```

```
    Dim myTag As TagElement
    Dim sibTags() As TagElement
    Dim myElemEnum As ElementEnumerator
    Dim I As Long
    Set myElemEnum = _
                Application.ActiveModelReference.GetSelectedElements
    While myElemEnum.MoveNext
        Select Case myElemEnum.Current.Type
            Case MsdElementType.msdElementTypeTag
                Set myTag = myElemEnum.Current
                sibTags = myTag.BaseElement.GetTags
                For I = LBound(sibTags) To UBound(sibTags)
                    MsgBox sibTags(I).TagSetName _
                            & vbTab & sibTags(I).TagDefinitionName _
                            & vbTab & sibTags(I).Value
                Next I
                Exit Sub
        End Select
    Wend
End Sub
```

我们使用了选择标签的 **BaseElement** 的 GetTags 方法。GetTags 填充一个由标签组成的数组，这些标签都属于一个元素。用 For ... Next 循环来查看每个标签的 TagSetName、TagDefinitionName 和 Value 属性。

取得文件中的所有标签

您已经看过了如何基于 MicroStation 的选择集提取标签信息。现在让我们从选择集方法转移到扫描整个文件中的标签元素并在消息框中显示每个标签的信息。

```
Sub GetTagsA()
    Dim myTag As TagElement
```

```
    Dim myElemEnum As ElementEnumerator
    Dim myFilter As New ElementScanCriteria
    myFilter.ExcludeAllTypes
    myFilter.IncludeType msdElementTypeTag
    Set myElemEnum = _
                    Application.ActiveModelReference.Scan(myFilter)
    While myElemEnum.MoveNext
        Set myTag = myElemEnum.Current
        MsgBox myTag.TagSetName & vbTab & myTag.TagDefinitionName _
                & vbTab & myTag.Value
    Wend
End Sub
```

建立一个新的 ElementScanCriteria 对象后，从过滤器中排除所有类型然后包含标签元素类型，这使得扫描整个文件但仅枚举出标签元素。有时想让用户选择标签，有时想处理整个文件而不要用户干涉 — 这两种情况我们都展示过了。

后续章节中将讨论如何将标签信息提取到 Microsoft Excel 中。

使用标签集

标签被分组在标签集中。前面的例子已经显示了每个标签的 TagSetName 属性。让我们查看一下当前设计文件中的标签集。

```
Sub GetTagsSetsA()
    Dim myTagSet As TagSet
    For Each myTagSet In Application.ActiveDesignFile.TagSets
        MsgBox myTagSet.Name
    Next
End Sub
```

既然知道了如何识别文件中的每个标签集，那么我们也能查看文件中每个标签集的每个标签定义(TagDefinition)。

```
Sub GetTagsSetsB()
    Dim myTagSet As TagSet
```


China BDN

```
    Dim myTagDef As TagDefinition
    For Each myTagSet In Application.ActiveDesignFile.TagSets
        For Each myTagDef In myTagSet.TagDefinitions
            MsgBox myTagSet.Name & vbTab & myTagDef.Name
        Next
    Next
End Sub
```

现在我们将执行相同的基本功能，但显示更多的标签信息：

```
Sub GetTagsSetsC()
    Dim myTagSet As TagSet
    Dim myTagDef As TagDefinition
    For Each myTagSet In Application.ActiveDesignFile.TagSets
        For Each myTagDef In myTagSet.TagDefinitions
            MsgBox "SetName: " & myTagSet.Name & vbCr & _
                   "TagName: " & myTagDef.Name & vbCr & _
                   "Prompt: " & myTagDef.Prompt & vbCr & _
                   "Default: " & myTagDef.DefaultValue
        Next
    Next
End Sub
```

取得一个文件夹下所有文件中的所有标签

只用几行代码就能取得文件中的所有标签，但打开指定文件夹下的每个文件并运行一个宏需要花费一些时间，例如运行宏 GetTagsSetsC，尤其是当文件夹下有成百甚至上千的文件时就更是如此了。让我们看一些代码，这些代码打开文件夹下的每个文件并将其中的标签信息打印到 VBA 的立即窗口中。当然了，把信息写入文件会更有用，所以随后也能看到这样的一个例子。

在继续编写代码前需要添加引用“Microsoft Scripting Runtime”。文件系统对象帮助我们遍历文件夹下的所有文件。

```
Sub GetFolderTags()
```

```
Dim myDGN As DesignFile
Dim myFSO As New Scripting.FileSystemObject
Dim myFolder As Scripting.Folder
Dim myFile As Scripting.File
Dim myTagSet As TagSet
Dim myTagDef As TagDefinition
Dim TargetTagset As String
Dim myTag As TagElement
Dim myElemEnum As ElementEnumerator
Dim myFilter As New ElementScanCriteria
TargetTagset = "TitleBlock"
Set myFolder = myFSO.GetFolder("C:\Documents and Settings\" & _
                               "All Users\Application Data\" & _
                               "Bentley\WorkSpace\Projects\" & _
                               "Examples\Building\Dgn")
For Each myFile In myFolder.Files
    Select Case myFile.Type
    Case "Bentley MicroStation Design File"
        Set myDGN = Application.OpenDesignFileForProgram( _
                                        myFile.Path, True)
        For Each myTagSet In myDGN.TagSets
            Select Case UCase(myTagSet.Name)
            Case UCase(TargetTagset)
                myFilter.ExcludeAllTypes
                myFilter.IncludeType msdElementTypeTag
                Set myElemEnum = _
                                myDGN.Models(1).Scan(myFilter)
                While myElemEnum.MoveNext
                    Set myTag = myElemEnum.Current
                    Debug.Print myFile.Name & vbTab & _
```


China BDN

```
                                    myTag.TagSetName & vbTab & _
                                    myTag.TagDefinitionName; vbTab & _
                                    myTag.Value & vbTab & _
                                    vbTab & myTag.ID.High & _
                                    vbTab & myTag.ID.Low
                        Wend
                    End Select
                Next
                myDGN.Close
            End Select
        Next
End Sub
```

本例以“for program”方式打开指定文件夹下的每个设计文件。“for program”的意思是在内存中打开文件但不在 MicroStation 中显示。当文件不在屏幕上显示时，能非常快速地打开并操作它们。

本例中提取了一些以前没有提取过的信息：ID 属性的 High 和 Low 分量。这个 ID 属性非常重要，因为它是 MicroStation 中一个元素的唯一标识，该值即使在不同的会话期也保持不变。所以在数据库中存储元素的 ID 属性能让我们快速且容易地识别 MicroStation 中的元素，即使在我们首次使用完这个文件后数小时、数天、数周或数月之后也没关系。

改变标签的值

现在我们将建立并使用过程 ChangeTag。该过程用 ID 属性的 High 和 Low 分量来取得和设置标签值。下面就是这个过程：

```
Sub ChangeTag(IDHigh As Long, IDLow As Long, NewValue As String)
    Dim TagID As DLong
    Dim myTag As TagElement
    TagID.High = IDHigh
    TagID.Low = IDLow
    Set myTag = Application.ActiveDesignFile.GetElementByID(TagID)
```

```
        myTag.Value = NewValue
        myTag.Rewrite
    End Sub
```

当有了一个 ID 后就能用 GetElementByID 过程来取得对应的元素。设置完基于 ID 的标签值后重写该标签元素。该过程不能独立运行，需要其它代码来调用它。让我们看一个调用它的过程：

```
    Sub TestChangeTagA()
        Dim myTag As TagElement
        Dim myEnum As ElementEnumerator
        Set myEnum = ActiveModelReference.GetSelectedElements
        While myEnum.MoveNext
            Select Case myEnum.Current.Type
                Case MsdElementType.msdElementTypeTag
                    ChangeTag myEnum.Current.ID.High, _
                              myEnum.Current.ID.Low, "ABC"
            End Select
        Wend
    End Sub
```

在现实世界中，我们把 ID 属性的 High 和 Low 分量保存在数据库中或其它一些存储机制中。这个例子是从选择的元素取得 ID 属性，然后用新过程 ChangeTag 来改变标签的值。

改变多个文件中的多个标签值

随着更深入地学习 VBA 编程，您会发现在很短的时间内就能在很大的范围内对某些东西作出很显著的改变。所以您的代码能做许多有益的事情或者干许多破坏性的事情。当开始用 VBA 操作多个文件时，一定要特别小心来确保您所做的修改是有益的而不是破坏性的。

改变单个文件中的标签不像改变多个文件中的多个标签那样令人激动，当然也不像那样危险。将要建立的过程 ChangeTag2 需要指定四个参数。

```
Sub ChangeTag2(FileName As String, TagSet As String, _
```

```
                    TagName As String, NewValue As String)
    Dim myDGN As DesignFile
    Dim myFilter As New ElementScanCriteria
    Dim myElemEnum As ElementEnumerator
    Dim myTag As TagElement
    Dim myModelRef As ModelReference
    Set myDGN = Application.OpenDesignFileForProgram(FileName, False)
    myFilter.ExcludeAllTypes
    myFilter.IncludeType msdElementTypeTag
    For Each myModelRef In myDGN.Models
        Set myElemEnum = myModelRef.Scan(myFilter)
        While myElemEnum.MoveNext
            Set myTag = myElemEnum.Current
            If StrComp(myTag.TagSetName, TagSet) = 0 Then
                If StrComp(myTag.TagDefinitionName, _
                                    TagName) = 0 Then
                    myTag.Value = NewValue
                    myTag.Rewrite
                End If
            End If
        Wend
    Next
    myDGN.Save
    myDGN.Close
End Sub
```

本过程以“For Program”方式打开指定的文件，以指定的标签集和标签名扫描文件并设置标签值。下面是使用 ChangeTag2 的一个过程。

```
    Sub TestChangeTagB()
        Dim myFSO As New Scripting.FileSystemObject
        Dim myFolder As Scripting.Folder
```

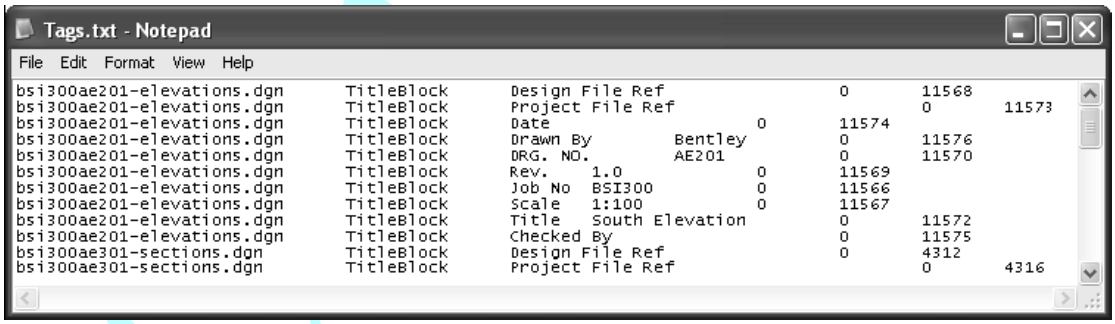

China BDN

```
    Dim myFile As Scripting.File
    Set myFolder = myFSO.GetFolder("C:\Documents and Settings\" & _
                                   "All Users\Application Data\" & _
                                   "Bentley\WorkSpace\Projects\" & _
                                   "Examples\Building\Dgn")
    For Each myFile In myFolder.Files
        Select Case myFile.Type
            Case "Bentley MicroStation Design File"
                ChangeTag2 myFile.Path, "TitleBlock", _
                                        "Checked By", "J Winters"
        End Select
    Next
End Sub
```

以上过程修改指定文件夹下的每个设计文件，保存它们并给所有位于标题框标签集中叫做“Checked By”的标签一个“J Winters”的值。功能强大吗？是的。危险吗？有潜在的危险性。小心编制您的程序，让它对大家有益而不是有害。

把标签信息导出到一个文件中

所有的消息框和 Debug.Print 语句做得都不好，因为提取和显示的数据不能保存。我们将以一个简单的 ASCII 文本文件开始。



这就是我们想建立的输出。它与在立即窗口中打印的信息一样。

```
Sub ExportFolderTags()
    Dim myDGN As DesignFile
    Dim myFSO As New Scripting.FileSystemObject
    Dim myFolder As Scripting.Folder
```

```
Dim myFile As Scripting.File
Dim myTagSet As TagSet
Dim myTagDef As TagDefinition
Dim TargetTagset As String
Dim myTag As TagElement
Dim myElemEnum As ElementEnumerator
Dim myFilter As New ElementScanCriteria
Dim FFile As Long
FFile = FreeFile
Open "C:\MicroStation VBA\Tags.txt" For Output As #FFile
TargetTagset = "TitleBlock"
Set myFolder = myFSO.GetFolder("C:\Documents and Settings\" & _
                               "All Users\Application Data\" & _
                               "Bentley\WorkSpace\Projects\" & _
                               "Examples\Building\Dgn")
For Each myFile In myFolder.Files
    Select Case myFile.Type
    Case "Bentley MicroStation Design File"
        Set myDGN = Application.OpenDesignFileForProgram( _
                                    myFile.Path, True)
        For Each myTagSet In myDGN.TagSets
            Select Case UCase(myTagSet.Name)
            Case UCase(TargetTagset)
                myFilter.ExcludeAllTypes
                myFilter.IncludeType msdElementTypeTag
                Set myElemEnum = _
                        myDGN.Models(1).Scan(myFilter)
                While myElemEnum.MoveNext
                    Set myTag = myElemEnum.Current
                    Print #FFile, myFile.Name & vbTab & _
```


China BDN

```
                                    myTag.TagSetName & vbTab & _
                                    myTag.TagDefinitionName; vbTab & _
                                    myTag.Value & vbTab & _
                                    vbTab & myTag.ID.High & _
                                    vbTab & myTag.ID.Low
                            Wend
                        End Select
                    Next
                    myDGN.Close
                End Select
            Next
            Close #FFile
End Sub
```

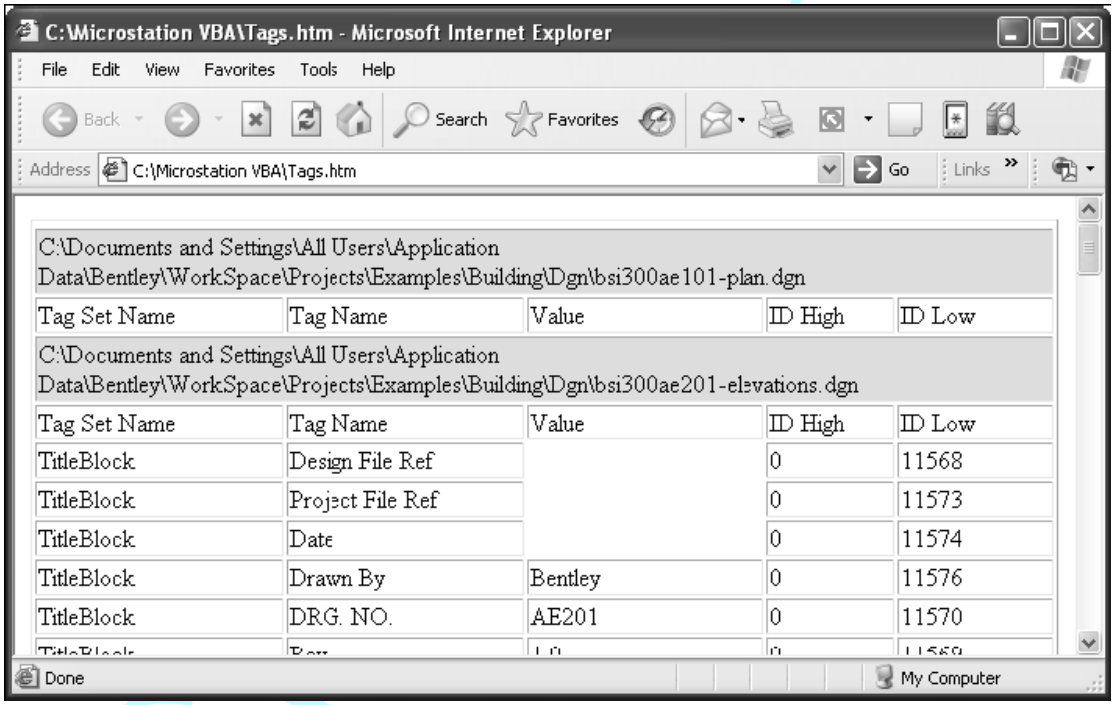
写文本文件很简单。当然了，一个 ASCII 文本文件有助于在记事本程序中浏览但格式不太好。让我们修改上例，以建立.htm 文件来代替建立.txt 文件。

```
Sub ExportFolderTagsToHTML()
    Dim myDGN As DesignFile
    Dim myFSO As New Scripting.FileSystemObject
    Dim myFolder As Scripting.Folder
    Dim myFile As Scripting.File
    Dim myTagSet As TagSet
    Dim myTagDef As TagDefinition
    Dim TargetTagset As String
    Dim myTag As TagElement
    Dim myElemEnum As ElementEnumerator
    Dim myFilter As New ElementScanCriteria
    Dim FFile As Long
    FFile = FreeFile
    Open "C:\MicroStation VBA\Tags.htm" For Output As #FFile
    Print #FFile, "<table width=660 border=1>" & vbCr
```

```
Print #FFile, vbTab & "<tr><td></td></tr>" & vbCr
TargetTagset = "TitleBlock"
Set myFolder = myFSO.GetFolder("C:\Documents and Settings\" & _
                               "All Users\Application Data\" & _
                               "Bentley\WorkSpace\Projects\" & _
                               "Examples\Building\Dgn")
For Each myFile In myFolder.Files
    Select Case myFile.Type
    Case "Bentley MicroStation Design File"
        Print #FFile, "<tr><td colspan=5>" & _
                      myFile.Path & "</td></tr>" & vbCr
        Print #FFile, vbTab & "<tr><td>Tag Set Name</td>" & _
                      "<td>Tag Name</td>" & _
                      "<td>Value</td>" & _
                      "<td>ID High</td>" & _
                      "<td>ID Low</td></tr>"
        Set myDGN = Application.OpenDesignFileForProgram( _
                                myFile.Path, True)
        For Each myTagSet In myDGN.TagSets
            Select Case UCase(myTagSet.Name)
            Case UCase(TargetTagset)
                myFilter.ExcludeAllTypes
                myFilter.IncludeType msdElementTypeTag
                Set myElemEnum = myDGN.Models(1).Scan(myFilter)
                While myElemEnum.MoveNext
                    Set myTag = myElemEnum.Current
                    Print #FFile, vbTab & "<tr><td>" & _
                        myTag.TagSetName & "</td>" & _
                        "<td>" & myTag.TagDefinitionName & _
                        "</td>" & _
```


China BDN

```
                                    "<td>" & myTag.Value & "</td>" & _
                                    "<td>" & myTag.ID.High & "</td>" & _
                                    "<td>" & myTag.ID.Low & "</td></tr>" & vbCr
                            Wend
                        End Select
                    Next
                    myDGN.Close
                End Select
            Next
            Print #FFile, "</table>"
            Close #FFile
End Sub
```



编写很少的 HTML 代码就能使导出的数据以更可视化和更佳的结构格式来显示。由于网页浏览器能够不依赖于操作系统在大多数计算机上找到，所以 HTML 成为一种几乎能被所有人读取的标准格式。

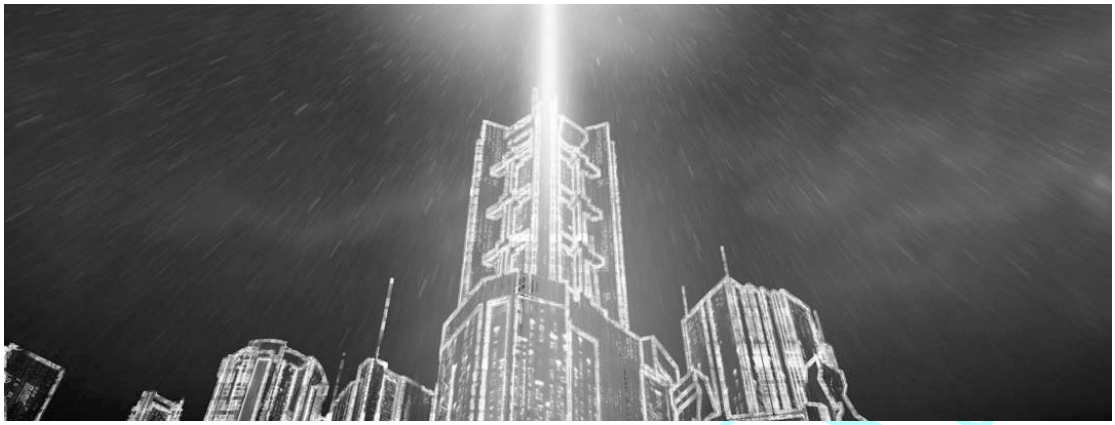
本章回顾

标签包含有用的信息。用 VBA 存取标签能使我们控制当前设计文件中的标签，也能控制指定文件夹下每个文件中的标签。正如您刚看到的，读取、改变标

签值也很容易。把标签数据导出成 ASCII 文件能让我们使用这些数据，或者在像记事本和网页浏览器之类的程序中查看它们。

后续的一章提供了将标签信息提取到 Microsoft Excel 中的例子。还能将标签信息提取到数据库中或作为电子邮件的主体。没有做不到只有想不到。

China BDN



29. XML

想象一下你面对这样一个任务：设计一种新的描述和保存数据的方法，该方法要足够灵活以能处理各种数据模型、数据类型等等。它必须能描述一辆轿车、一家人和一个农场。你能拿出什么样的方案？希望你能拿出像 XML 这样的一些东西，因为 XML 就是用于处理各种数据结构和类型的。

本章包括：

- XML 是什么？
- XML 文件结构
- 用 VBA 读取 XML 文件

XML 是什么？

XML 是扩展标记语言(Extensible Markup Language)的缩写。XML 文档通常具有“xml”文件扩展名。虽然 XML 文件符合一个通用规范，但它们所包含的数据会因文件或结构的不同而有很大的不同。

公司和组织建立其自己的 XML 文档定义，以其自己的数据结构来保存其自己的数据类型。例如，用于保存财务交易的 XML 文档、保存 GIS 信息的 XML 文档和保存建筑材料清单的 XML 文档。XML 格式已经被用于保存家谱、运动统计和菜谱。Microsoft Excel 和 Microsoft Access 能将其数据导出为 XML 文件。

XML 文件结构

XML 文件结构要足够灵活和强大。下面就是 XML 文件的一小段：

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<dataroot xmlns:od="urn:schemas-microsoft-com:officedata"
generated="2005-12-28T08:57:09">
<parcel>
    <mslink>1</mslink>
    <old_map_no>119-L</old_map_no>
    <group_no>A</group_no>
    <parcel_no>9</parcel_no>
    <clt_no>119-LA 9</clt_no>
    <owner>CANTRELL WILLIAM B &amp;amp; EVELYN W</owner>
    <parc_value>9000</parc_value>
    <house_num>220</house_num>
    <str_name>BRIDGEWATER RD</str_name>
    <city>KNOXVILLE</city>
    <state>TENNESSEE</state>
    <zip_code>37919</zip_code>
    <county>KNOX</county>
    <district>NW-COUNTY</district>
    <zone_class>RA</zone_class>
    <block_num>0</block_num>
    <lot_num>0</lot_num>
    <subd_name>CRESTWOOD HILLS UNIT 1</subd_name>
    <parc_area>16036</parc_area>
    <perimeter>511</perimeter>
    <mapid>6</mapid>
    <currdate>1986-09-19T00:00:00</currdate>
    <txt>This is a memo field, used for really long text entries</txt>
</parcel>
```

这个 XML 文件是从 Microsoft Access 数据库导出的。说得更精确些，它是从随 MicroStation 安装的 gis.mdb 文件导出的，“Parcel”表被导出为一个 XML

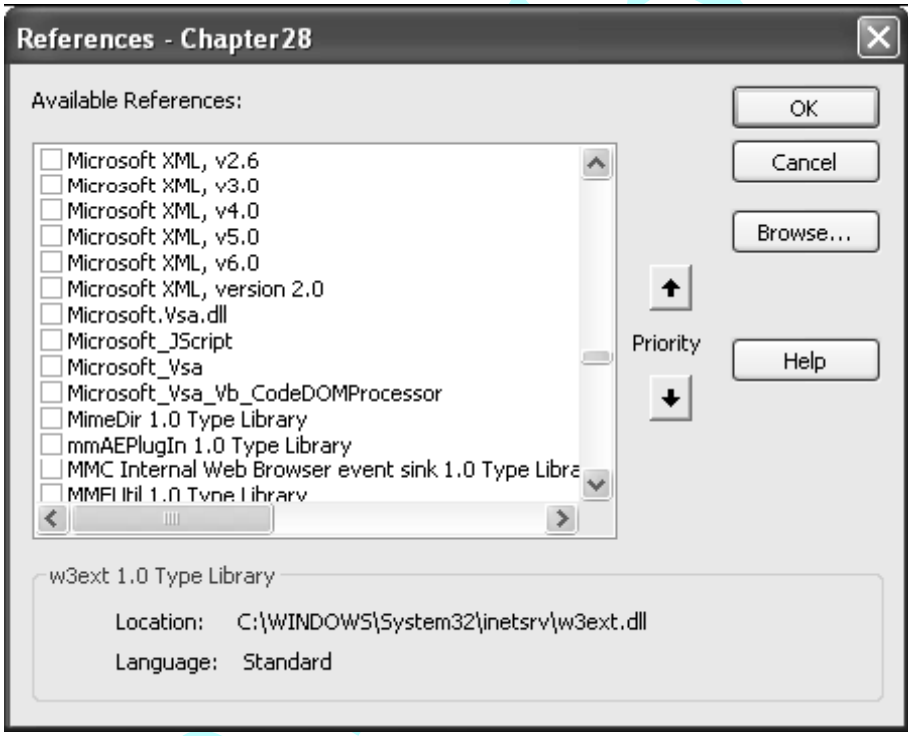
文件。

XML 文件正被越来越多地用于代替传统数据库，看一下它的结构就很容易明白这是为什么了。从这个文件中能看到节点 City、State、Zip_Code、County、District、Parc-Area、Perimeter 等等。

读取 XML 文件

有几种读取 XML 文件的方式。可以用标准的 VBA 文件输入/输出调用读取 XML 文件的每一行并分析它，但还有更好的方式。

微软已经向我们提供了操作 XML 文件的一些工具。第一步就是要添加 Microsoft XML DLL 文件引用。

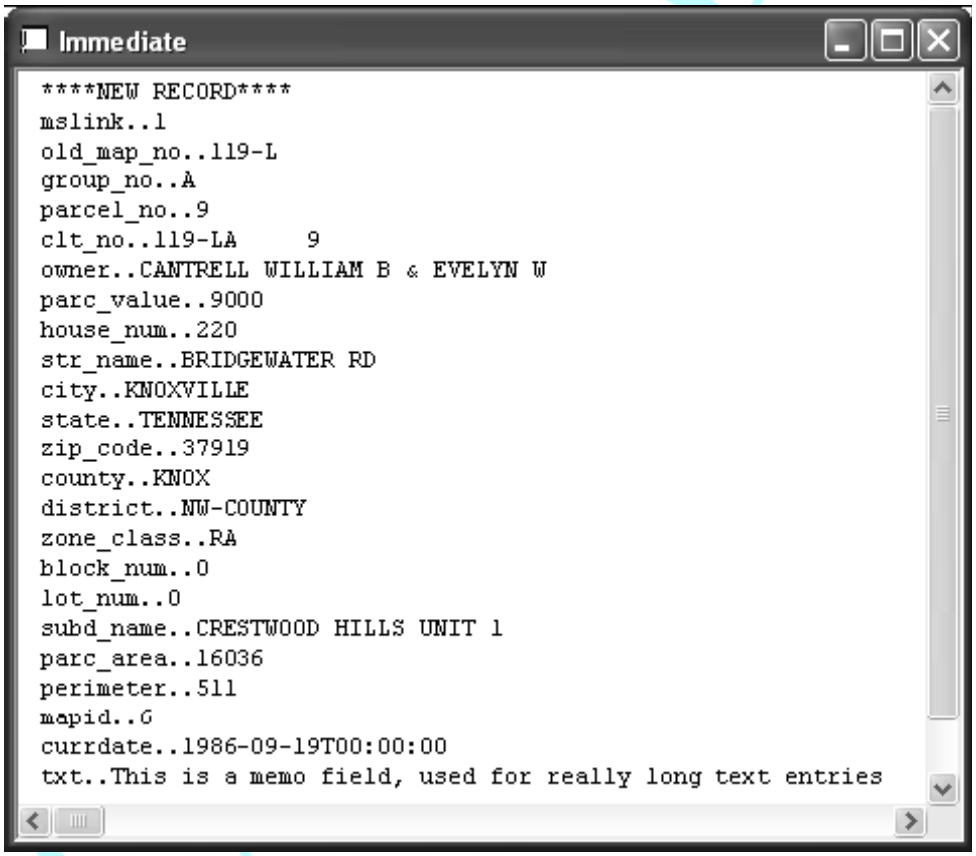


每台计算机都具有不同版本的 Microsoft XML 库。在上例中，v6.0 是最高的版本，所以我在这个例子中用它。

```
Sub ReadXMLFile()
    Dim myXML As New MSXML2.DOMDocument
    Dim myXElem As MSXML2.IXMLDOMElement
    Dim myXRecord As MSXML2.IXMLDOMElement
    Dim myXField As MSXML2.IXMLDOMElement
    myXML.async = False
```

```
    myXML.validateOnParse = False
    myXML.Load "c:\MicroStation VBA\parcel.xml"
    Set myXElem = myXML.documentElement
    For Each myXRecord In myXElem.childNodes
        Debug.Print "****NEW RECORD****"
        For Each myXField In myXRecord.childNodes
            Debug.Print myXField.baseName & ".." & myXField.Text
        Next
    Next
End Sub
```

本例装载 XML 文件然后在立即窗口中打印每条带有其子节点(字段)的记录。



ReadXMLFile 过程从文件头开始查看每一项直到文件尾为止。

如果仅想显示 mslink 为 531 的那条记录，可以使用如下代码：

```
Sub ReadXMLFileB()
    Dim myXML As New MSXML2.DOMDocument
    Dim myXElem As MSXML2.IXMLDOMElement
```

```
    Dim myXRecord As MSXML2.IXMLDOMElement
    Dim myXField As MSXML2.IXMLDOMElement
    myXML.async = False
    myXML.validateOnParse = False
    myXML.Load "c:\MicroStation VBA\parcel.xml"
    Set myXElem = myXML.documentElement
    For Each myXRecord In myXElem.childNodes
        If myXRecord.childNodes(0).Text = "531" Then
            For Each myXField In myXRecord.childNodes
                Debug.Print myXField.baseName & ".." & myXField.Text
            Next
        End If
    Next
End Sub
```

本例在 XML 文件中查看每条记录，当发现 mslink 为 531 的那条记录时将其字段打印到立即窗口。这段代码能正常工作但效率不是太高。

让我们看下一个例子。它不是在 XML 文件中查看每条记录，而是仅取得 mslink 属性，查看它们然后在立即窗口中打印每个字段。

```
Sub ReadXMLFileC()
    Dim myXML As New MSXML2.DOMDocument
    Dim myXElem As MSXML2.IXMLDOMElement
    Dim myXList As MSXML2.IXMLDOMNodeList
    Dim myXRecord As MSXML2.IXMLDOMElement
    Dim myXField As MSXML2.IXMLDOMElement
    myXML.async = False
    myXML.validateOnParse = False
    myXML.Load "c:\MicroStation VBA\parcel.xml"
    Set myXElem = myXML.documentElement
    Set myXList = myXML.getElementsByTagName("mslink")
    For Each myXRecord In myXList
```

```
        If myXRecord.Text = "531" Then
            For Each myXField In myXRecord.parentNode.childNodes
                Debug.Print myXField.baseName & ".." & myXField.Text
            Next
        End If
    Next
End Sub
```

这段代码运行得比上个例子快，但可以改进成查找第一个具有 mslink 值为 531 的节点。

```
    Sub ReadXMLFileD()
        Dim myXML As New MSXML2.DOMDocument
        Dim myXField As MSXML2.IXMLDOMElement
        Dim myXNode As MSXML2.IXMLDOMNode
        myXML.async = False
        myXML.validateOnParse = False
        myXML.Load "c:\MicroStation VBA\parcel.xml"
        Set myXElem = myXML.documentElement
        Set myXNode = myXML.selectSingleNode("//parcel[mslink=531]")
            For Each myXField In myXNode.childNodes
                Debug.Print myXField.baseName & ".." & myXField.Text
            Next
    End Sub
```

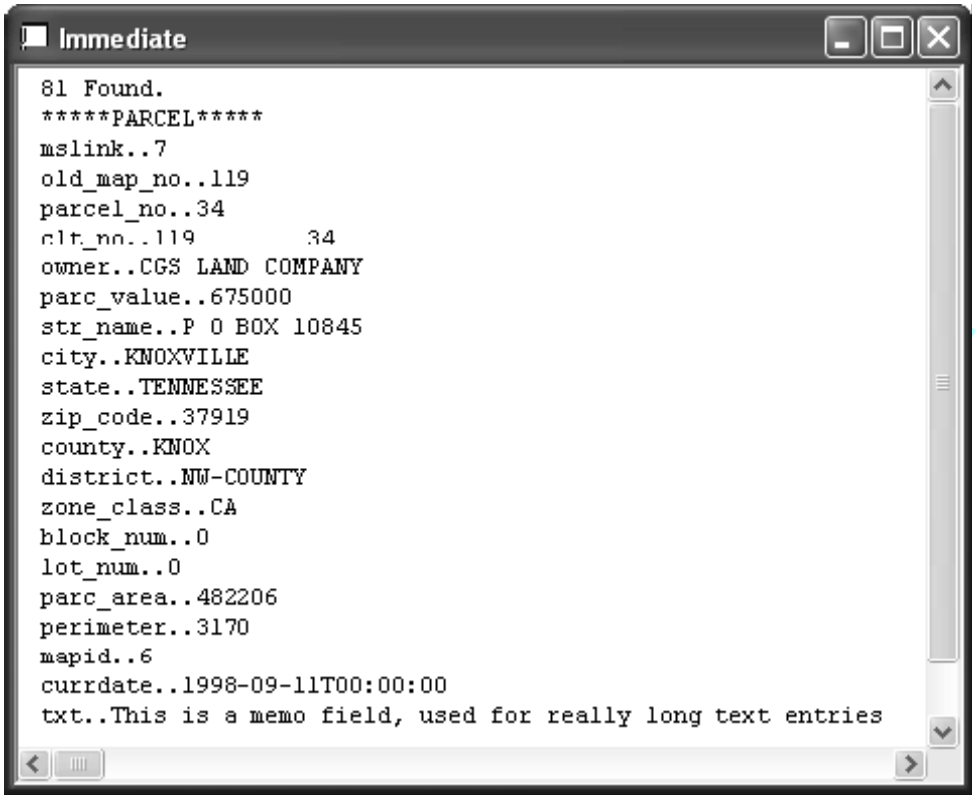
我们在使用 Microsoft XML 库方面取得了一些进展。它快速、高效且易于修改。无论 parcel 对象位于 XML 文件中多深的层，这段代码都能找到 parcel 对象的 mslink 属性等于 531 的第一个实例。

看一下前面所示的拷屏图，能看到大量使用的字段，其中之一是“parc_value”。您可以修改上面的代码，用 parc_value 代替 mslink 并查找 parcel 值为 250000 的记录。但这样做只能查到 parcel 值精确等于 250000 的记录。让我们分析下面的代码，它将返回所有大于或等于$250000 的所有 parcel(不仅一

个)。

```
Sub ReadXMLFileE()
    Dim myXML As New MSXML2.DOMDocument
    Dim myXList As MSXML2.IXMLDOMNodeList
    Dim myXField As MSXML2.IXMLDOMElement
    Dim myXNode As MSXML2.IXMLDOMNode
    myXML.async = False
    myXML.validateOnParse = False
    myXML.Load "c:\MicroStation VBA\parcel.xml"
    Set myXList = myXML.selectNodes("//parcel[parc_value>=250000]")
    Debug.Print myXList.Length & " Found."
    For Each myXNode In myXList
        Debug.Print "*****PARCEL*****"
        For Each myXField In myXNode.childNodes
            Debug.Print myXField.baseName & ".." & myXField.Text
        Next
    Next
End Sub
```

China BDN



改动部分的代码初看起来有点像 SQL 语句，其实它是一个 XPath 语句。

仔细比较这里所示的拷屏图和几页前的拷屏图能看出来，这里的拷屏图缺少了“group_no”字段和“house_num”字段。前一个例子用 childNodes(0)来取得“mslink”字段，通过索引号而不是其名字来定位它。在这里您能看到某些记录的字段不完全，所以在用索引来定位字段(或节点)时应当特别小心 — 用名字来定位一个字段要好得多。

下面的例子显示每条 parcel(一块土地)面积大于或等于 1 英亩的记录的信息。

```
Sub ReadXMLFileF()
    Dim myXML As New MSXML2.DOMDocument
    Dim myXList As MSXML2.IXMLDOMNodeList
    Dim myXField As MSXML2.IXMLDOMElement
    Dim myXNode As MSXML2.IXMLDOMNode
    myXML.async = False
    myXML.validateOnParse = False
    myXML.Load "c:\MicroStation VBA\parcel.xml"
    Set myXList = myXML.selectNodes("//parcel[parc_area>=43560]")
```

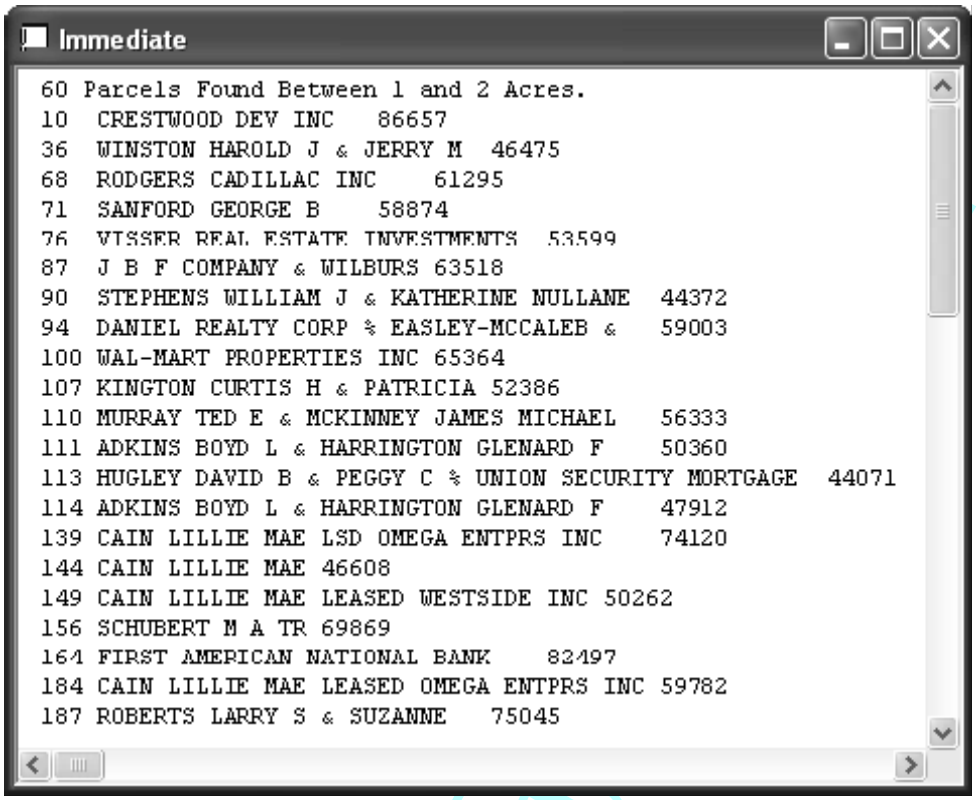
```
    Debug.Print myXList.Length & " Parcels Found >= 1 Acre."
    For Each myXNode In myXList
      Debug.Print myXNode.selectSingleNode(".//mslink").Text & vbTab & _
                  myXNode.selectSingleNode(".//owner").Text & vbTab & _
                  myXNode.selectSingleNode(".//parc_area").Text
    Next
End Sub
```

上例用 selectNodes 来选择多个满足指定准则的节点，然后查看找到的每个节点并取得其 mslink、owner 和 parc_area 的值。注意这里使用了带句点(.)的“.//”代替了“//”，使用一个句点表明想相对于当前上下文来查找。如果上例中不用句点，将只能在 XML 文档中找到第一个 mslink、owner 和 parc_area 节点。

再看一个读取该 XML 文档的例子，当然该 XML 文档是由 Access 导出的。

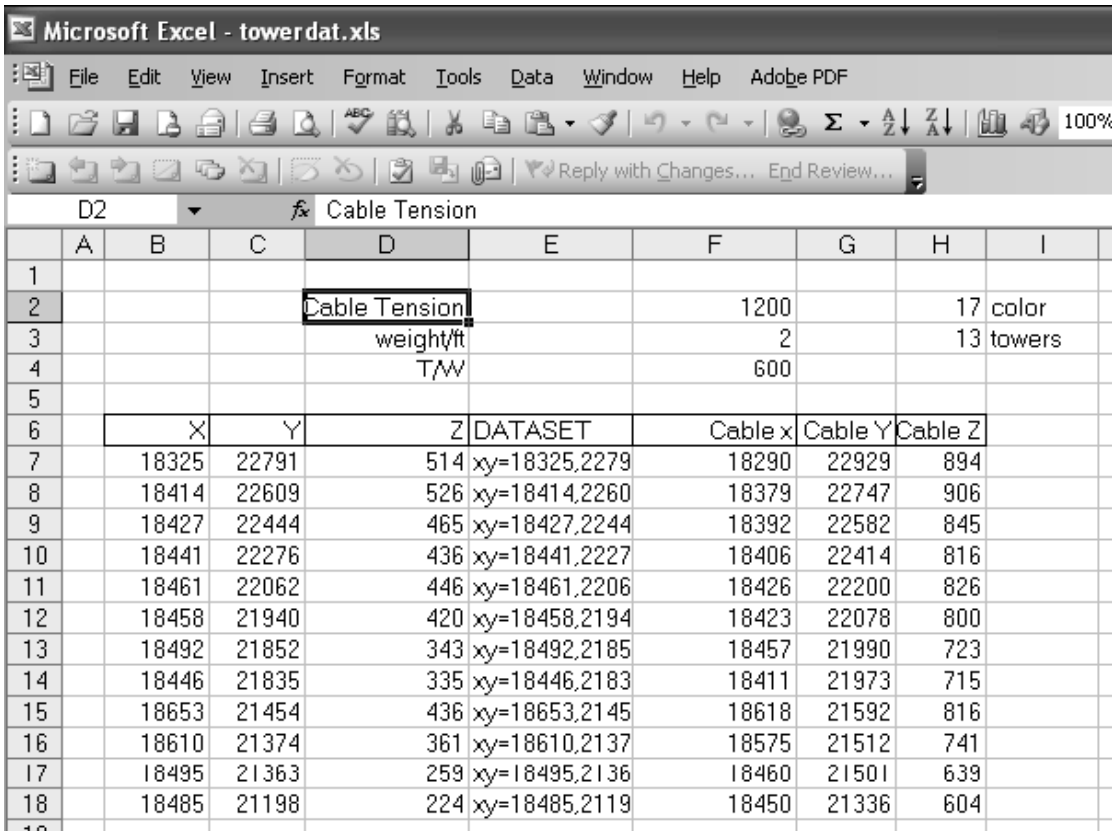
```
Sub ReadXMLFileG()
    Dim myXML As New MSXML2.DOMDocument
    Dim myXList As MSXML2.IXMLDOMNodeList
    Dim myXField As MSXML2.IXMLDOMElement
    Dim myXNode As MSXML2.IXMLDOMNode
    myXML.async = False
    myXML.validateOnParse = False
    myXML.Load "c:\MicroStation VBA\parcel.xml"
    Set myXList = myXML.selectNodes("//parcel[parc_area>=43560 " & _
                  "and parc_area<=87120]")
    Debug.Print myXList.Length & " Parcels Found Between 1 and 2 Acres."
    For Each myXNode In myXList
      Debug.Print myXNode.selectSingleNode(".//mslink").Text & _
                  vbTab & myXNode.selectSingleNode(".//owner").Text & _
                  vbTab & myXNode.selectSingleNode(".//parc_area").Text
    Next
End Sub
```

现在我们扩展过滤规则，使其仅返回 parc_area 值在 1 到 2 英亩间的节点。再次把 mslink、owner 和 parc_area 的节点值输出到了立即窗口中。



正如前面提及的，XML 文档具有特定的结构。我们一直使用的 XML 文件是从 Microsoft Access 导出的。现在看一个从 Microsoft Excel 导出的 XML 文件。

China



Microsoft Excel - towerdat.xls

File Edit View Insert Format Tools Data Window Help Adobe PDF

D2 Cable Tension

	A	B	C	D	E	F	G	H	I
1									
2				Cable Tension		1200		17	color
3				weight/ft		2		13	towers
4				T/W		600			
5									
6		X	Y	Z	DATASET	Cable x	Cable Y	Cable Z	
7		18325	22791	514	xy=18325,2279	18290	22929	894	
8		18414	22609	526	xy=18414,2260	18379	22747	906	
9		18427	22444	465	xy=18427,2244	18392	22582	845	
10		18441	22276	436	xy=18441,2227	18406	22414	816	
11		18461	22062	446	xy=18461,2206	18426	22200	826	
12		18458	21940	420	xy=18458,2194	18423	22078	800	
13		18492	21852	343	xy=18492,2185	18457	21990	723	
14		18446	21835	335	xy=18446,2183	18411	21973	715	
15		18653	21454	436	xy=18653,2145	18618	21592	816	
16		18610	21374	361	xy=18610,2137	18575	21512	741	
17		18495	21363	259	xy=18495,2136	18460	21501	639	
18		18485	21198	224	xy=18485,2119	18450	21336	604	

“towerdat.xls”电子表格随 MicroStation 安装。当把它导出为“电子表格 XML”文档时，可以用 Microsoft XML 引用来遍历其数据。

```
Sub ReadXMLFile1()
    Dim myXML As New MSXML2.DOMDocument
    Dim myWSheets As MSXML2.IXMLDOMNodeList
    Dim myWSheet As MSXML2.IXMLDOMNode
    Dim myRows As MSXML2.IXMLDOMNodeList
    Dim myRow As MSXML2.IXMLDOMNode
    Dim myCells As MSXML2.IXMLDOMNodeList
    Dim myCell As MSXML2.IXMLDOMNode
    myXML.async = False
    myXML.validateOnParse = False
    myXML.Load "c:\MicroStation VBA\towerdat.xml"
    Set myWSheets = myXML.getElementsByTagName("Worksheet")
    For Each myWSheet In myWSheets
        Set myRows = myWSheet.selectNodes(".//Row")
```

```
            For Each myRow In myRows
                Set myCells = myRow.selectNodes(".//Cell")
                For Each myCell In myCells
                    Debug.Print myCell.Text
                Next
            Next
        Next
End Sub
```

Microsoft Excel XML 文件的结构不同于 Microsoft Access 数据库 XML 的导出。在声明完使用和装载 XML 文件的变量后，取得文件中的所有工作表，然后查看每个工作表并取得工作表中的所有行，进而再取得每行中的所有单元并查看每个单元。这第一个 Excel XML 宏的结果是，将每个单元中的值(存储在 Text 属性中)显示在立即窗口中。



> 虽然XML提供了一种标准的保存和检索信息的机制，但其结构随格式的不同而不同。您能在 http://www.w3.org/XML读到完整的XML文件规范。

您能看到值了，但当您取得一个值时如何知道它在哪一行和哪一列？要回答这个问题，必须多了解一些工作表 XML 的结构。

每个单元都具有一个“Address”属性，它为开发者带来了方便，但将增大 XML 文件的大小。所以在需要指定一个行或单元的位置时往往使用索引属性。

工作表的第一行有一个叫做“Index”的属性，它表明您要查看的行号。每行的第一个单元也有一个叫做“Index”的属性，单元的索引表明单元所在的列。

所以在行索引和单元索引间，您能准确地知道第一行的第一个单元位于何处。

第一行的第二个单元含义不同。如果第一个单元位于第 2 列且第二个单元没有索引属性，则第二个单元位于第 3 列。同一行的下个单元如果没有索引属性则应在第 4 列。如果下个单元的索引属性值为 8，则它位于第 8 列。在第一个单元使用索引，单元不与另一单元直接相邻时也要使用索引。

第二行以相似的方式工作。如果 XML 文件中的第二行具有一个索引属性，则其行号在索引属性中指定。如果没有索引属性，该行就直接位于前一行的下面。

对原来的代码稍作改动就能跟踪单元的行和列。

```
Sub ReadXMLFile2()
    Dim myXML As New MSXML2.DOMDocument
    Dim myWSheets As MSXML2.IXMLDOMNodeList
    Dim myWSheet As MSXML2.IXMLDOMNode
    Dim myRows As MSXML2.IXMLDOMNodeList
    Dim myRow As MSXML2.IXMLDOMNode
    Dim myCells As MSXML2.IXMLDOMNodeList
    Dim myCell As MSXML2.IXMLDOMNode
    Dim myRowAtt As MSXML2.IXMLDOMAttribute
    Dim myColAtt As MSXML2.IXMLDOMAttribute
    Dim myAtts As MSXML2.IXMLDOMNamedNodeMap
    Dim CurRow As Long
    Dim CurCol As Long
    myXML.async = False
    myXML.validateOnParse = False
    myXML.Load "c:\MicroStation VBA\towerdat.xml"
    Set myWSheets = myXML.getElementsByTagName("Worksheet")
    For Each myWSheet In myWSheets
        CurRow = 1
        CurCol = 1
        Set myRows = myWSheet.selectNodes(".//Row")
        For Each myRow In myRows
```

```
            Set myAtts = myRow.Attributes
            Set myRowAtt = myAtts.getNamedItem("ss:Index")
            If Not myRowAtt Is Nothing Then
                CurRow = myRowAtt.Text
            Else
                CurRow = CurRow + 1
            End If
            CurCol = 1
            Set myCells = myRow.selectNodes(".//Cell")
            For Each myCell In myCells
                Set myAtts = myCell.Attributes
                Set myColAtt = myAtts.getNamedItem("ss:Index")
                If Not myColAtt Is Nothing Then
                    CurCol = myColAtt.Text
                Else
                    CurCol = CurCol + 1
                End If
                Debug.Print "R" & CurRow & "C" & CurCol & "=" & myCell.Text
            Next
        Next
    Next
End Sub
```

上面的 ReadXMLFile2 过程打印出单元的行和列以及文本。实现方法是：查找行和单元的索引属性，找到时用它们作为变量 CurRow 和 CurCol 的当前值；未找到时对 CurRow 和 CurCol 的值加 1(根据所查找的是一个行对象还是一个单元对象而定，当为行对象时对 CurRow 加 1；当为单元对象时对 CurCol 加 1)。

China BDN



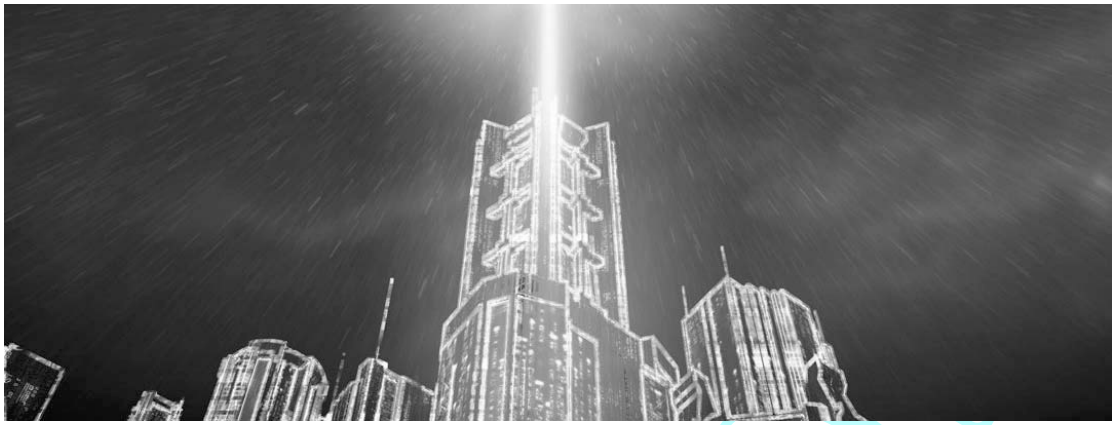
这里是 ReadXMLFile2 的输出。注意，有些单元的文本属性是空串。

本章回顾

XML 是一项谈论了好几年的技术，它也正成为应用更广泛的技术。读取 XML 文件不仅现在有用，而且在将来的日子里会成为更重要的技术。

由于每个 XML 文档定义都不一样，所以需要化些时间学习如何遍历一种新的 XML 文件类型。

China



30. 批处理

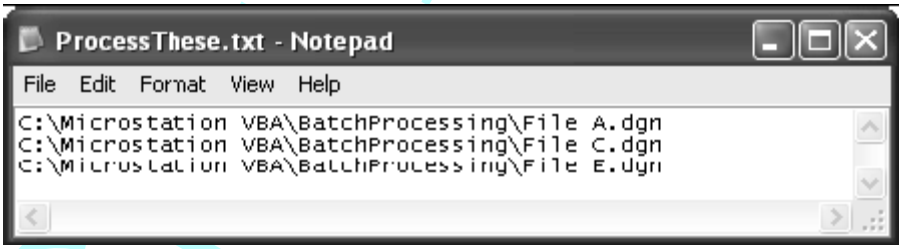
快速处理 10 个、100 个甚至 1000 个文件其实很简单。

本章包括：

- 处理列在一个 ASCII 文件中的文件
- 处理一个文件夹下的所有文件
- 处理一个文件夹及其子文件夹下的所有文件
- 建立选择文件的用户界面
- 记录批文件处理过程

处理列在一个 ASCII 文件中的文件

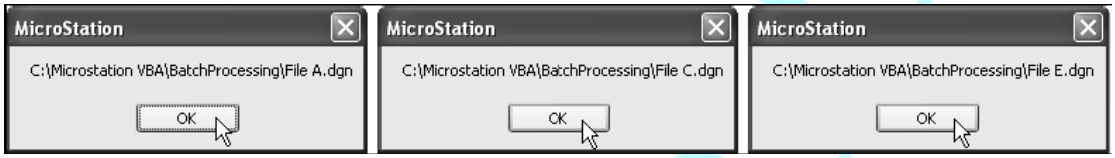
通常把要处理的文件列在一个 ASCII 文本文件中。下面就是一个例子：



第一个例子打开文件 ProcessThese.txt，读取其中的每一行，然后在消息框中显示每一行。

```
Sub ProcessASCII()
    Dim FileToOpen As String
    Dim BatchFile As String
    Dim FFile As Long
```

```
    FFile = FreeFile
    BatchFile = "C:\MicroStation VBA\BatchProcessing\ProcessThese.txt"
    Open BatchFile For Input As #FFile
    While EOF(FFile) = False
        Line Input #FFile, FileToOpen
        MsgBox FileToOpen
    Wend
End Sub
```



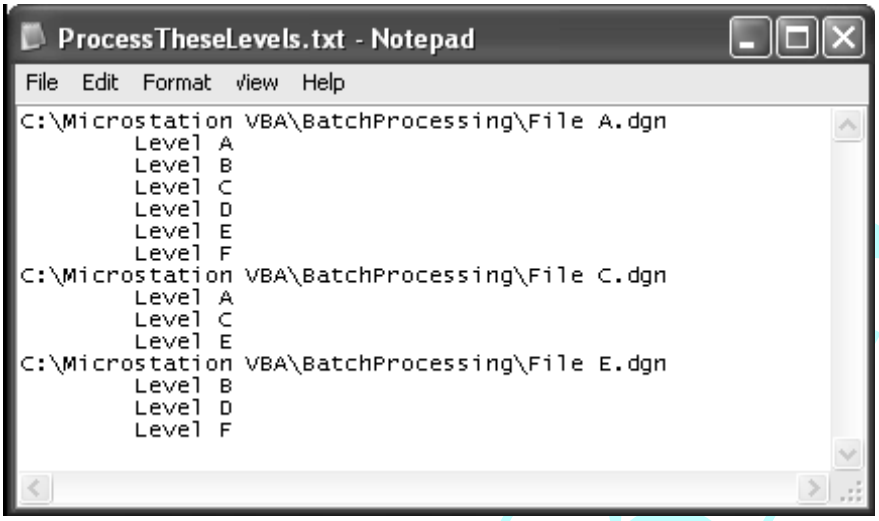
既然有了基础，就让我们扩展它们吧。

```
Sub ProcessASCIIB()
    Dim FileToOpen As String
    Dim BatchFile As String
    Dim FFile As Long
    FFile = FreeFile
    BatchFile = "C:\MicroStation VBA\BatchProcessing\ProcessThese.txt"
    Open BatchFile For Input As #FFile
    While EOF(FFile) = False
        Line Input #FFile, FileToOpen
        Application.OpenDesignFile FileToOpen, True
        MsgBox "Do Something"
    Wend
End Sub
```

以上代码不是只在消息框中显示文件名，而是在 MicroStation 中打开每个文件，“只读”参数设置为 True。当需要从文件中提取信息但不想作任何改变时，这样做很有用。注意，我们仅打开了文件而没有关闭它们。

让我们改变 ASCII 文件的格式。在文件路径和文件名后是要处理的层的行。文件中的每个层行应以一个制表符开始然后是层名。向文件中添加完层名后执行

“另存为”将文件保存为新名字 **ProcessTheseLevels.txt**。

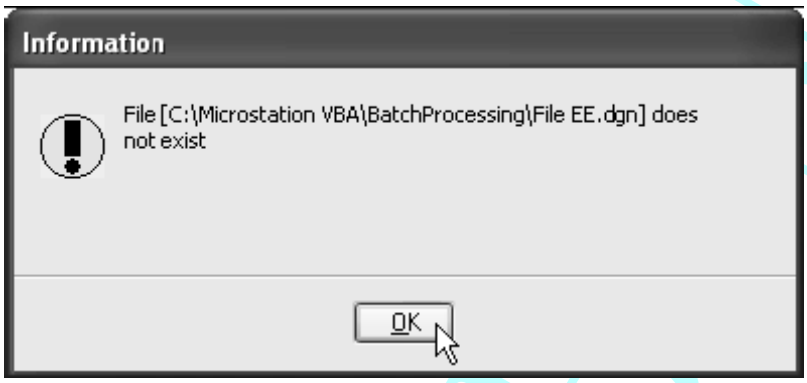


下面的代码打开每个文件并在消息框中显示每个层名。

```
Sub ProcessASCIIC()
    Dim FileToOpen As String
    Dim BatchFile As String
    Dim strLIN As String
    Dim FFile As Long
    FFile = FreeFile
    BatchFile = _
        "C:\MicroStation VBA\BatchProcessing\ProcessTheseLevels.txt"
    Open BatchFile For Input As #FFile
    While EOF(FFile) = False
        Line Input #FFile, strLIN
        Select Case Left(strLIN, 1)
        Case vbTab
            MsgBox "Do something to Level " & _
                                Replace(strLIN, vbTab, "")
        Case Else
            FileToOpen = strLIN
            Application.OpenDesignFile FileToOpen, True
        End Select
```

```
    Wend
End Sub
```

如果指定的文件都存在，则过程 ProcessASCIIC 运行良好。若要试图打开一个不存在的文件时会发生什么呢？



在试图打开任何文件前，都要确保该文件的存在。

```
Sub ProcessASCIID()
    Dim FileToOpen As String
    Dim BatchFile As String
    Dim strLIN As String
    Dim FFile As Long
    Dim FileIsOpen As Boolean
    FFile = FreeFile
    BatchFile =
            "C:\MicroStation VBA\BatchProcessing\ProcessTheseLevels.txt"
    Open BatchFile For Input As #FFile
    While EOF(FFile) = False
        Line Input #FFile, strLIN
        Select Case Left(strLIN, 1)
        Case vbTab
            If FileIsOpen = True Then
                MsgBox "Do something to Level " & _
                                Replace(strLIN, vbTab, "")
            End If
        Case Else
```

```
            FileToOpen = strLIN
            If Dir(FileToOpen) = "" Then
                FileIsOpen = False
            Else
                FileIsOpen = True
                Application.OpenDesignFile FileToOpen, True
            End If
        End Select
    Wend
End Sub
```

Dir 函数是一个标准的 VBA 函数，如果文件存在时它返回文件的名称；如果文件不存在，Dir 函数返回空串(“”)。

本例中如果文件不存在，就跳过它移到下一个文件。我们可以显示一个消息框来指明该文件不存在，但这样做会有问题。当批处理文件时，我们可能在周五下午 4:55 开始处理并希望它在周末一直运行。假如在处理到一个不存在的文件时显示一个出错对话框，那么这个消息框将一直从周五下午 5:05 直到下周一早上我们来到办公室还在显示着。

我们将在本章后续部分看到记录批处理事件。把不存在的文件记录下来会比显示一个消息框并等待用户输入要好得多。

处理一个文件夹下的所有文件

当用一个 ASCII 文件准确地说明要打开哪些文件时，一切都很清楚，我们知道文件的位置(或它们假设所在的位置)。然而，最经常要做的是打开指定文件夹下的每个设计文件。

取得一个文件夹下的所有文件名有几种方式。可以使用 VB 的 Dir 函数，但让我们尝试不同的方式。在我们的例子中将使用 Microsoft Scripting Runtime 引用。添加完该引用后下来代码就能执行了：

```
Sub ProcessInFolderA()
    Dim myFSO As New Scripting.FileSystemObject
    Dim myFolder As Scripting.Folder
```

```
    Dim myFile As Scripting.File
    Dim RootFolder As String
    RootFolder = "C:\MicroStation VBA\BatchProcessing"
    Set myFolder = myFSO.GetFolder(RootFolder)
    For Each myFile In myFolder.Files
        Select Case UCase(myFile.Type)
            Case "BENTLEY MICROSTATION DESIGN FILE"
                Debug.Print myFile.Path
        End Select
    Next
End Sub
```

RootFolder 中的每个文件都被检查了。本例查看文件的类型，文件类型与文件扩展名相关联。可以用 Windows 资源管理器来查看文件类型，查找设计文件的另一种方式是查看文件的扩展名。

```
Sub ProcessInFolderB()
    Dim myFSO As New Scripting.FileSystemObject
    Dim myFolder As Scripting.Folder
    Dim myFile As Scripting.File
    Dim RootFolder As String
    RootFolder = "C:\MicroStation VBA\BatchProcessing"
    Set myFolder = myFSO.GetFolder(RootFolder)
    For Each myFile In myFolder.Files
        Select Case UCase(Right(myFile.Name, 3))
            Case "DGN"
                Debug.Print myFile.Path
        End Select
    Next
End Sub
```

本例不是查看文件类型，而是查看文件名的最后三个字母。如果找到了 DGN

文件，就知道它是一个 MicroStation 设计文件并将其打印到立即(调试)窗口。

我们从好的开端继续进行。不再将 **RootFolder** 下的每个 DGN 文件的路径打印到调试窗口，我们要在 MicroStation 中打开文件并“处理”它。只要被处理的文件位于 **C:\MicroStation VBA\BatchProcessing** 文件夹下，过程 ProcessInFolderB 就能正常运行。当然，如果要处理的文件位于不同的文件夹时，我们并不想每次都修改变代码，能做到这一点吗？

```
Sub ProcessInFolderC()
    Dim myFSO As New Scripting.FileSystemObject
    Dim myFolder As Scripting.Folder
    Dim myFile As Scripting.File
    Dim RootFolder As String
    RootFolder = InputBox("Enter Root Folder:")
    Set myFolder = myFSO.GetFolder(RootFolder)
    For Each myFile In myFolder.Files
        Select Case UCase(Right(myFile.Name, 3))
            Case "DGN"
                Debug.Print myFile.Path
        End Select
    Next
End Sub
```

这是朝正确方向迈进的一步。我们使用 InputBox 让用户键入 **RootFolder** 路径。这虽然有些乏味但能摆脱在过程中硬编码路径。

让用户浏览并选择 **RootFolder** 该有多棒，但如何做到呢？当使用 VBA 时，随手就有大量的资源可用。现在就让我们利用这些资源中的一个。

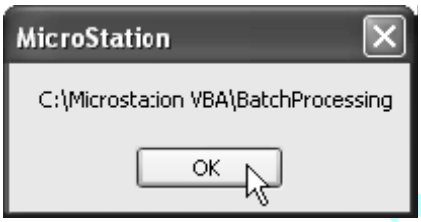
我们已经看过添加对 DLL 和库的引用能使我们使用强大的功能。我们要添加的引用叫做“Microsoft Shell Controls and Automation”。添加完这个引用后就能运行下一个例子了：

```
Sub PickAFolder()
    Dim myShell As New Shell32.Shell
    Dim myRootFolder As Shell32.Folder3
```


China BDN

```
    Set myRootFolder = myShell.BrowseForFolder(0, "Pick", 0)
    MsgBox myRootFolder.Self.Path
End Sub
```





它能再容易些吗？添加一个简单的引用和几行代码，然后用户就能选择计算机上的任何文件夹或网络上的任何位置。让我们把它与批处理代码放在一起工作。

```
Sub ProcessInFolderD()
    Dim myFSO As New Scripting.FileSystemObject
    Dim myFolder As Scripting.Folder
    Dim myFile As Scripting.File
    Dim myShell As New Shell32.Shell
    Dim myRootFolder As Shell32.Folder3
    Set myRootFolder = myShell.BrowseForFolder(0, "Pick", 0)
    Set myFolder = myFSO.GetFolder(myRootFolder.Self.Path)
    For Each myFile In myFolder.Files
        Select Case UCase(Right(myFile.Name, 3))
            Case "DGN"
```

```
                Debug.Print myFile.Path
            End Select
        Next
End Sub
```

现在不是让用户键入根文件夹了，取而代之的是用户选择文件夹。每个在选择文件夹下的 DGN 文件显示在立即窗口中。

处理一个文件夹及其子文件夹下的所有文件

在用户选择一个文件夹后，我们现在想处理被选文件夹下的所有文件以及其子文件夹下的所有文件。在弄清楚实现这个功能需要什么之前，需要讨论一个叫做“递归执行”的编程技术。

通常，一个过程或函数调用另一个过程或函数。然而，当一个函数或过程调用它自身时，它就是“递归”。让我们看一个使用了“Microsoft Scripting Runtime”和“Microsoft Shell Controls and Automation”引用的例子。

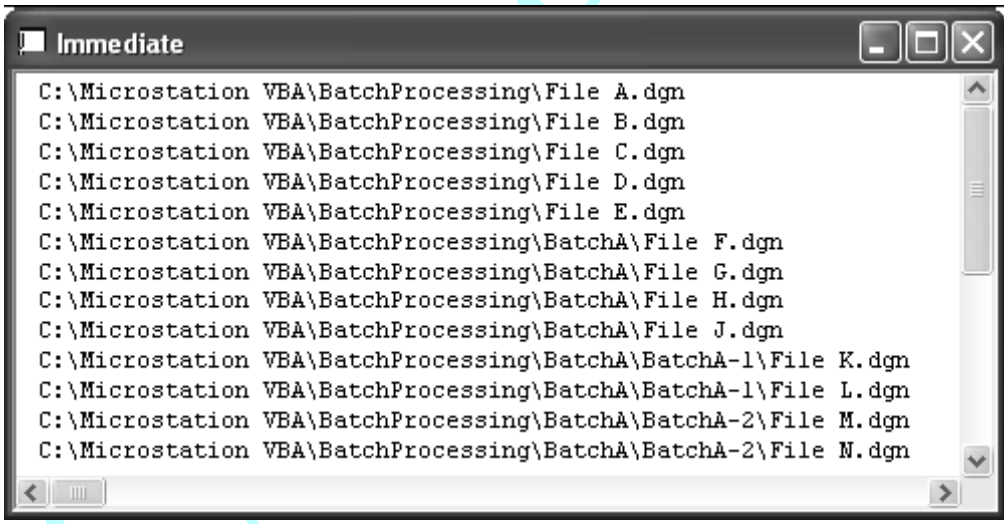
```
Sub ProcessInFolderE()
    Dim myFSO As New Scripting.FileSystemObject
    Dim myFolder As Scripting.Folder
    Dim myShell As New Shell32.Shell
    Dim myRootFolder As Shell32.Folder3
    Set myRootFolder = myShell.BrowseForFolder(0, "Pick", 0)
    Set myFolder = myFSO.GetFolder(myRootFolder.Self.Path)
    ProcessFilesAndSubs myFolder
End Sub
Sub ProcessFilesAndSubs(FolderIn As Scripting.Folder)
    Dim myFile As Scripting.File
    Dim mySubFolder As Scripting.Folder
    For Each myFile In FolderIn.Files
        Select Case UCase(Right(myFile.Name, 3))
            Case "DGN"
                Debug.Print myFile.Path
```

```
            End Select
        Next
        For Each mySubFolder In FolderIn.SubFolders
            ProcessFilesAndSubs mySubFolder
        Next
End Sub
```

ProcessInFolderE 看上去应该很熟悉。我们让用户选择一个文件夹，选择完后调用 ProcessFilesAndSubs 过程，变量 **myFolder** 供给 **FolderIn** 参数。

现在来看 ProcessFilesAndSubs。在声明了一些变量后检查“FolderIn”文件夹下的每个文件。如果文件名最右的三个字符是 DGN，就将文件路径打印到立即窗口(也叫做调试窗口)。在查找完所提供文件夹“FolderIn”下的每个文件后，开始查找其下的每个子文件夹。对于每个找到的子文件夹，我们把它作为 ProcessFilesAndSubs 的“FolderIn”参数，而我们目前就在这个过程之中。这就使得该过程成为递归，我们已经在这个过程中且让 VBA 再次开始执行该过程。这次，我们使用一个不同的文件夹作为“FolderIn”参数。

下面是这个过程的输出：



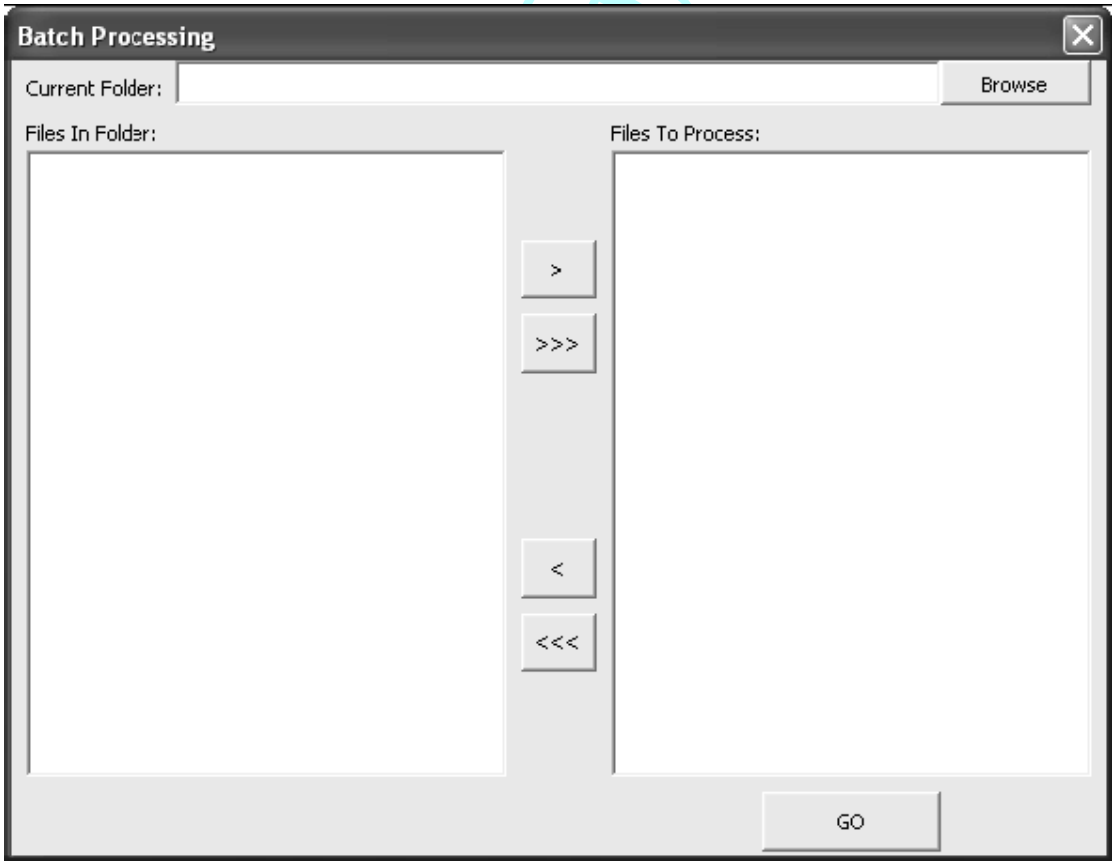
正如在这里看到的，我们从处理 C:\MicroStation VBA\BatchProcessing 中的文件开始。处理完这个文件夹后，开始查找其下的子文件夹，首先找到了“BatchA”。每当找到一个子文件夹时就查看其下的文件并查找其下的子文件夹。这听起来有些令人迷惑。您能看到在路径 C:\MicroStation VBA\BatchProcessing 下找到了两个子文件夹“BatchA-1”和“BatchA-2”。

以这种方式使用如上所示代码生成的结果是，每个子文件夹下的每个文件以及主文件夹下的每个文件都被识别到。在这些文件被识别后就能处理它们了。

建立选择文件的用户界面

我们知道了如何读取一个 ASCII 文件，也知道了如何处理指定文件夹(及其子文件夹)下的所有文件。假如只想处理一个文件夹下的指定文件又当如何做呢？我们可以对本章所示的过程加以修改，在处理每个文件前用一个消息框提问用户是否要处理。虽然这样做能给用户一点选择的机会，但用户不得不坐在电脑前不断地回答“是”、“是”、“否”、“是”、“否”、“否”。能选择吗？能。是批处理吗？不真正是。

我们将建立一个用户界面来让用户选择要处理的文件。文件选择完后，按 GO 按钮开始处理。GUI 中有两个列表框：一个显示当前选择的文件夹下的所有文件，另一个用于保存为批处理选择的文件。用户还能选择要从哪个文件夹下来选文件。两个列表框间有按钮能移动“选中的”或“所有的”文件，还能让用户双击文件实现该文件在两个列表框间的移动。



下面是界面背后的代码。

```
Private Sub btnBrowse_Click()
    Dim myFSO As New Scripting.FileSystemObject
    Dim myFolder As Scripting.Folder
    Dim myFile As Scripting.File
    Dim myShell As New Shell32.Shell
    Dim myRootFolder As Shell32.Folder3
    Set myRootFolder = myShell.BrowseForFolder(0, "Pick", 0)
    If myRootFolder Is Nothing Then Exit Sub
    Set myFolder = myFSO.GetFolder(myRootFolder.Self.Path)
    txtCurrentFolder.Text = myRootFolder.Self.Path
    lstFilesInFolder.Clear
    For Each myFile In myFolder.Files
        Select Case UCase(Right(myFile.Name, 3))
            Case "DGN"
                If IsFileIn(myFile.Path, lstFilesToProcess) _
                                = False Then
                    lstFilesInFolder.AddItem myFile.Path

                End If
        End Select
    Next
End Sub
Private Sub btnGo_Click()
    Dim I As Long
    For I = 1 To lstFilesToProcess.ListCount
        Debug.Print lstFilesToProcess.List(I - 1)
    Next I
End Sub
Private Sub btnIntoAll_Click()
    Dim I As Long
```

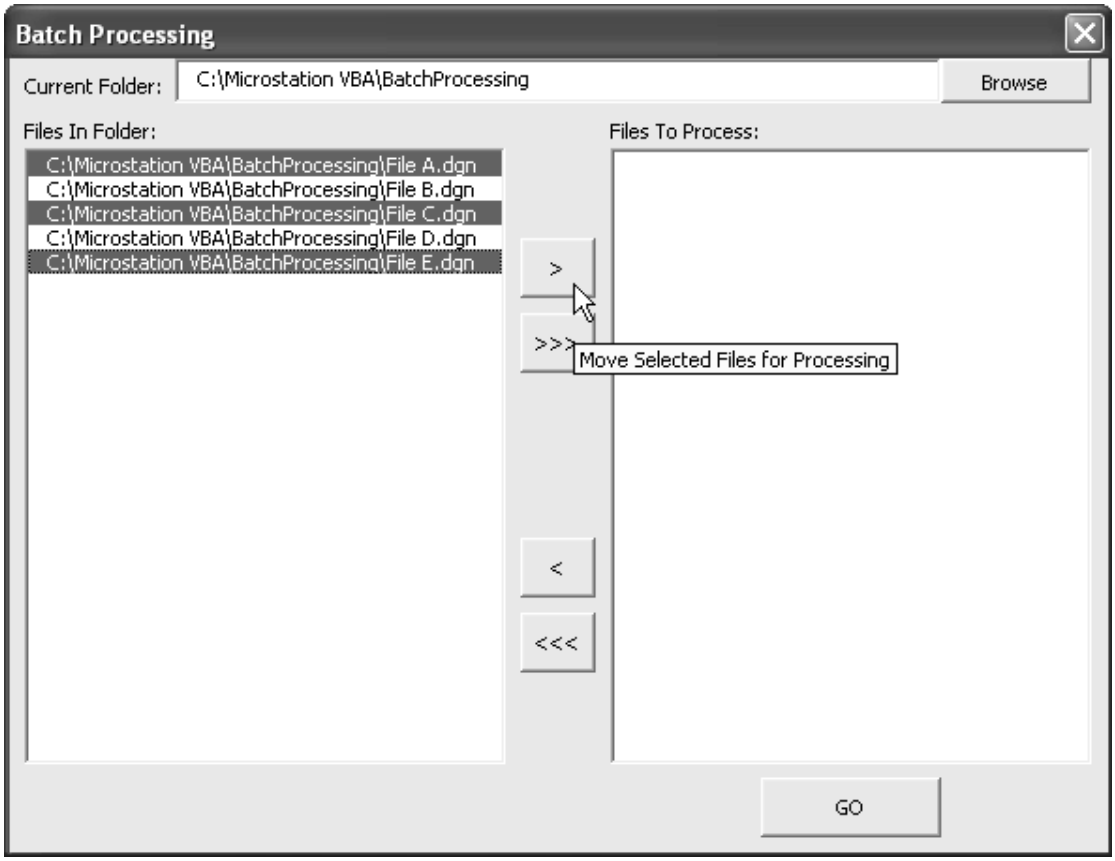
China BDN

```
    For I = 1 To lstFilesInFolder.ListCount
        lstFilesInFolder.Selected(I - 1) = True
    Next I
    MoveSelection lstFilesInFolder, lstFilesToProcess
End Sub
Private Sub btnOutAll_Click()
    Dim I As Long
    For I = 1 To lstFilesToProcess.ListCount
        lstFilesToProcess.Selected(I - 1) = True
    Next I
    MoveSelection lstFilesToProcess, lstFilesInFolder
End Sub
Private Sub btnIntoSelected_Click()
    MoveSelection lstFilesInFolder, lstFilesToProcess
End Sub
Private Sub btnOutSingle_Click()
    MoveSelection lstFilesToProcess, lstFilesInFolder
End Sub
Private Sub lstFilesInFolder_DblClick(ByVal Cancel As _
                        MSForms.ReturnBoolean)
    MoveSelection lstFilesInFolder, lstFilesToProcess
End Sub
Private Sub lstFilesToProcess_DblClick(ByVal Cancel As _
                        MSForms.ReturnBoolean)
    MoveSelection lstFilesToProcess, lstFilesInFolder
End Sub
Function IsFileIn(FilePath As String, ListToCheck As ListBox) As Boolean
    Dim I As Long
    IsFileIn = False
    For I = 1 To ListToCheck.ListCount
```


China BDN

```
            If StrComp(ListToCheck.List(I - 1), FilePath) = 0 Then
                IsFileIn = True
                Exit Function
            End If
        Next I
End Function
Function MoveSelection(ListBoxFrom As ListBox, ListBoxTo As ListBox)
        Dim I As Long
        For I = 1 To ListBoxFrom.ListCount
            If ListBoxFrom.Selected(I - 1) = True Then
                If IsFileIn(ListBoxFrom.List(I - 1), ListBoxTo) = False Then
                        ListBoxTo.AddItem ListBoxFrom.List(I - 1)
                End If
            End If
        Next I
        For I = ListBoxFrom.ListCount To 1 Step -1
            If ListBoxFrom.Selected(I - 1) = True Then
                ListBoxFrom.RemoveItem I - 1
            End If
        Next I
End Function
```

China BDN



有几项值得提醒：

- 设置列表框的 MultiSelect 属性为“Extended”，这样可以实现多选，所以乎还能用**<Ctrl>**和**<Shift>**键辅助选择列表项；
- 为了简化设计过程和减少潜在的错误，只用了一个过程“MoveSelection”在列表框间移动列表项。每个按钮动作都调用这个过程，即使列表框的双击动作也不例外；
- 点击 GO 按钮后只是简单地将待处理列表框中的文件打印到了立即窗口。读者可以插入任何处理文件的代码；
- 虽然当前工程中未包含该功能，但只要增加几行代码就能使用户通过选择一个复选框来指定“搜索子文件夹”；
- “Current Folder(当前文件夹)”文本框被锁定。这能让用户从该文本框选择和复制路径到 Windows 的剪切板，但用户不能在该文本框中键入内容。这有助于我们确认文本框中的路径是实际存在的，因为它是被用户选择到的而不是从键盘上键入的。

记录批文件处理过程

记录文件不是新东西。它告诉我们已经做了什么、什么还没有做、正在做什么以及将要做什么。

下面是一些能跟踪处理活动的途径：

- 使用记录文件。例如，一个带有“.log”文件扩展名的 ASCII 文件；
- 在数据库中跟踪处理活动；
- 在注册表中保存处理信息；
- 通过 Internet 记录活动；
- 用电子邮件将记录发送不同的收件人；

使用记录文件

使用记录文件或许是记录活动最容易的方法。使用记录文件的一些好处是：

- 使用记事本程序就能容易地查找和读取.log 文件
- .log 文件可以在本地计算机或网络上
- .log 文件可以用其它资源备份
- .log 文件不依赖于其它资源，如 Internet 连接
- 使用.log 文件能使批处理程序在断电、文件破坏等情况下“恢复”

在使用.log 文件时有几个问题需要回答。首先是文件位于什么位置？它可以位于像 C:\Log Files 这样的硬盘文件夹下；可以与.mvba 文件同一个路径；也可以位于像 Z:\Batch Processing\Log Files 这样的网络共享上。

把记录文件保存在与.mvba 文件相同的文件夹下很有用。如果我们正在运行一个宏，就知道.mvba 文件一定存在。如果这个.mvba 文件被保存，则它就具有一个实际的路径。如果我们能找到这个路径，就能将.log 文件写入这个路径并能容易地找到它。

```
Sub TestLogFile()
    WriteToLog "Opening C:\test.dgn"
End Sub
Sub WriteToLog(LogFileText As String)
    Dim myVBE As Object
```

```
    Dim xSplit As Variant
    Dim FFile As Long
    Set myVBE = Application.VBE
    xSplit = Split(myVBE.ActiveVBProject.Filename, "\")
    xSplit(UBound(xSplit)) = "BatchProcess.log"
    FFile = FreeFile
    Open Join(xSplit, "\") For Append As #FFile
    Print #FFile, LogFileText
    Close #FFile
End Sub
```

当有多个用户使用同一个.mvba 文件时情况又是什么呢？是的，记录文件将被附加，但如果记录文件已经被一个计算机或用户打开而另一个计算机又试图打开它时就会出问题。

让我们看下一个例子，它为每个计算机名建立唯一的一个.log 文件。这要用到一个叫做 GetComputerName 的 Windows API 函数。

```
‘General Declarations Area
Public Declare Function GetComputerName Lib "kernel32" Alias _
                "GetComputerNameA" (ByVal lpBuffer As String, _
                nSize As Long) As Long
Sub TestLogFileB()
    WriteToLogB "Opening C:\test.dgn"
End Sub
Sub WriteToLogB(LogFileText As String)
    Dim myVBE As Object
    Dim xSplit As Variant
    Dim FFile As Long
    Set myVBE = Application.VBE
    xSplit = Split(myVBE.ActiveVBProject.Filename, "\")
    xSplit(UBound(xSplit)) = "BatchProcess_" & ThisComputerName & ".log"
    FFile = FreeFile
```

```
    Open Join(xSplit, "\") For Append As #FFile
    Print #FFile, LogFileText
    Close #FFile
End Sub
Function ThisComputerName() As String
    ThisComputerName = Space(255)
    GetComputerName ThisComputerName, Len(ThisComputerName)
    ThisComputerName = Left(ThisComputerName, InStr(1, _
                                ThisComputerName, Chr(0)) - 1)
End Function
```

现在，每台运行该代码的计算机都有其自己的批处理.log 文件。我们也能基于登录计算机的“UserName(用户名)”来确定 log 文件，但这样做可能会出问题，因为我们可能会在多台计算机上用相同的“UserName”登录执行批处理。计算机名在网络上会随机器的不同而不同，它比登录到每台计算机的用户名会更有可能为唯一。

用数据库跟踪活动

用数据库跟踪活动非常有用。数据库易于查询信息，从数据库中能做报表，还能创建一个.asp 文件从网页浏览器运行以提供实时的批处理跟踪。使用数据库的好处不胜枚举。

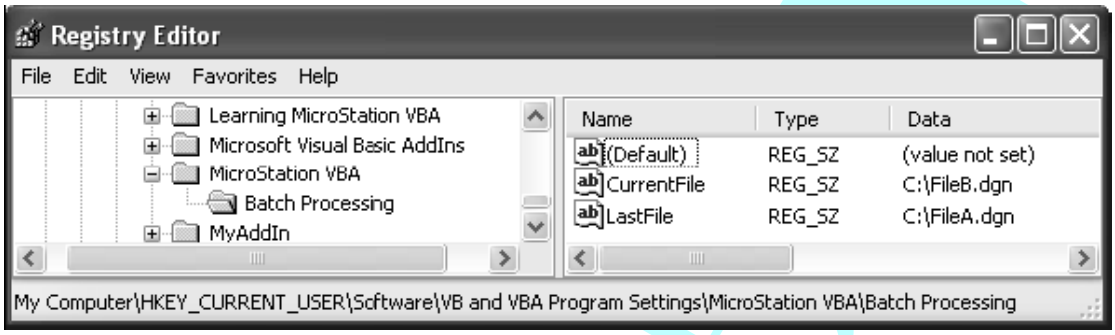
本书后续部分会专门讨论数据库，所以在此将不提供例子。当我们谈到数据库时请记住记录活动这回事。

在注册表中保存信息

Windows 注册表已经成为各种信息的通用存储媒介。我们所看到的“最近打开的文件”列表就是保存在注册表中。软件系列号、安装日期、安装位置和文件关联也是能在注册表中找到的众多内容中的一小部分。虽然我们不想添加大量信息到注册表中，但还是能添加一些诸如“当前文件”和“最近文件”这样的记录信息的。

```
Sub WriteToRegistry()
```

```
    SaveSetting "MicroStation VBA", "Batch Processing", _
                                    "LastFile", "C:\FileA.dgn"
    SaveSetting "MicroStation VBA", "Batch Processing", _
                                    "CurrentFile", "C:\FileB.dgn"
End Sub
```



写信息到注册表很简单，从注册表中读信息和写信息一样简单。

```
Sub ReadFromRegistry()
    Dim LastFile As String
    Dim CurrentFile As String
    LastFile = GetSetting("MicroStation VBA", _
                          "Batch Processing", "LastFile")
    CurrentFile = GetSetting("MicroStation VBA", _
                          "Batch Processing", "CurrentFile")
    Debug.Print LastFile
    Debug.Print CurrentFile
End Sub
```

通过 Internet 记录活动

Internet 用于各种活动。通常我们用它来获取信息，也能用它来传输信息。例如，如果打开网络浏览器到

http://www.google.com/search?q=MicroStation

Google 将返回成千上万与 MicroStation 相关的网页。

然而，如果输入

http://www.microstationlogging.com?filename=filea.dgn

这个假想的网站将记录文件、日期/时间、IP 地址等。但我们并不想让用户在每

次处理文件时键入 URL。对吗？让我们用软件来发送这种请求。

该是使用另一个引用的时候了。

“Microsoft Internet Controls”

```
Sub LogToWeb()
    Dim LocalFile As String
    Dim myInet As New InternetExplorer
    LocalFile = "filea.dwg"
    myInet.Navigate _
            "http://www.microstationlogging.com?filename=" & LocalFile
    While myInet.Busy
        DoEvents
    Wend
End Sub
```

这段代码只有几行。LogToWeb访问指定的URL，当前情况下就是这个假想的网站 www.microstationlogging.com。假如这是一个真实的网站，且假如该网站的缺省网页接受参数“filename”，那么该网站能将传送给它的文件名记录到Web服务器上的一个数据库中。

把信息记录到网站有其优点。其中之一就是，无论运行该程序的个人在世界上的哪个位置都没有关系，只要其计算机连通 Internet，就能记录其活动。Web服务器特为高流量、大容量情况而设计，所以即使有成百上千的人使用该网站记录其活动也不会出问题。另一些人可以点击网站来查看实时统计信息，如正在发生什么、谁正在工作、谁未在工作、每个文件的平均时间等。Internet 没有任何限制，只要在 VBA 中进行一个简单的引用然后访问特定的 URL 就能实现一切。

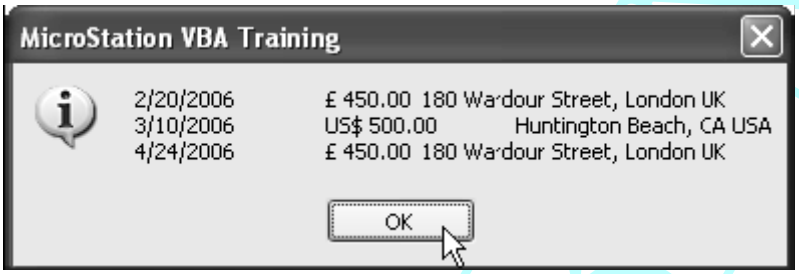
这样的技术能用于从一个网站获取信息。虽然这不是关于 Internet 技术的章节，但看一个这方面的小例子也没有什么坏处。

```
Sub GetFromWeb()
    Dim I As Long
    Dim LocalFile As String
    Dim myInet As New InternetExplorer
    Dim TheURL As String
```

```
Dim xSplitA As Variant
Dim xSplitB As Variant
Dim xSplitC As Variant
Dim xSplitD As Variant
Dim TrainingDate As Date
Dim TrainingLocation As String
Dim TrainingCost As String
Dim MsgBoxText As String
TheURL = "http://bentleyinstitute.bentley.com/" & _
         "courseinfo.aspx?course=TRC001810-1/0001"
myInet.Navigate TheURL
While myInet.ReadyState <> READYSTATE_COMPLETE
    DoEvents
Wend
xSplitA = Split(myInet.Document.body.innerHTML, "InPersonLabClasses")
xSplitB = Split(xSplitA(1), "<TR ")
For I = LBound(xSplitB) To UBound(xSplitB)
    xSplitC = Split(xSplitB(I), "<TD>")
    If IsArray(xSplitC) = True Then
        If UBound(xSplitC) >= 3 Then
            xSplitD = Split(xSplitC(1), "<")
            TrainingDate = CDate(Replace(xSplitD(0), ".", "/"))
            xSplitD = Split(xSplitC(2), "<")
            TrainingLocation = xSplitD(0)
            xSplitD = Split(xSplitC(3), "<")
            TrainingCost = xSplitD(0)
            MsgBoxText = MsgBoxText & TrainingDate & vbTab & _
            TrainingCost & vbTab & TrainingLocation & vbCr
        End If
    End If
```

```
    Next I
    MsgBox MsgBoxText, vbInformation, "MicroStation VBA Training"
End Sub
```

Bentley 学院会不定期地举办 VBA 培训，能在其网站上找到培训日期、地点和费用。该宏就是将这一信息显示在消息框中。

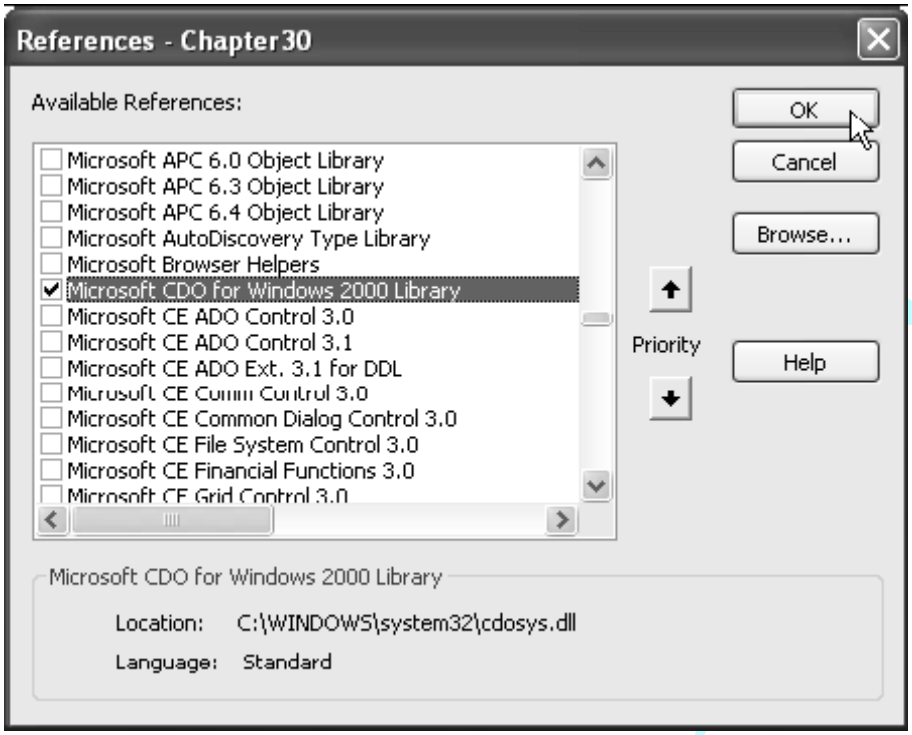


编写这本书时，Bentley 安排有三次 VBA 培训。其日期、费用和地点在消息框中显示。这段代码用于查找特定的 HTML 标签和注释来提取我们想要的信息。该页的 HTML 将来可能会改变，但这段代码表明，用 VBA 建立我们自己的信息收集工具是多么的容易。

电子邮件处理记录

电子邮件，它好像已经成为人们不可离开的东西了。电子邮件曾经只能在工作或家中的计算机上收发，现在它跟随我们在 PDA、手机和黑霉设备上。那么，VBA 能使用这一技术吗？当然能，只需要附加一个字符串。这个字符串当然是一个引用。

China-pub.com



如果该引用未在列表中显示，请点击 Browse(浏览)按钮，浏览到文件 C:\Windows\system32\cdosys.dll 也能实现该引用。

CDO 代表什么？协作数据对象(Collaborative Data Object)。用直白的语言来说就是电子邮件。

```
Sub TestEmail()
    Dim myMail As New CDO.Message
    myMail.To = "batch@microstationlogging.com"
    myMail.From = "batch@microstationlogging.com"
    myMail.Subject = "MicroStation VBA Batch Process Log"
    myMail.HTMLBody = "<b>File name: filea.dgn</b><br>" & _
                      "Computer: " & ThisComputerName & "<br>" & _
                      "Date: 1/1/2005"
    myMail.Configuration.Fields.Item("http://schemas.microsoft.com/" & _
                      "cdo/configuration/sendusing") = 2
    myMail.Configuration.Fields.Item("http://schemas.microsoft.com/" & _
                      "cdo/configuration/smtpserver") = _
                      "yoursmtpserver.com"
    myMail.Configuration.Fields.Item("http://schemas.microsoft.com/" & _
```

```
                "cdo/configuration/smtpserverport") = 25
        myMail.Configuration.Fields.Update
        myMail.Send
    End Sub
```

这段代码的难度大致与获得 CDO 引用一样。有了正确的引用和少许代码，我们的程序就能把批处理信息通过电子邮件发送到一个 E-Mail 地址。要使这段代码正确运行，“To”属性应当设置为一个合法的电子邮件地址，SMTPServer 字段应当设置为一个合法的 SMTP 服务器。

给这封电子邮件附加一个文件又该如何做呢？

```
Sub TestEmail2()
    Dim myMail As New CDO.Message
    myMail.To = "batch@microstationlogging.com"
    myMail.From = "batch@microstationlogging.com"
    myMail.Subject = "MicroStation VBA Batch Process Log"
    myMail.HTMLBody = "<b>File name: filea.dgn</b><br>" & _
                      "Computer: " & ThisComputerName & "<br>" & _
                      "Date: 1/1/2005"
    myMail.Configuration.Fields.Item("http://schemas.microsoft.com/" & _
                      "cdo/configuration/sendusing") = 2
    myMail.Configuration.Fields.Item("http://schemas.microsoft.com/" & _
                      "cdo/configuration/smtpserver") = _
                      "yoursmtpserver.com"
    myMail.Configuration.Fields.Item("http://schemas.microsoft.com/" & _
                      "cdo/configuration/smtpserverport") = 25
    myMail.Configuration.Fields.Update
    myMail.AddAttachment "C:\test.dgn"
    myMail.Send
End Sub
```

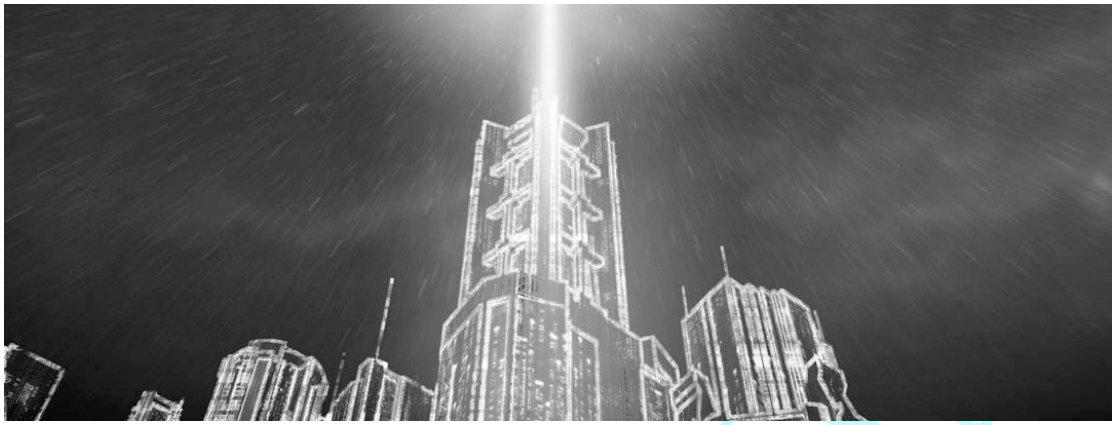
仅需一行代码。AddAttachment。就是它！

本章回顾

批处理任务完成起来相对容易。首先需要知道要处理哪些文件，这可以通过使用 ASCII 文件、让用户选择要处理的文件夹或在文件夹内选择独立的文件来找到。其次需要处理文件。接下来我们能通过各种方法记录批处理活动。

一旦批处理代码编制完毕，不论要处理 5 个文件或是 5000 个文件都没有太大的不同。相比较于手工一个个打开文件的工作方式，批处理的生产力和准确性成指数级增加。

China BDV



31. 标准检查器

标准检查器有助于保持文件与您所建立的标准的一致性。使用 MicroStation 安装时带的这个功能在执行基本标准检查时做得很好 — 再加上 VBA 的强大和灵活，什么也不能阻挡我们前进的步伐。

首先应当注意，发生在标准检查器程序中的未处理错误能造成严重错误。有些错误甚至能彻底关掉 MicroStation，所以编写标准检查代码时需要格外小心。确保经常保存 VBA 工程，在 MicroStation 中打开的任何 DGN 文件也要保存。

本章包括：

- 标准检查器的基本实现
- 标准检查器的设置
- 检查标准
- 标准检查报告
- 自动装载自定义标准检查器外接程序

标准检查器的基本实现

依靠 IStandardsChecker 接口来实现标准检查器。正如已经实现的其它接口一样，要使用一个类模块进事件中编写代码。

```
‘General Declarations
Implements IStandardsChecker
Private Sub IStandardsChecker_AddedCheckerToStandardsCheckerApps( _
                                ByVal ApplicationXMLNode As Object)
```

```
End Sub
Private Property Get IStandardsChecker_CallForEachModel() As Boolean
End Property
Private Sub IStandardsChecker_CreateSettings()
End Sub
Private Sub IStandardsChecker_DeleteSettings()
End Sub
Private Property Get IStandardsChecker_Description() As String
End Property
Private Property Get IStandardsChecker_DialogString() As String
End Property
Private Sub IStandardsChecker_EditSettings(ByVal IsReadOnly As Boolean)
End Sub
Private Property Get IStandardsChecker_FoundSettings() As Boolean
End Property
Private Sub IStandardsChecker_GetFixDetail(Fixes() As String, _
        ByVal SelectedFix As Long, FixPropertiesLabel As String, _
        FixProperties() As String)
End Sub
Private Property Get IStandardsChecker_HasSettings() As Boolean
End Property
Private Property Get IStandardsChecker_IdentityString() As String
End Property
Private Sub IStandardsChecker_RunCheck(ByVal _
                ModelToCheck As ModelReference, _
                ByVal FirstModel As Boolean, ByVal Options As Long)
End Sub
Private Property Get IStandardsChecker_VersionString() As String
End Property
```

China BDN

该接口实现了六个事件和七个属性。当用户选择 **Utilities > Standards Checker > Check(实用程序 > 标准检查器 > 检查)**开始进行标准检查时，类模块中的属性用在标准检查器的用户界面。让我们实现一个标准检查器，它除了最基本的属性以及用 Debug.Print 语句来看清事件发生顺序外什么也不做。

标准检查 A

下列代码位于类模块 **clsStandCheckA** 中。

```
Option Explicit
Implements IStandardsChecker
Private Sub IStandardsChecker_AddedCheckerToStandardsCheckerApps( _
                                        ByVal ApplicationXMLNode As Object)
    Debug.Print "AddedCheckerToStandardsCheckerApps"
End Sub
Private Property Get IStandardsChecker_CallForEachModel() As Boolean
    Debug.Print "CallForEachModel"
    IStandardsChecker_CallForEachModel = True
End Property
Private Sub IStandardsChecker_CreateSettings()
    Debug.Print "CreateSettings"
End Sub
Private Sub IStandardsChecker_DeleteSettings()
    Debug.Print "DeleteSettings"
End Sub
Private Property Get IStandardsChecker_Description() As String
    Debug.Print "Description"
    IStandardsChecker_Description = "VBA StandChk A Desc"
End Property
Private Property Get IStandardsChecker_DialogString() As String
    Debug.Print "DialogString"
    IStandardsChecker_DialogString = "VBA StandChk A Dial"
```

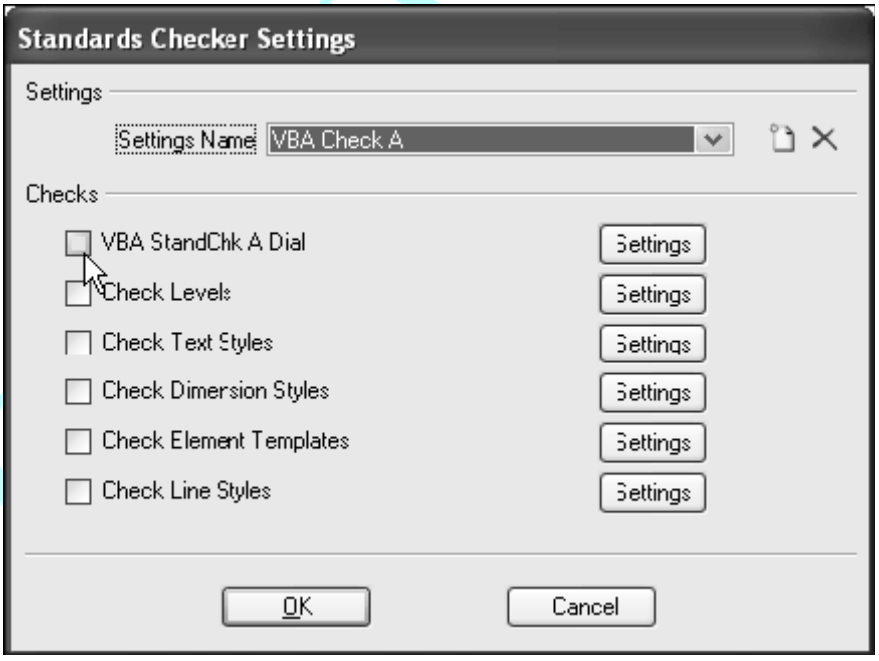
```
End Property
Private Sub IStandardsChecker_EditSettings(ByVal IsReadOnly As Boolean)
    Debug.Print "EditSettings"
End Sub
Private Property Get IStandardsChecker_FoundSettings() As Boolean
    Debug.Print "FoundSettings"
End Property
Private Sub IStandardsChecker_GetFixDetail(Fixes() As String, _
            ByVal SelectedFix As Long, FixPropertiesLabel As String, _
            FixProperties() As String)
    Debug.Print "GetFixDetail"
End Sub
Private Property Get IStandardsChecker_HasSettings() As Boolean
    Debug.Print "HasSettings"
    IStandardsChecker_HasSettings = True
End Property
Private Property Get IStandardsChecker_IdentityString() As String
    Debug.Print "IdentityString"
    IStandardsChecker_IdentityString = "VBA_SC_A"
End Property
Private Sub IStandardsChecker_RunCheck(ByVal _
                ModelToCheck As ModelReference, _
                ByVal FirstModel As Boolean, ByVal Options As Long)
    Debug.Print "IdentityString"
End Sub
Private Property Get IStandardsChecker_VersionString() As String
    Debug.Print "VersionString"
    IStandardsChecker_VersionString = "1.0.0.0"
End Property
```

China BDN

现在需要编写一些代码来把以上类模块添加到 MicroStation 的标准检查器中。这段代码位于代码模块中。

```
Option Explicit
Private StandChk As clsStandCheckA
Sub AddChecker()
    RemoveChecker
    Set StandChk = New clsStandCheckA
    StandardsCheckerController.AddStandardsChecker StandChk, 1000
End Sub
Sub RemoveChecker()
    If Not StandChk Is Nothing Then
        StandardsCheckerController.RemoveStandardsChecker StandChk
    End If
    Set StandChk = Nothing
End Sub
```

运行 AddChecker 时，类模块被实例化且添加到 MicroStation 的标准检查器对话框中。



当运行标准检查器时，能看到在类模块中返回的属性是如何被使用的。在以上图示中看到的“VBA StandChk A Dial”就来自于 DialogString 属性。在建立

类模块之前，标准检查器设置配置对话框中的“VBA Check A”设置被建立。

我们可以钩选“VBA StandChk A Dial”复选框并点击 Settings(设置)按钮。此时触发一些事件，相关联的 Debug.Print 语句被执行。

现在，我们能将自己的标准检查器植入系统的标准检查器对话框中，这是一个好的起点，但没什么值得详述的。在继续之前让我们讨论一下整个标准检查器的处理过程吧。

1、建立了一个类模块，它实现 IStandardsChecker 接口；

2、在一个过程中声明一个变量为该类模块类型，然后将用 AddStandardsChecker 添加该变量；

3、当用户点击 **Utilities > Standards Checker > Check(实用程序 > 标准检查器 > 检查)**时，按下列顺序触发事件和属性：

IdentityString

DialogString

HasSettings

CallForEachModel

FoundSettings

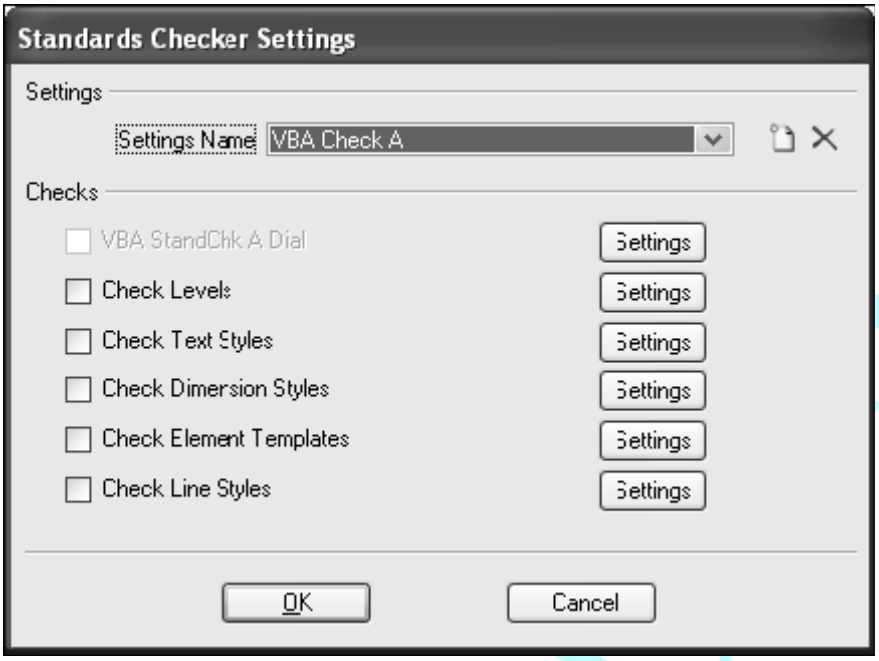
IdentityString

DialogString

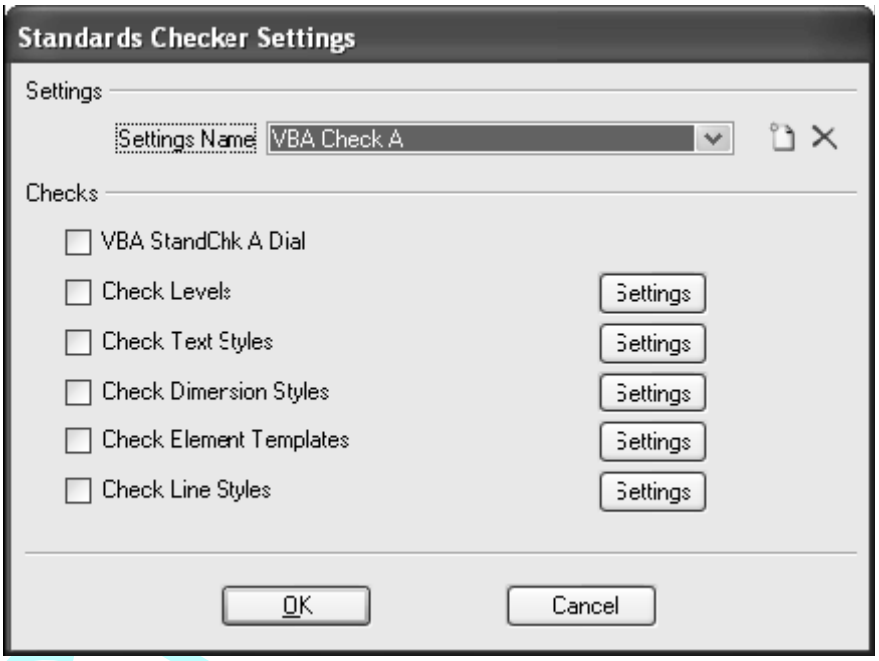
下一个看到的对话框会根据 HasSettings 属性值而变化。如果该属性返回一个 True 值，我们将看到一个设置对话框；

4、完成设置后，就能选择我们新建的自定义标准检查器外接程序了。

China BDN



然而，假如 HasSettings 属性返回一个 False 值，在标准检查设置中就不会出现一个 Settings(设置)按钮，我们建立的标准检查器被激活并能被用户选择；



5、如果 Settings 按钮可用，用户就点击该按钮；

6、EditSettings 事件被触发，带有值为 False 的“IsReadOnly”参数。该参数的 False 值表明，我们的标准检查器应该允许用户改变设置。此时，可以显示一个对话框来让用户选择针对我们的标准检查器的各种设置；

7、EditSettings 事件退出执行后，我们建立的标准检查器激活并可选用；

8、每当我们建立的标准检查器被选择或被取消选择，IdentityString 属性都

被检索；

9、当用户点击 OK 按钮时，我们建立的标准检查器被选择。

现在让我们实现一个设置对话框。要显示一个设置对话框，我们的标准检查器类模块中需要有如下代码：

```
Private Property Get IStandardsChecker_HasSettings() As Boolean
    Debug.Print "HasSettings"
    IStandardsChecker_HasSettings = True
End Property
Private Sub IStandardsChecker_EditSettings(ByVal IsReadOnly As Boolean)
    Debug.Print "EditSettings"
    UserForm1.Show
End Sub
```

当然了，这里的 Debug.Print 语句只是为了帮助我们看清事件触发的顺序，重要的是设置 IStandardsChecker_HasSettings 为 True 和在 EditSettings 事件中显示一个用户窗体。我们这里忽略了 EditSettings 事件的参数 IsReadOnly。

那么，我们让用户设置哪些设置呢？缺省情况下，MicroStation 具有对层、文本样式、标注样式、元素模板和线型的设置。还有什么要检查的呢？

标准检查器的设置

标准检查器接口能利用“设置”。

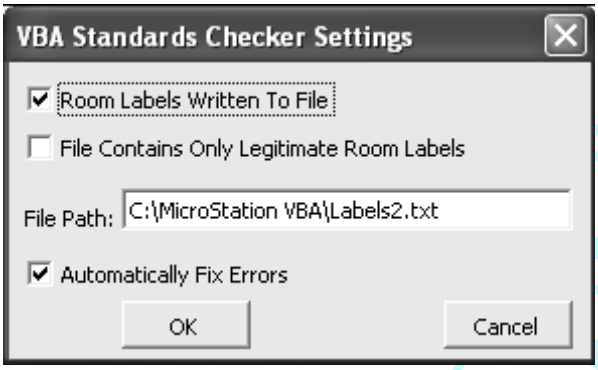
```
Private Property Get IStandardsChecker_HasSettings() As Boolean
    IStandardsChecker_HasSettings = True
End Property
```

如果 HasSettings 属性为 True，Settings(设置)按钮就会显示在标准检查器的设置对话框中(**Utilities > Standards Checker > Configure**)。当点击 Settings 按钮后，EditSettings 事件被触发。

```
Private Sub IStandardsChecker_EditSettings(ByVal IsReadOnly As Boolean)
    UserForm1.Show
End Sub
```

现在，当点击 Settings 按钮后显示一个叫做“UserForm1”的用户窗体。

让我们设计类似如下所示的一个图形用户界面：



这个例子允许输入和存储设计文件中与房间标记相关的设置。需要让用户选择如下属性：

- 房间标记写入一个文件中
- 文件仅包含合法的房间标记
- 文件路径
- 标准检查器遇到错误时修正吗？

界面相当简单和直接，输入数据是很容易实现的。设置将如何储存呢？有集中选择：

- 把设置保存在一个 ASCII 文本文件中；
- 把设置保存在设计文件中，这样设置对每个设计文件都是唯一的；
- 把设置保存在 Windows 注册表中；
- 把设置作为 XML 保存在设置 dgnlib 文件中。

我们将分析一种保存标准检查器设置的方法。这里是将值保存在 Windows 注册表中，可以在 MicroStation VBA 帮助文件中找到另一种保存标准检查器设置的方法(在一个.dgnlib 文件中以 XML 格式储存)。

本例在 Windows 注册表中保存设置。下面是 GUI 背后的代码：

```
Private Sub UserForm_Initialize()
    chkLabelsWritten.Value = GetSetting("VBA Standards Checker", _
                                        "Settings", "Labels Written", False)
    chkCleanFile.Value = GetSetting("VBA Standards Checker", _
                                        "Settings", "Clean File", False)
    chkFixErrors.Value = GetSetting("VBA Standards Checker", _
                                        "Settings", "Fix Errors", False)
```

```
    txtFilePath.Text = GetSetting("VBA Standards Checker", _
                "Settings", "File Path", "C:\MicroStation VBA\Labels.txt")
End Sub
Private Sub btnOK_Click()
    If txtFilePath.Text = "" Then
        MsgBox "A file path must be entered."
        Exit Sub
    End If
    SaveSetting "VBA Standards Checker", "Settings", _
                "Labels Written", chkLabelsWritten.Value
    SaveSetting "VBA Standards Checker", "Settings", _
                "Clean File", chkCleanFile.Value
    SaveSetting "VBA Standards Checker", "Settings", _
                "Fix Errors", chkFixErrors.Value
    SaveSetting "VBA Standards Checker", "Settings", _
                "File Path", txtFilePath.Text
    Unload Me
End Sub
Private Sub btnCancel_Click()
    Unload Me
End Sub
```

设置窗体初始化时，我们用标准 VBA 调用 GetSetting 查看 Windows 注册表并取得缺省值。缺省值很重要，因为设置窗体首次显示时注册表中没有所需的设置信息。

用户点击 OK 按钮后，我们用 SaveSetting 调用将复选框和文本框的属性保存到注册表中。这些值保存到注册表后，卸载该窗体。

如果点击了 Cancel(取消)按钮，我们卸载窗体而不保存值到注册表中。

检查标准

到目前为止，我们能添加自定义标准检查器显示到系统的标准检查器对话框

中。当用户点击标准检查器设置对话框中的 Settings 按钮时还能显示和保存所做的设置。

那么，如何实际检查文档呢？首先需要一个待检查的文件，让我们打开文件**BSI300AE101-Plan.dgn**。该文件随 MicroStation 一起安装。

现在即将开始检查该文件。

我们将要分析一个过程，它检查、报告和修复(如果设置这么做的话)文件**BSI300AE101-Plan.dgn** 中与房间标记相关的情形。该过程叫做 RoomLabelChecks，分为七个段落：

- 声明变量
- 读取图中所有房间标记的标签
- 从注册表中取得设置
- 读取房间标记文件
- 检查标准文件是否存在
- 检查图形文件中的标签
- 检查以保证标准文件中的标记具有 DGN 文件中的标签

该过程的每一段都以一个以上公布的项目作为注释开头。我们在 IStandardsChecker 接口的 RunCheck 事件中检查文件。

```
Private Sub IStandardsChecker_RunCheck(ByVal _
                    ModelToCheck As ModelReference, _
                    ByVal FirstModel As Boolean, ByVal Options As Long)
    RoomLabelChecks ModelToCheck
End Sub
```

下面是过程 RoomLabelChecks：

```
Sub RoomLabelChecks(ModelToScan As ModelReference)
    '----------------------Declare Variables
    Dim myElementEnum As ElementEnumerator
    Dim myScanCriteria As New ElementScanCriteria
    Dim myTags() As String
    ReDim myTags(0) As String
    Dim myTagsFilter() As String
```

```
Dim RoomTag As TagElement
Dim I As Long
Dim AutoFix As Boolean
Dim LabelsInFile As Boolean
Dim LegitimateLabels As Boolean
Dim myFileRooms() As String
ReDim myFileRooms(0) As String
Dim FFile As Long
Dim FileName As String
Dim LineIn As String
'-----------------------Read All Room Label Tags in Drawing
myScanCriteria.ExcludeAllTypes
myScanCriteria.IncludeType msdElementTypeTag
Set myElementEnum = ModelToScan.Scan(myScanCriteria)
While myElementEnum.MoveNext
    Set RoomTag = myElementEnum.Current
    Select Case UCase(RoomTag.TagDefinitionName)
    Case "ROOM LABEL"
        myTagsFilter = Filter(myTags, RoomTag.Value, True, _
                              vbBinaryCompare)
        If UBound(myTagsFilter) = -1 Then
            myTags(UBound(myTags)) = RoomTag.Value
            ReDim Preserve myTags(UBound(myTags) + 1)
        End If
    End Select
Wend
'*********
If UBound(myTags) >= 1 Then
    ReDim Preserve myTags(UBound(myTags) - 1)
End If
```

China BDN

```
'-----------------------Get Settings from Registry
FileName = GetSetting("VBA Standards Checker", "Settings", _
                      "File Path", "C:\MicroStation VBA\Labels.txt")
AutoFix = GetSetting("VBA Standards Checker", "Settings", _
                      "Fix Errors", "False")
LabelsInFile = GetSetting("VBA Standards Checker", _
                      "Settings","Labels Written", "False")
LegitimateLabels = GetSetting("VBA Standards Checker", _
                      "Settings", "Clean File", "False")
'-----------------------Read Room Label File
FFile = FreeFile
If Dir(FileName) <> "" Then
    Open FileName For Input As #FFile
    While EOF(FFile) = False
        Line Input #FFile, myFileRooms(UBound(myFileRooms))
        ReDim Preserve myFileRooms(UBound(myFileRooms) + 1)
    Wend
    If UBound(myFileRooms) >= 1 Then
        ReDim Preserve myFileRooms(UBound(myFileRooms) - 1)
    End If
    Close #FFile
End If
'----------------------Check for File's Existence
If Dir(FileName) = "" Then
    Debug.Print "The File " & FileName & " does not exist."
    If AutoFix = True Then
        Open FileName For Output As #FFile
        For I = LBound(myTags) To UBound(myTags)
            Print #FFile, myTags(I)
        Next I
```

```
            Close #FFile
            Debug.Print "The File " & FileName & " has been created."
        End If
    End If
    '-----------------------Check for Tags in File
    If LabelsInFile = True Then
        For I = LBound(myTags) To UBound(myTags)
            myTagsFilter = Filter(myFileRooms, myTags(I), True, _
                                  vbBinaryCompare)
            If UBound(myTagsFilter) = -1 Then
                Debug.Print "Tag " & myTags(I) & " not found in file."
                If AutoFix = True Then
                    Open FileName For Append As #FFile
                    Print #FFile, myTags(I)
                    Debug.Print "Label " & myTags(I) & " added to File."
                    Close #FFile
                End If
            End If
        Next I
    End If
    '-----------------------Check to make sure Labels in the File have
    'Tags in the DGN file
    If LegitimateLabels = True Then
        If Dir(FileName) = "" Then
            Debug.Print "File " & FileName & " does not exist."
        Else
            For I = LBound(myFileRooms) To UBound(myFileRooms)
                myTagsFilter = Filter(myTags, myFileRooms(I), _
                                      True, vbBinaryCompare)
                If UBound(myTagsFilter) = -1 Then
```


China BDN

```
                    Debug.Print "File Room " & _
                                myFileRooms(I) & " is not in the DGN File."
                End If
            Next I
        End If
    End If
End Sub
```

该过程有点长，所以让我们分段来讨论它。

1、声明变量

这一部分是自解释的。使用了几个动态数组，因为在代码执行前不知道在图中或文件中能找到多少个房间标记。

2、读取图中所有房间标记的标签

我们建立了对标签的扫描准则过滤，然后查看图中的哪些标签是“房间标记”。找到一个后就把它赋给动态数组 **myTags** 的最后一个元素，然后对数组大小加 1，增加数组大小时保留已经存在的值。

3、从注册表中取得设置

把标准检查器设置放入变量中，这样做只需一次读取，随后的代码读取这些变量时就比每次用 GetSetting 调用从注册表中读取要容易。

4、读取房间标记文件

我们在设置对话框中指定了一个文件名，该文件包含我们的 DGN 文件中所有房间标记的值。如果存在该文件，就读取它并把文件的每一行放入动态数组的一个元素中。

5、检查文件是否存在

如果放房间标记值的文件不存在，而且“Automatically Fix Errors(自动修复错误)”设置为 True 时，我们就建立该文件并写内容到其中。

6、检查文件中的标签

我们查看图形文件中找到的每个房间标记的标签，在标准文件中检查它是否存在。如果房间标记标签在标准文件中未找到，就把它加入到标准文件中(如果 AutoFix 为 True 的话)并在调试(立即)窗口中包括错误。

7、检查以保证标准文件中具有 DGN 文件的标签

我们查看的标准文件中也可能存在图形文件中没有的房间标记值。这可能是由于数据输入错误，或者是从图形文件中删除房间标记造成的结果。不管是哪种情况，只要在 ASCII 文件中发现一个房间标记在图形文件中没有相关联的房间标记值，就把问题报告到立即窗口中。

标准检查报告

现在让我们使用一些内置错误报告功能来扩展标准检查器。我们在类模块的通用声明区添加一个自定义的枚举，在类模块体中增加一个过程。所有改变的代码以黑体显示。

使用枚举“ErrorType”来帮助描述所报告的错误种类。正如您在此看到的，枚举中有七个元素。

```
Private Enum ErrorType
    NoError = 0
    NotInFile = 1
    NotInDGN = 2
    FixedNotInFile = 11
    FixedNotInDGN = 12
    NotFixedNotInFile = 21
    NotFixedNotInDGN = 22
End Enum
```

当发现要报告的东西时使用过程 ReportError。调用该过程时指定遇到的错误类型以及未找到的、校正的或未校正的房间号。

```
Private Sub ReportError(Error As ErrorType, RoomNumber As String)
    Dim mySCC As StandardsCheckerController
    Set mySCC = StandardsCheckerController
    Select Case Error
    Case ErrorType.NotInDGN
        mySCC.TotalProblems = mySCC.TotalProblems + 1
    Case ErrorType.NotInFile
```

```
            mySCC.TotalProblems = mySCC.TotalProblems + 1
        Case ErrorType.FixedNotInDGN
            mySCC.FixedProblems = mySCC.FixedProblems + 1
        Case ErrorType.FixedNotInFile
            mySCC.FixedProblems = mySCC.FixedProblems + 1
        Case ErrorType.NotFixedNotInDGN
            mySCC.IgnoredProblems = mySCC.IgnoredProblems + 1
        Case ErrorType.NotFixedNotInFile
            mySCC.IgnoredProblems = mySCC.IgnoredProblems + 1
        End Select
End Sub
```

我们在 RoomLabelChecks 中所做的唯一不同是，遇到错误或修故错误时使用 ReportError 过程。

```
Sub RoomLabelChecks(ModelToScan As ModelReference)
    '-----------------------Declare Variables
    Dim myElementEnum As ElementEnumerator
    Dim myScanCriteria As New ElementScanCriteria
    Dim myTags() As String
    ReDim myTags(0) As String
    Dim myTagsFilter() As String
    Dim RoomTag As TagElement
    Dim I As Long
    Dim AutoFix As Boolean
    Dim LabelsInFile As Boolean
    Dim LegitimateLabels As Boolean
    Dim myFileRooms() As String
    ReDim myFileRooms(0) As String
    Dim FFile As Long
    Dim FileName As String
    Dim LineIn As String
```

```
'-----------------------Read All Room Label Tags in Drawing
myScanCriteria.ExcludeAllTypes
myScanCriteria.IncludeType msdElementTypeTag
Set myElementEnum = ModelToScan.Scan(myScanCriteria)
While myElementEnum.MoveNext
    Set RoomTag = myElementEnum.Current
    Select Case UCase(RoomTag.TagDefinitionName)
    Case "ROOM LABEL"
        myTagsFilter = Filter(myTags, RoomTag.Value, True, _
                              vbBinaryCompare)
        If UBound(myTagsFilter) = -1 Then
            myTags(UBound(myTags)) = RoomTag.Value
            ReDim Preserve myTags(UBound(myTags) + 1)
        End If
    End Select
Wend
'*********
If UBound(myTags) >= 1 Then
    ReDim Preserve myTags(UBound(myTags) - 1)
End If
'-----------------------Get Settings from Registry
FileName = GetSetting("VBA Standards Checker", "Settings", _
                      "File Path", "C:\MicroStation VBA\Labels.txt")
AutoFix = GetSetting("VBA Standards Checker", "Settings", _
                      "Fix Errors", "False")
LabelsInFile = GetSetting("VBA Standards Checker", _
                      "Settings","Labels Written", "False")
LegitimateLabels = GetSetting("VBA Standards Checker", _
                      "Settings", "Clean File", "False")
'-----------------------Read Room Label File
```


China BDN

```
FFile = FreeFile
If Dir(FileName) <> "" Then
    Open FileName For Input As #FFile
    While EOF(FFile) = False
        Line Input #FFile, myFileRooms(UBound(myFileRooms))
        ReDim Preserve myFileRooms(UBound(myFileRooms) + 1)
    Wend
    If UBound(myFileRooms) >= 1 Then
        ReDim Preserve myFileRooms(UBound(myFileRooms) - 1)
    End If
    Close #FFile
End If
'-----------------------Check for File's Existence
If Dir(FileName) = "" Then
    Debug.Print "The File " & FileName & " does not exist."
    If AutoFix = True Then
        Open FileName For Output As #FFile
        For I = LBound(myTags) To UBound(myTags)
            Print #FFile, myTags(I)
        Next I
        Close #FFile
        Debug.Print "The File " & FileName & " has been created."
    End If
End If
'----------------------Check for Tags in File
If LabelsInFile = True Then
    For I = LBound(myTags) To UBound(myTags)
        myTagsFilter = Filter(myFileRooms, myTags(I), True, _
                              vbBinaryCompare)
        If UBound(myTagsFilter) = -1 Then
```


China BDN

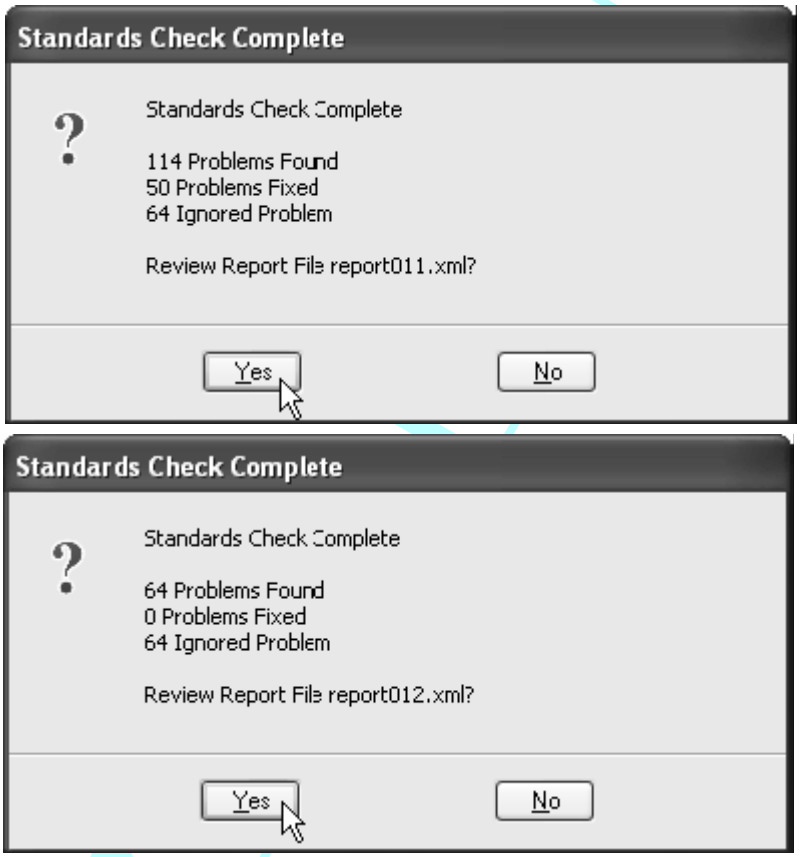
```
                ReportError NotInFile, myTags(I)
                Debug.Print "Tag " & myTags(I) & " not found in file."
                If AutoFix = True Then
                    ReportError FixedNotInFile, myTags(I)
                    Open FileName For Append As #FFile
                    Print #FFile, myTags(I)
                    Debug.Print "Label " & myTags(I) & " added to File."
                    Close #FFile
                Else
                    ReportError NotFixedNotInFile, myTags(I)
                End If
            End If
        Next I
End If
'-----------------------Check to make sure Labels in the File have
'Tags in the DGN file
If LegitimateLabels = True Then
    If Dir(FileName) = "" Then
        Debug.Print "File " & FileName & " does not exist."
    Else
        For I = LBound(myFileRooms) To UBound(myFileRooms)
            myTagsFilter = Filter(myTags, myFileRooms(I), _
                                  True, vbBinaryCompare)
            If UBound(myTagsFilter) = -1 Then
                ReportError NotInDGN, myFileRooms(I)
                ReportError NotFixedNotInDGN, myFileRooms(I)
                Debug.Print "File Room " & _
                      myFileRooms(I) & " is not in the DGN File."
            End If
        Next I
```


China BDN

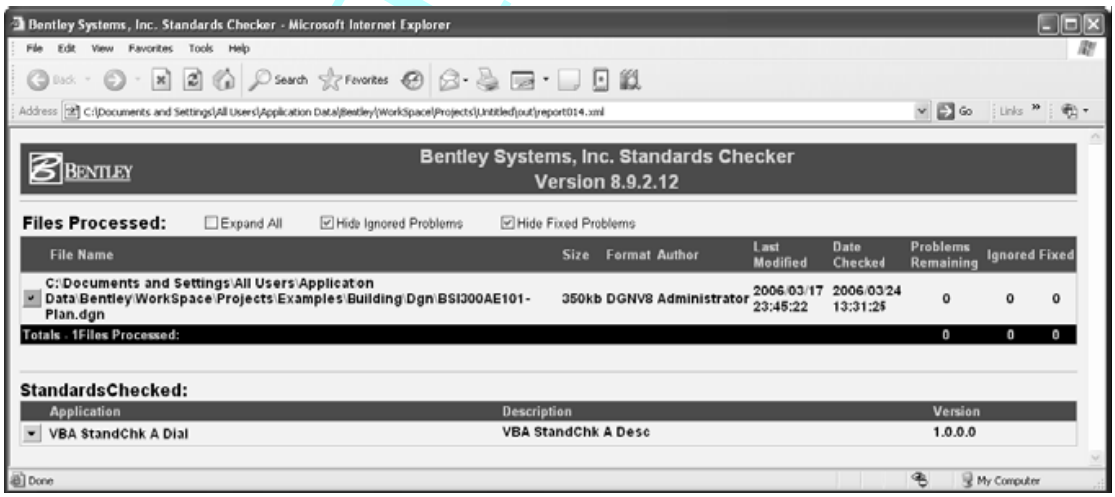
```
        End If
    End If
End Sub
```

我们已经添加了一些代码行，获得了什么呢？

运行完标准检查器后，显示如右图：



到目前为止，在我们的标准检查器“外接程序”开发中没有向报告文件报告任何东西。如果在标准检查器完成对话框中点击 Yes，将看到一个报告。



我们只是触及了在标准检查器程序中发现的问题报告的表面。让我们在

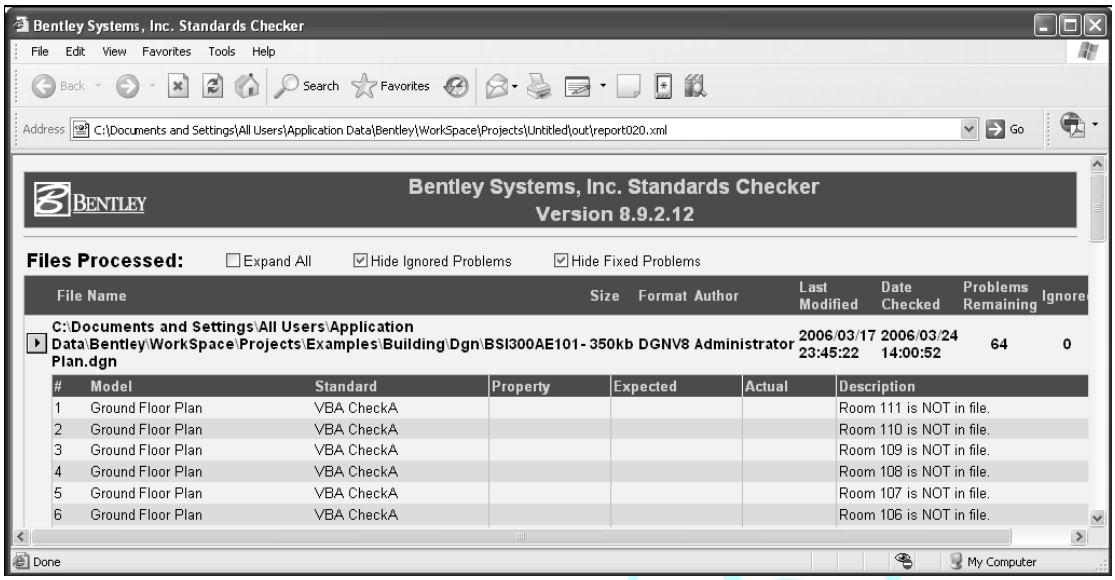
ReportError 过程中添加一些代码来扩展它。

```
Private Sub ReportError(Error As ErrorType, RoomNumber As String)
    Dim mySCC As StandardsCheckerController
    Dim myProb As StandardsCheckerProblem
    Dim myRep As StandardsCheckerReport
    Set mySCC = StandardsCheckerController
    Set myRep = mySCC.Report
    Select Case Error
    Case ErrorType.NotInDGN
        mySCC.TotalProblems = mySCC.TotalProblems + 1
        Set myProb = myRep.AddProblem("Room " & RoomNumber & _
                                      " is NOT in DGN.", "VBA CheckA", False)
    Case ErrorType.NotInFile
        mySCC.TotalProblems = mySCC.TotalProblems + 1
        Set myProb = myRep.AddProblem("Room " & RoomNumber & _
                                      " is NOT in file.", "VBA CheckA", False)
    Case ErrorType.FixedNotInDGN
        mySCC.FixedProblems = mySCC.FixedProblems + 1
    Case ErrorType.FixedNotInFile
        mySCC.FixedProblems = mySCC.FixedProblems + 1
    Case ErrorType.NotFixedNotInDGN
        mySCC.IgnoredProblems = mySCC.IgnoredProblems + 1
    Case ErrorType.NotFixedNotInFile
        mySCC.IgnoredProblems = mySCC.IgnoredProblems + 1
    End Select
End Sub
```

现在来看检查完成后生成的报告，从报告中能看出所指出的几个问题。

如果我们扩展文件名下的列表，将能看到以上添加的几个问题。

China BDN



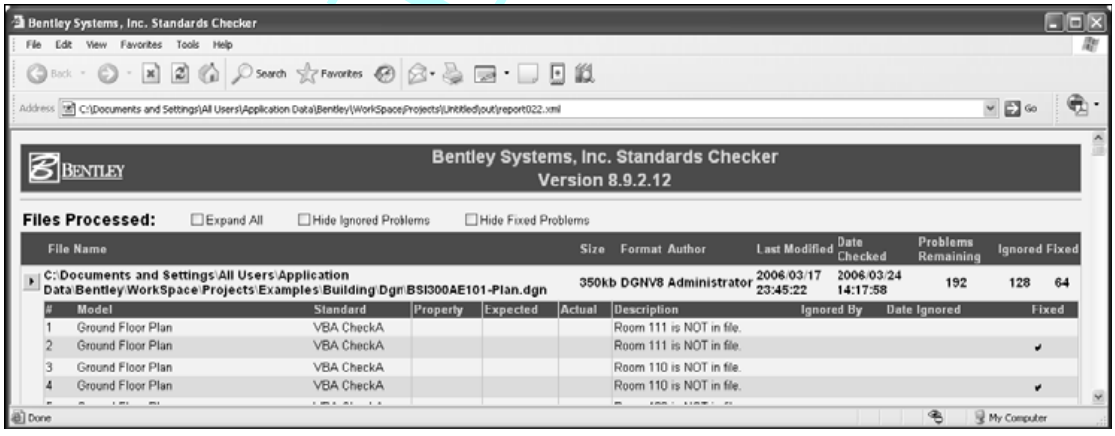
让我们在 ReportError 过程中再添加点内容。

```
Private Sub ReportError(Error As ErrorType, RoomNumber As String)
    Dim mySCC As StandardsCheckerController
    Dim myProb As StandardsCheckerProblem
    Dim myRep As StandardsCheckerReport
    Set mySCC = StandardsCheckerController
    Set myRep = mySCC.Report
    Select Case Error
    Case ErrorType.NotInDGN
        mySCC.TotalProblems = mySCC.TotalProblems + 1
        Set myProb = myRep.AddProblem("Room " & RoomNumber & _
                    " is NOT in DGN.", "VBA CheckA", False)
    Case ErrorType.NotInFile
        mySCC.TotalProblems = mySCC.TotalProblems + 1
        Set myProb = myRep.AddProblem("Room " & RoomNumber & _
                    " is NOT in file.", "VBA CheckA", False)
    Case ErrorType.FixedNotInDGN
        mySCC.FixedProblems = mySCC.FixedProblems + 1
        Set myProb = myRep.AddProblem("Room " & RoomNumber & _
                    " is NOT in DGN.", "VBA CheckA", True)
```

```
        Case ErrorType.FixedNotInFile
            mySCC.FixedProblems = mySCC.FixedProblems + 1
            Set myProb = myRep.AddProblem("Room " & RoomNumber & _
                    " is NOT in file.", "VBA CheckA", True)
        Case ErrorType.NotFixedNotInDGN
            Set myProb = myRep.AddIgnoredProblem("Room " & _
                    RoomNumber & _
                    " is NOT in DGN.", "VBA CheckA", UserName, Now)
            mySCC.IgnoredProblems = mySCC.IgnoredProblems + 1
        Case ErrorType.NotFixedNotInFile
            Set myProb = myRep.AddIgnoredProblem("Room " & _
                    RoomNumber & _
                    " is NOT in file.", "VBA CheckA", UserName, Now)
            mySCC.IgnoredProblems = mySCC.IgnoredProblems + 1
        End Select
End Sub
```

现在，除了向报告中添加一个问题来表明该问题未被修复(在 AddProblem 中的参数 False)，我们还添加了修复的文件(Fixed Problem)和忽略的问题(Ignored Problem)。

下面是现在的报告：



现在，除了看到未被修复的问题外，还看到了被修复的问题和被忽略的问题。

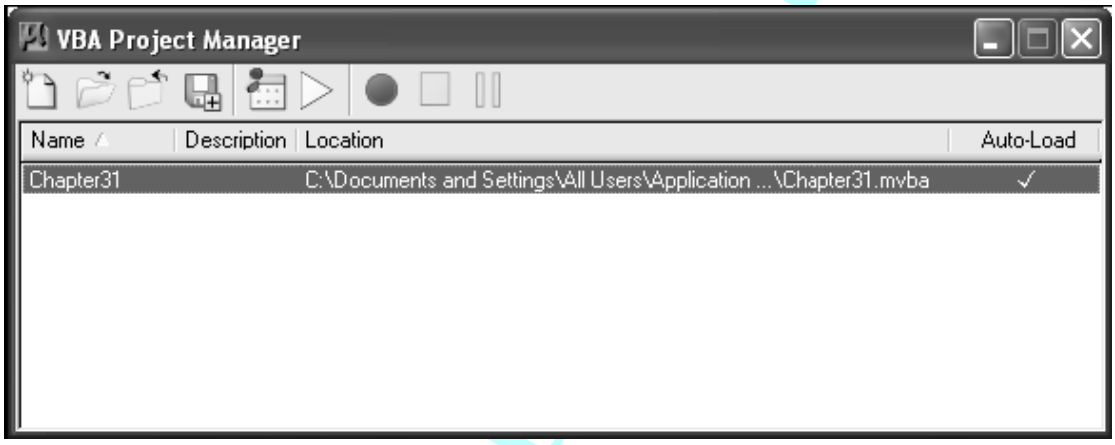
自动装载自定义标准检查器外接程序

标准检查器功能可通过两种机制来自动装载。两种机制都使用了 OnProjectLoad 过程，该过程可放置在一个 MicroStation VBA 工程中。当打开一个 VBA 工程(.mvba 文件)时，OnProjectLoad 过程被执行。该过程中的代码对于两种自动装载机制都是相同的：

```
Sub OnProjectLoad()
    AddChecker
End Sub
```

OnProjectLoad 过程对于两种自动装载机制都是相同的。这两种机制是什么？

1、第一种机制是在 VBA 工程管理器中选择“Auto-Load(自动装载)”：

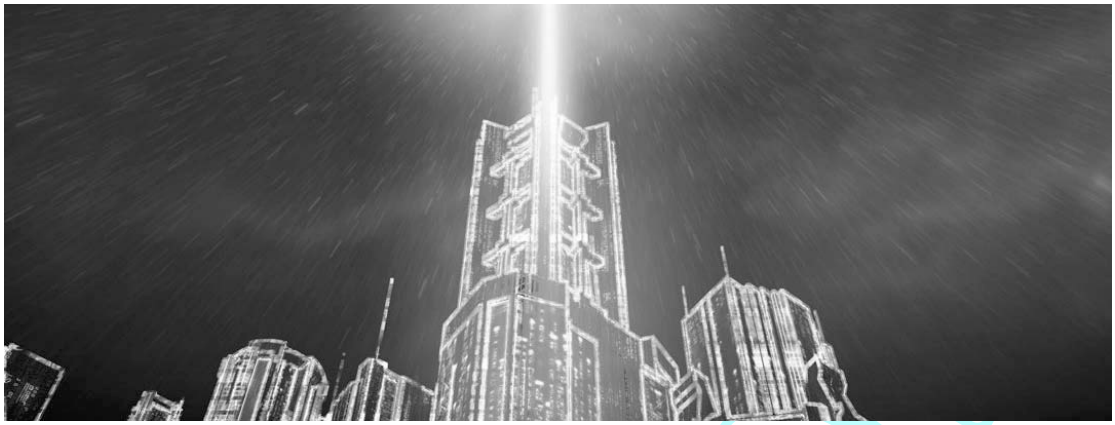


当一个 VBA 工程被设为“Auto-Load”后，该工程在 MicroStation 启动时被装载。工程装载后，OnProjectLoad 被执行，我们建立的标准检查器外接程序也被装载。

2、另一种机制涉及配置变量 MS_STANDARDCHECKERAPPS。如果将某个 VBA 工程文件(.mvba 文件)添加到这个变量中，当标准检查器初始化时该 VBA 工程将被装载。这种方法比上面描述的那种要好，因为标准检查器代码并不被装载和添加，除非我们决定用标准检查器做某些事情。

本章回顾

标准检查器接口能使我们建立自定义的标准检查器程序。MicroStation VBA 文档包含更多的进一步实现 IStandardsChecker 接口的例子。



32. 使用 Windows API

我们添加过的引用有“Microsoft Scripting Runtime”、“Microsoft ActiveX Data Objects”、“Microsoft CDO for Windows 2000”等。当这样做时，就能立刻访问本不属于 VBA 的功能。

虽然使用 Windows API 不像添加引用那样简单和直接，但其过程也不是很难且能产生强大的功能。

本章包括：

- 声明 API 调用
- 声明类型
- 利用 API 调用

声明 API 调用

可以在代码模块的通用声明区声明 Windows API 调用。一旦声明完后，就可以像使用任何其它的函数或过程那样使用 API 调用了。

下面是一个例子：

```
Public Declare Function Beep Lib "kernel32" (ByVal dwFreq As Long, _
                                             ByVal dwDuration As Long) As Long
```

以上声明中的函数名为“Beep”。它是一个能发出哔吡声的令人惊异的 API 调用，能按照指定的声音长度和高度(或低度)发声。API 函数和过程位于 DLL(动态链接库)文件中，这个 Beep 函数位于 **kernel32.dll** 文件中。让我们尝试着使用它。

在代码模块的通用声明区声明完 Beep 函数后，可如下所示使用它：

```
Sub TestBeep()
    Beep 4000, 250
    Beep 2000, 250
    Beep 1000, 250
    Beep 500, 250
End Sub
```

听到了四声鸣叫，每个以不同的频率持续了 1/4 秒(250 毫秒)。频率越高，声音就越高。每个声音的频率都是前一个的一半，这产生了四个音符，每个音阶都比前一个低。

Beep API 函数不是我们所知的最有用的 API 函数，但在此处使用它主要是演示如何声明函数。随后将看到大量利用更强大和更有用的 API 函数的例子。

许多 Windows API 调用都被声明为函数。这就意味着它们返回一个值。这个返回值常常表示该 API 调用是否正常运行或是遇到错误。

在声明中除了指定函数名外，还要指定位置(该函数位于的 DLL 文件)和参数，Windows API 调用还经常有一个别名。当声明了一个 API 函数但在代码中不用这个函数名而用函数或过程名时，别名很重要。

声明类型

有些 Windows API 调用使用类型。类型类似于对象，它具有特定的属性或成员。我们经常把一个变量声明为一种类型，然后设置它的一些属性，属性设置完后就能用它作为 API 调用的一个参数了。

和 API 函数一样，也在通用声明区声明 API 类型。

```
Public Declare Sub GetSystemInfo Lib "kernel32" (lpSystemInfo _
                                                 As SYSTEM_INFO)
Public Type SYSTEM_INFO
    dwOemID As Long
    dwPageSize As Long
    lpMinimumApplicationAddress As Long
    lpMaximumApplicationAddress As Long
```

```
    dwActiveProcessorMask As Long
    dwNumberOfProcessors As Long
    dwProcessorType As Long
    dwAllocationGranularity As Long
    dwReserved As Long
End Type
```

GetSystemInfo API 调用使用了一个“SYSTEM_INFO”类型。声明完后就可以在一个宏中使用该过程和类型了：

```
Sub TestSystemInfo()
    Dim mySystemInfo As SYSTEM_INFO
    GetSystemInfo mySystemInfo
    MsgBox mySystemInfo.dwNumberOfProcessors & " Processors."
End Sub
```

随着查看更多 API 的例子，您将能看到更多声明类型的例子。

利用 API 调用

有数百个 API 调用可为我们所用。这里列出的这些没有以某种顺序排列，相互之间也没有任何关系。某个调用可能涉及计算机上的逻辑驱动器，而另一个则可能与屏幕分辨率有关。不管怎样，这里列出的这些 API 调用应该对您有所帮助。

GetLogicalDrives

```
Public Declare Function GetLogicalDriveStrings Lib "kernel32" _
                        Alias "GetLogicalDriveStringsA" _
                        (ByVal nBufferLength As Long, _
                        ByVal lpBuffer As String) As Long
Sub TestGetLogicalDriveStrings()
    Dim DriveLetters As String
    Dim xSplit As Variant
    Dim I As Long
```

```
    DriveLetters = Space(255)
    GetLogicalDriveStrings Len(DriveLetters), DriveLetters
    xSplit = Split(DriveLetters, Chr(0))
    For I = LBound(xSplit) To UBound(xSplit) - 2
        MsgBox "Drive " & xSplit(I) & " Found."
    Next I
End Sub
```

在使用 Windows API 调用时，如果有一个输入输出型的字符串参数，那就必须首先用空格填充这个字符串，**DriveLetters** 变量就是这样的一个例子。当变量 **DriveLetters** 进入 GetLogicalDriveStrings 函数时，其中含有 255 个空格字符。当该变量返回时，其中含有一系列以 null 字符(Chr(0))分割的驱动器字母字符。我们从 **xSplit** 数组的上界减二(2)来取得系统中找到的驱动器字母。

运行 TestGetLogicalDriveStrings 时，能看到系统中每个驱动器字母的消息框。返回了所有的驱动器，包括物理的也包括网络映射的驱动器。

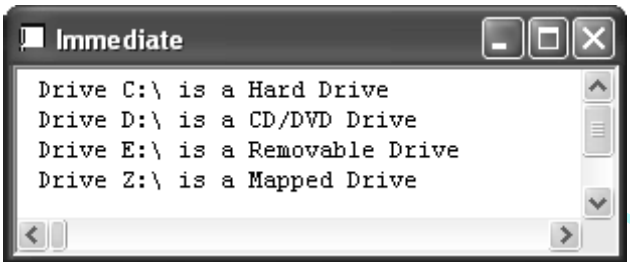
GetDriveType

可以用 GetLogicalDriveStrings 取得系统中的驱动器字母，但如何知道这些驱动器的类型呢？硬盘？CD-ROM？还是软驱？可以使用 GetDriveType 函数。本例也用了 GetLogicalDriveStrings。

```
Public Declare Function GetDriveType Lib "kernel32" Alias _
                        "GetDriveTypeA" (ByVal nDrive As String) As Long
Public Const DRIVE_CDROM = 5
Public Const DRIVE_FIXED = 3
Public Const DRIVE_RAMDISK = 6
Public Const DRIVE_REMOTE = 4
Public Const DRIVE_REMOVABLE = 2
Sub TestDriveType()
    Dim DriveLetters As String
    Dim xSplit As Variant
    Dim I As Long
```

```
    DriveLetters = Space(255)
    GetLogicalDriveStrings Len(DriveLetters), DriveLetters
    xSplit = Split(DriveLetters, Chr(0))
    For I = LBound(xSplit) To UBound(xSplit) - 2
        Debug.Print "Drive " & xSplit(I) & " is a " & _
                    ReturnDriveType(CStr(xSplit(I)))
    Next I
End Sub
Function ReturnDriveType(DriveLetter As String) As String
    Dim lngDriveType As Long
    lngDriveType = GetDriveType(DriveLetter)
    Select Case lngDriveType
    Case DRIVE_CDROM
        ReturnDriveType = "CD/DVD Drive"
    Case DRIVE_FIXED
        ReturnDriveType = "Hard Drive"
    Case DRIVE_RAMDISK
        ReturnDriveType = "RAM Disk"
    Case DRIVE_REMOTE
        ReturnDriveType = "Mapped Drive"
    Case DRIVE_REMOVABLE
        ReturnDriveType = "Removable Drive"
    Case Else
        ReturnDriveType = lngDriveType
    End Select
End Function
```

China BDN



本计算机含有一个硬盘驱动器(C)、一个 CD/DVD 驱动器(D)、一个可移动驱动器(E)恰巧是一个 U 盘、还有一个映射驱动器(Z)。

GetComputerName

```
Public Declare Function GetComputerName Lib "kernel32" _
                Alias "GetComputerNameA" (ByVal lpBuffer As String, _
                nSize As Long) As Long
Sub TestGetComputerName()
    Dim CompName As String
    CompName = Space(255)
    GetComputerName CompName, Len(CompName)
    CompName = Left(CompName, InStr(1, CompName, Chr(0)) - 1)
    MsgBox CompName
End Sub
```

知道我们的代码所运行的计算机名是一个很有用的信息。我们再次使用了一个填充的字符串，在该字串中查找 null 字符 Chr(0)并取得左侧所有的内容。

GetVersionEx

计算机运行的是什么操作系统？只需要用 GetVersionEx 就能查出来。

```
Public Declare Function GetVersionEx Lib "kernel32" Alias _
            "GetVersionExA" (lpVersionInformation As OSVERSIONINFO) _
            As Long
Public Type OSVERSIONINFO
    dwOSVersionInfoSize As Long
    dwMajorVersion As Long
    dwMinorVersion As Long
```

```
    dwBuildNumber As Long
    dwPlatformId As Long
    szCSDVersion As String * 128
End Type
Public Const VER_PLATFORM_WIN32_WINDOWS = 1
Public Const VER_PLATFORM_WIN32_NT = 2
Sub TestOSVersion()
    Dim myVerInfo As OSVERSIONINFO
    Dim strServicePack As String
    myVerInfo.dwOSVersionInfoSize = 148
    GetVersionEx myVerInfo
    Select Case myVerInfo.dwPlatformId
    Case VER_PLATFORM_WIN32_WINDOWS
        Select Case myVerInfo.dwMinorVersion
        Case 0
            MsgBox "Windows 95"
        Case 10
            MsgBox "Windows 98"
        Case 90
            MsgBox "Windows ME"
        End Select
    Case VER_PLATFORM_WIN32_NT
        Select Case myVerInfo.dwMajorVersion
        Case Is <= 4
            MsgBox "Windows NT Build " & myVerInfo.dwBuildNumber
        Case 5
            Select Case myVerInfo.dwMinorVersion
            Case 1
                strServicePack = myVerInfo.szCSDVersion
                strServicePack = Left(strServicePack, _
```


China BDN

```
                                    InStr(1, strServicePack, Chr(0)) - 1)
                MsgBox "Windows XP Build " & _
                        myVerInfo.dwBuildNumber & _
                        vbCr & strServicePack
            Case 2
                MsgBox "Windows .NET Server Build " & _
                        myVerInfo.dwBuildNumber
            Case Else
                MsgBox "Windows 2000 Build " & _
                         myVerInfo.dwBuildNumber
            End Select
        End Select
    End Select
End Sub
```

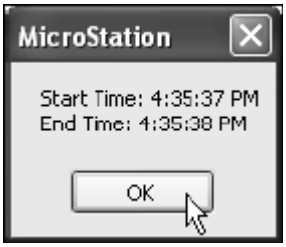


一个消息框显示了操作系统名称，有时情况下还显示有建立号和服务包版本。

Sleep

有时需要暂时停止程序的执行，Sleep 函数能让我们指定暂停多少毫秒。

```
Public Declare Sub Sleep Lib "kernel32" (ByVal _
                        dwMilliseconds As Long)
Sub TestSleep()
    Dim StartTime As Date
    Dim EndTime As Date
    StartTime = Now
    Sleep 1000
    EndTime = Now
    MsgBox "Start Time: " & FormatDateTime(StartTime, vbLongTime) _
            & vbCr & "End Time: " & FormatDateTime(EndTime, vbLongTime)
End Sub
```



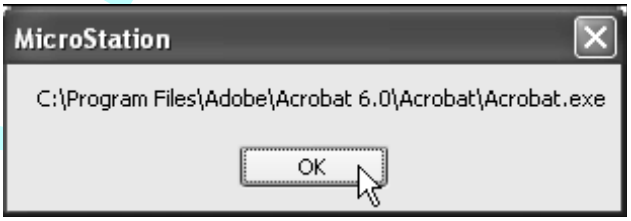
China BDN

在这个例子中，取当前时间到名为“StartTime”的变量中，暂停 1 秒(1000 毫秒)，然后取当前时间到名为“EndTime”的变量中。最后在消息框中显示 StartTime(起始时间)和 EndTime(终止时间)。

FindExecutable

同一个.pdf 文件在一台计算机上可能由 Adobe Acrobat Reader 7 来打开，而在另一台计算机上则是由 Adobe Acrobat 6 来打开。哪个程序被注册用来打开.pdf 文件？哪个程序被注册用来打开.jpg 文件？FindExecutable 告诉我们注册用于打开某特定文件类型的程序路径。

```
Public Declare Function FindExecutable Lib "shell32.dll" _
                Alias "FindExecutableA" _
                (ByVal lpFile As String, ByVal lpDirectory As String, _
                ByVal lpResult As String) As Long
Sub TestFindExecutable()
    Dim strFileName As String
    Dim strDirName As String
    Dim strExeFile As String
    strFileName = "eula.pdf"
    strDirName = "C:\Program Files\Bentley\MicroStation"
    strExeFile = Space(255)
    FindExecutable strFileName, strDirName, strExeFile
    strExeFile = Left(strExeFile, InStr(1, strExeFile, Chr(0)) - 1)
    MsgBox strExeFile
End Sub
```



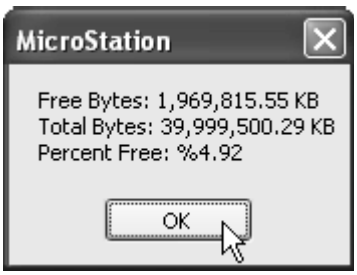
China BDI

GetDiskFreeSpace

GetDiskFreeSpace 能给出指定磁盘名的磁盘信息。

```
Public Declare Function GetDiskFreeSpace Lib "kernel32" Alias _
        "GetDiskFreeSpaceA" _
        (ByVal lpRootPathName As String, lpSectorsPerCluster As Long, _
        lpBytesPerSector As Long, lpNumberOfFreeClusters As Long, _
        lpTotalNumberOfClusters As Long) As Long
Sub TestFreeDiskSpace()
    Dim mySpace As Long
    Dim lngSectorsPerCluster As Long
    Dim lngBytesPerSector As Long
    Dim lngFreeClusters As Long
    Dim lngTotalClusters As Long
    Dim FreeBytes As Double
    Dim TotalBytes As Double
    Dim PercentFree As Double
    GetDiskFreeSpace "C:\", lngSectorsPerCluster, _
                    lngBytesPerSector, lngFreeClusters, lngTotalClusters
    FreeBytes = FormatNumber(lngFreeClusters / 1000 * _
                lngBytesPerSector * lngSectorsPerCluster, 2)
    TotalBytes = FormatNumber(lngTotalClusters / 1000 * _
                lngBytesPerSector * lngSectorsPerCluster, 2)
    PercentFree = Round((FreeBytes / TotalBytes) * 100, 2)
  MsgBox "Free Bytes: " & FormatNumber(FreeBytes, 2, False, False, True) _
        & " KB" & vbCr & "Total Bytes: " & _
        FormatNumber(TotalBytes, 2, False, False, True) & _
        " KB" & vbCr & "Percent Free: %" & PercentFree
End Sub
```

China BDN



GetSystemMetrics

了解我们代码所运行的计算机的信息非常有用。可以使用 GetSystemMetrics 来返回大量有用的信息。

```
Public Declare Function GetSystemMetrics Lib "user32" _
                                (ByVal nIndex As Long) As Long
Public Const SM_CMETRICS = 44
Public Const SM_CMOUSEBUTTONS = 43
Public Const SM_CXBORDER = 5
Public Const SM_CXCURSOR = 13
Public Const SM_CXDLGFRAME = 7
Public Const SM_CXDOUBLECLK = 36
Public Const SM_CXFIXEDFRAME = SM_CXDLGFRAME
Public Const SM_CXFRAME = 32
Public Const SM_CXFULLSCREEN = 16
Public Const SM_CXHSCROLL = 21
Public Const SM_CXHTHUMB = 10
Public Const SM_CXICON = 11
Public Const SM_CXICONSPACING = 38
Public Const SM_CXMIN = 28
Public Const SM_CXMINTRACK = 34
Public Const SM_CXSCREEN = 0
Public Const SM_CXSIZE = 30
Public Const SM_CXSIZEFRAME = SM_CXFRAME
Public Const SM_CXVSCROLL = 2
```

China BDN

```
Public Const SM_CYBORDER = 6
Public Const SM_CYCAPTION = 4
Public Const SM_CYCURSOR = 14
Public Const SM_CYDLGFRAME = 8
Public Const SM_CYDOUBLECLK = 37
Public Const SM_CYFIXEDFRAME = SM_CYDLGFRAME
Public Const SM_CYFRAME = 33
Public Const SM_CYFULLSCREEN = 17
Public Const SM_CYHSCROLL = 3
Public Const SM_CYICON = 12
Public Const SM_CYICONSPACING = 39
Public Const SM_CYMENU = 15
Public Const SM_CYKANJIWINDOW = 18
Public Const SM_CYMINTRACK = 35
Public Const SM_CYMIN = 29
Public Const SM_CYSCREEN = 1
Public Const SM_CYSIZE = 31
Public Const SM_CYSIZEFRAME = SM_CYFRAME
Public Const SM_CYVSCROLL = 20
Public Const SM_CYVTHUMB = 9
Public Const SM_DBCSENABLED = 42
Public Const SM_DEBUG = 22
Public Const SM_MENUDROPALIGNMENT = 40
Public Const SM_MOUSEPRESENT = 19
Public Const SM_SWAPBUTTON = 23
Sub TestSystemMetrics()
    MsgBox "Screen Resolution: " & vbCr & _
            GetSystemMetrics(SM_CXSCREEN) & " X " & _
            GetSystemMetrics(SM_CYSCREEN)
    MsgBox "Mouse Buttons: " & _
```


China BDN

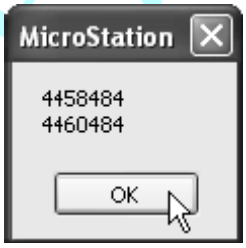
```
                GetSystemMetrics(SM_CMOUSEBUTTONS)
End Sub
```

过程 TestSystemMetrics 返回屏幕分辨率以及连接到计算机上的鼠标的按键个数。

GetTickCount

自计算机开机至今经过了多长时间？GetTickCount 以毫秒数回答这个问题。

```
Public Declare Function GetTickCount Lib "kernel32" Alias _
                        "GetTickCount" () As Long
Sub TestTickCount()
    Dim StartTicks As Long
    Dim EndTicks As Long
    StartTicks = GetTickCount
    Sleep 2000
    EndTicks = GetTickCount
    MsgBox StartTicks & vbCr & EndTicks
End Sub
```



GetUserName

谁登录到了计算机中？

```
Public Declare Function GetUserName Lib "advapi32.dll" Alias _
    "GetUserNameA" (ByVal lpBuffer As String, nSize As Long) As Long
Sub TestUserName()
    Dim UserName As String
    Dim xSplit As Variant
    UserName = Space(255)
    GetUserName UserName, Len(UserName)
    xSplit = Split(UserName, Chr(0))
    UserName = xSplit(0)
```

China BDN

```
    MsgBox UserName
End Sub
```

GetUserName 对于记录谁在执行什么功能很有用处。如果也取得了计算机名，我们就知道了谁、何处、干什么，用 Now 函数也能知道是什么时间。对于问题“为什么？”，微软还正在开发这个 API 调用(译者注：作者的幽默说法)。

> **注意**：MicroStation 的 Application 对象有其自己的 UserName 属性，所以在使用 MicroStation VBA 时不需要这个 API 调用。

GetWindowsDirectory

Windows 安装在什么位置？C:\Winnt？C:\Windows？

GetWindowsDirectory 告诉我们。

```
Public Declare Function GetWindowsDirectory Lib _
                "kernel32" Alias _
                "GetWindowsDirectoryA" (ByVal lpBuffer As String, _
                ByVal nSize As Long) As Long
Sub TestWindowsDir()
    Dim WindowsDir As String
    Dim xSplit As Variant
    WindowsDir = Space(255)
    GetWindowsDirectory WindowsDir, Len(WindowsDir)
    xSplit = Split(WindowsDir, Chr(0))
    WindowsDir = xSplit(0)
    MsgBox WindowsDir
End Sub
```

如果我们想查找一个特定的文件(如在 System32 目录中)，或者想在 Windows 目录下添加自己的文件或文件夹，知道 Windows 的安装位置是很有帮助的。

LogonUser

每个人的内心都有安全意识。如何知道在某台机器上所做的修改是由登录的

人作出的？如何知道在佛瑞德午餐时没有人溜进他的卧室弄乱一个文件？

我们知道如何取得当前用户。下面看一下如何用 Windows API 来向用户提问密码并验证输入的密码与系统密码匹配。

```
Private Declare Function LogonUser Lib "Advapi32" Alias "LogonUserA" _
        (ByVal lpszUsername As String, ByVal lpszDomain As String, _
        ByVal lpszPassword As String, ByVal dwLogonType As Long, _
        ByVal dwLogonProvider As Long, phToken As Long) As Long
Function CurrentUserName() As String
    Dim UserName As String
    Dim xSplit As Variant
    UserName = Space(255)
    GetUserName UserName, Len(UserName)
    xSplit = Split(UserName, Chr(0))
    UserName = xSplit(0)
    CurrentUserName = UserName
    End Function
    Sub TestLogonUser()
    Dim Successful As Long
    Dim PasswordEntry As String
    PasswordEntry = InputBox("Re-Enter Password:", "Logon Validation")
    Successful = LogonUser(CurrentUserName, "", PasswordEntry, 2, 0, 0)
    Select Case Successful
    Case 1
        MsgBox "You have been validated.", vbInformation
    Case Else
        MsgBox "Invalid Username/Password Combination.", vbCritical
    End Select
End Sub
```

本例用一个输入框来输入密码。虽然这不是最佳的提问密码方式(因为输入

的密码对旁边的人也可见)，但这段代码仍然演示了该 API 调用。我们有一个叫做 CurrentUserName 的函数，它返回当前用户名。注意，我们在域参数处给了一个空串。如果我们位于一个域中，就要在这个参数中指定域(domain)。

LogonUser，功能非常强大。

MessageBeep

反馈很好，对吗？除了可视化的反馈外，还能通过各种方法提供声音反馈。MessageBeep 函数播放控制面板“声音和音频设备”部分当前提供的.wav 文件。这些是显示不同的消息框时我们听到的声音，MessageBeep 只发声而不显示消息框。

```
Public Declare Function MessageBeep Lib "user32" _
                        (ByVal wType As Long) As Long
Public Const MB_OK = &H0&
Public Const MB_ICONSTOP = &H10&
Public Const MB_ICONQUESTION = &H20&
Public Const MB_ICONEXCLAMATION = &H30&
Public Const MB_ICONASTERISK = &H40&
Sub TestMessageBeep()
    MessageBeep MB_OK
    Sleep 500
    MessageBeep MB_ICONSTOP
    Sleep 500
    MessageBeep MB_ICONQUESTION
    Sleep 500
    MessageBeep MB_ICONEXCLAMATION
    Sleep 500
    MessageBeep MB_ICONASTERISK
End Sub
```

China BDN

PlaySound

这里是另一个涉及音频反馈的 API 调用。PlaySound 让我们指定要播放的.wav 文件以及如何播放该文件。

```
Public Declare Function PlaySound Lib "winmm.dll" Alias "PlaySoundA" _
                    (ByVal lpszName As String, ByVal hModule As Long, _
                    ByVal dwFlags As Long) As Long
Public Const SND_SYNC = &H0
Public Const SND_ASYNC = &H1
Public Const SND_LOOP = &H8
Sub TestPlaySoundA()
    PlaySound "C:\Windows\Media\chimes.wav", 0, SND_SYNC
    PlaySound "C:\Windows\Media\chord.wav", 0, SND_SYNC
End Sub
```

这第一个例子使用“SND_SYNC”标记。该标记的意思是直到 **chimes.wav** 文件播放完毕后才播放 **chord.wav** 文件。

```
Sub TestPlaySoundB()
    PlaySound "C:\Windows\Media\chimes.wav", 0, SND_ASYNC
    Sleep 150
    PlaySound "C:\Windows\Media\chimes.wav", 0, SND_ASYNC
    Sleep 200
    PlaySound "C:\Windows\Media\chimes.wav", 0, SND_ASYNC
    Sleep 250
    PlaySound "C:\Windows\Media\chimes.wav", 0, SND_ASYNC
    Sleep 300
    PlaySound "C:\Windows\Media\chimes.wav", 0, SND_ASYNC
End Sub
```

当使用“SND_ASYNC”标记时，指定的文件开始播放，程序代码不等待文件播放完就继续执行。在每行代码间放置了 sleep 调用后，我们听到了 150 毫秒的 **chimes.wav** 然后是 200 毫秒、250 毫秒等等。PlaySound 一次只能播

放一个.wav 文件，所以每次调用它时，正在播放的文件被停止，新文件开始播放。

假如不知道如何关掉它，那么下个例子将会很烦人。我们使用“SND_LOOP”加上“SND_ASYNC”标记来反复播放声音。即使该过程执行完后声音还在继续播放，VBA 工程卸载后声音还在反复播放。直到关闭 MicroStation 或执行 TestPlaySoundD 才会结束。

```
Sub TestPlaySoundC()
    PlaySound "C:\Windows\Media\chord.wav", 0, _
                SND_LOOP + SND_ASYNC
End Sub
Sub TestPlaySoundD()
    PlaySound vbNullString, 0, 0
End Sub
```

ShellExecute

在前面的例子中，我们能找到打开一种特定文件所注册的是哪个应用程序。ShellExecute 用所注册的应用程序实际打开文件。

```
Public Declare Function ShellExecute Lib "shell32.dll" Alias _
        "ShellExecuteA" (ByVal hwnd As Long, _
        ByVal lpOperation As String, ByVal lpFile As String, _
        ByVal lpParameters As String, ByVal lpDirectory As String, _
        ByVal nShowCmd As Long) As Long
Sub TestShellExecuteA()
    ShellExecute 0, "OPEN", _
        "Greenstone.bmp", "", "C:\Windows", vbMinimizedNoFocus
End Sub
Sub TestShellExecuteB()
    ShellExecute 0, "OPEN", _
        "Greenstone.bmp", "", "C:\Windows", vbMaximizedFocus
End Sub
```

这里示出了两个例子，一个执行程序并以最小窗口方式打开文件，另一个执行程序并以最大窗口方式打开文件。

在前面的一章中我们完成了一个建立.html 文件的例子，在建立完文件后可以用 ShellExecute 立即显示它。这样做比建立文件然后让用户找到文件并双击要好得多。

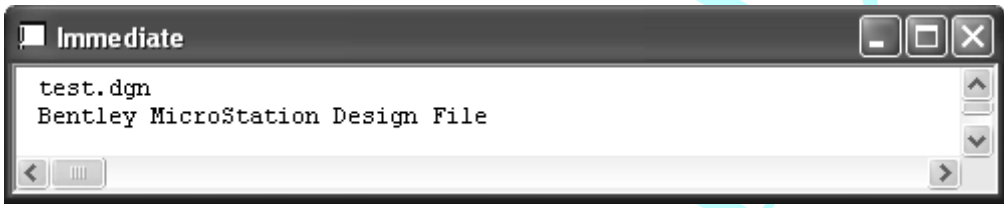
SHGetFileInfo

SHGetFileInfo 有多种用途。其中之一是能告诉我们查看的文件是哪一种类。

```
Public Type SHFILEINFO
    hIcon As Long
    iIcon As Long
    dwAttributes As Long
    szDisplayName As String * 255
    szTypeName As String * 80
End Type
Public Const SHGFI_ATTRIBUTES = &H800
Public Const SHGFI_EXETYPE = &H2000
Public Const SHGFI_DISPLAYNAME = &H200
Public Const SHGFI_ICON = &H100
Public Const SHGFI_ICONLOCATION = &H1000
Public Const SHGFI_LARGEICON = &H0
Public Const SHGFI_LINKOVERLAY = &H8000
Public Const SHGFI_OPENICON = &H2
Public Const SHGFI_PIDL = &H8
Public Const SHGFI_SELECTED = &H10000
Public Const SHGFI_SHELLICONSIZE = &H4
Public Const SHGFI_SMALLICON = &H1
Public Const SHGFI_SYSICONINDEX = &H4000
Public Const SHGFI_TYPENAME = &H400
Public Const SHGFI_USEFILEATTRIBUTES = &H10
```

```
Sub TestGetFileInfo()
    Dim myFI As SHFILEINFO
    SHGetFileInfo "C:\test.dgn", SHGFI_ATTRIBUTES, myFI, Len(myFI), _
            SHGFI_DISPLAYNAME + SHGFI_ICON + SHGFI_TYPENAME
    Debug.Print Replace(myFI.szDisplayName, Chr(0), "")
    Debug.Print Replace(myFI.szTypeName, Chr(0), "")
End Sub
```

程序执行后显示了文件名和文件类型。

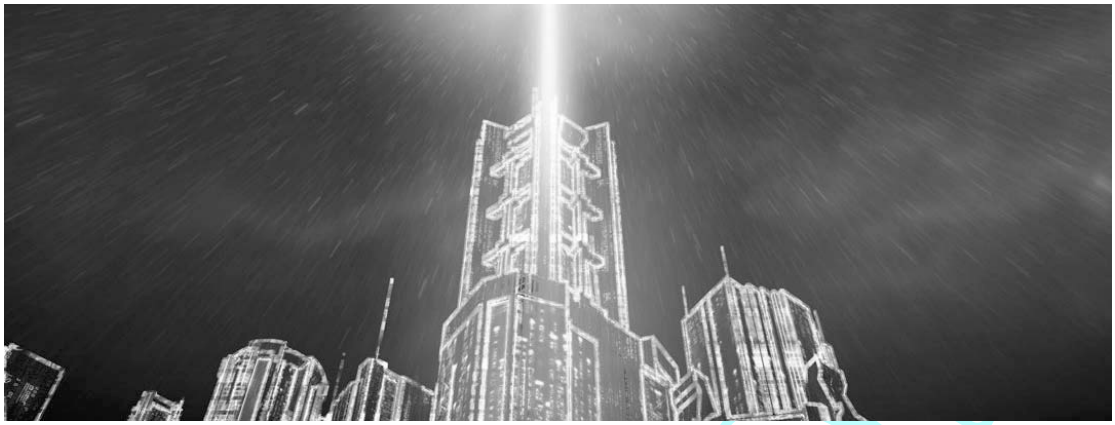


本章回顾

本章只触及了少数几个 Windows API 调用，API 调用有数百之多。有些 API 调用与显示有关，它们能在特定窗口绘制图形。另一些 API 调用涉及文件读写，还有一些涉及系统内存管理。

有的书和网站整个都在专门讨论 Windows API 调用的主题。Windows API 调用是我们能说“是的，能用 VBA 做这件事”的原因之一。

China DN



33. 使用第三方 ActiveX 控件和 DLL

我们大家都知道，没有必要再重新发明车轮。使用已有的资源能加快开发进度并能生成错误更少的应用程序。

本章包括：

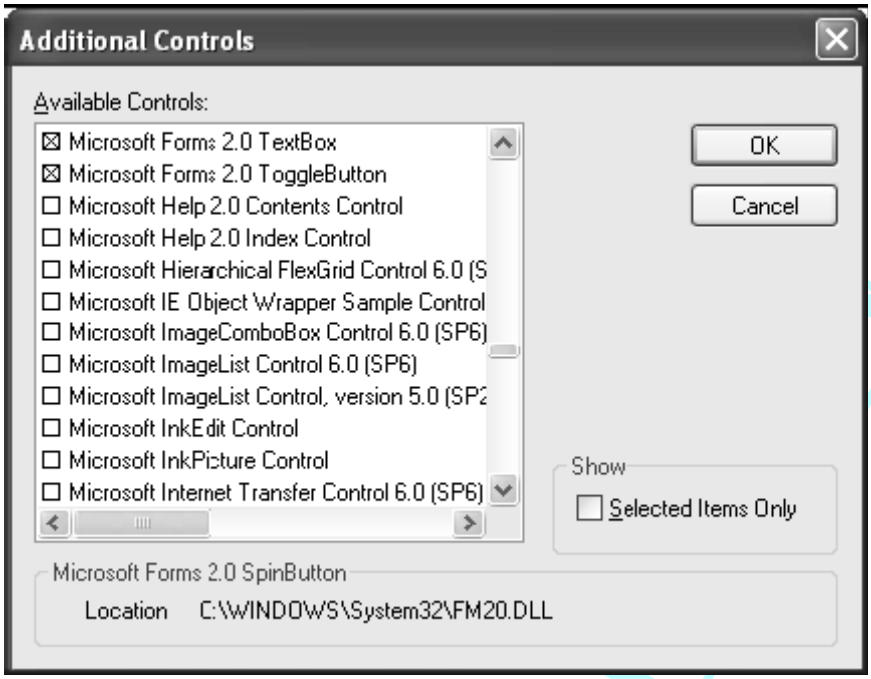
- 使用 ActiveX 控件
- 使用 DLL

使用 ActiveX 控件

ActiveX 控件用于图形用户界面的开发。我们已经使用过文本框、复合框、命令按钮、标签以及其它控件。

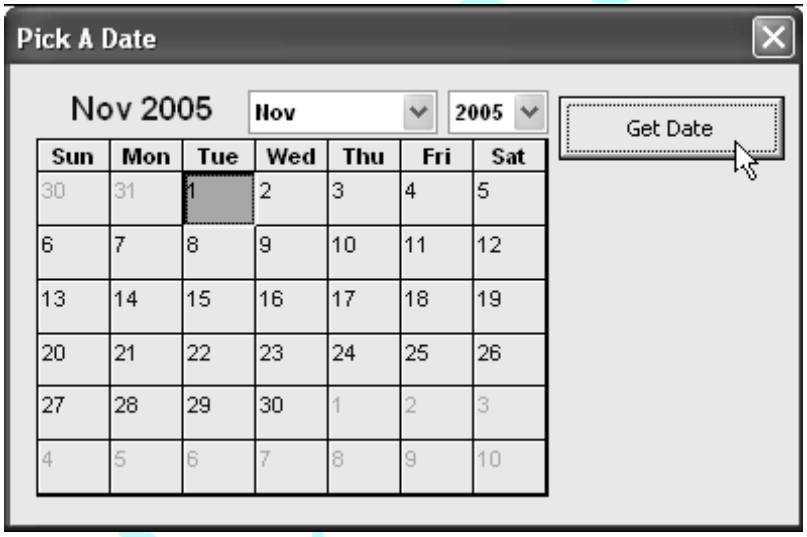


当在 VBA 工程中插入一个用户窗体时，通常会显示工具箱对话框。在工具箱上右击能让我们添加**附加控件(Additional Control)**。



随着在列表中的滚动，您看到了大量的控件。每台计算机上的列表可能不同，因为控件是随软件的安装而添加上的。上图所示的有些控件是在安装 Visual Basic 6.0 时安装上的。

列表中显示了控件并不意味着您就能够使用它，因为有些控件要求一个许可。让我们看一些可用的控件。



出于演示之目的，本例使用“Calendar Control 11.0”控件，该控件随 Microsoft Office 一起安装。从控件列表中选择该控件并点击 OK 按钮。

一旦该控件出现在工具箱中，我们就能把它与其它控件一起放入窗体中。点击上图中的“Get Date(取得日期)”按钮显示所选择的日期。

```
Private Sub btnDisplaySelection_Click()
    MsgBox Calendar1.Value
```

End Sub

如果您编写的应用程序使用了一个非标准的 ActiveX 控件，那么需要确保能成功地将该应用部署到其它计算机上。

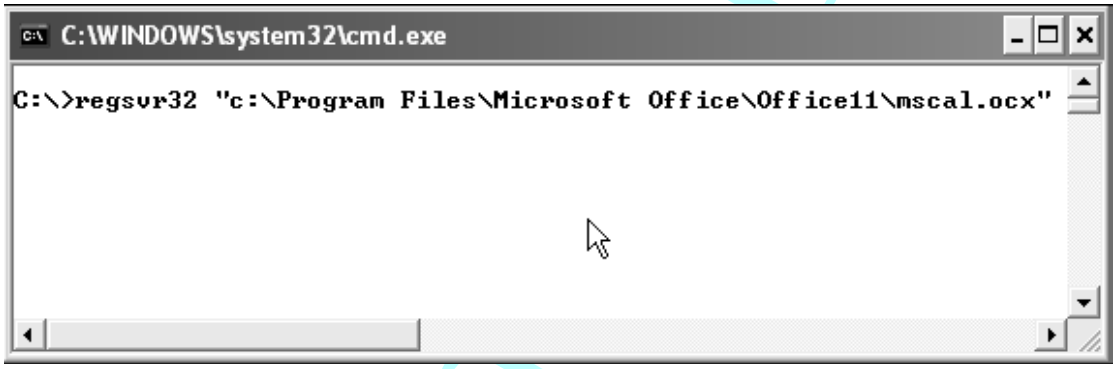
在这个例子中，需要分发.mvba 文件(MicroStation VBA 工程文件)和日历控件(.ocx 文件)。但光把.ocx 文件给用户还不够，控件需要在用户计算机上注册后才能使用。

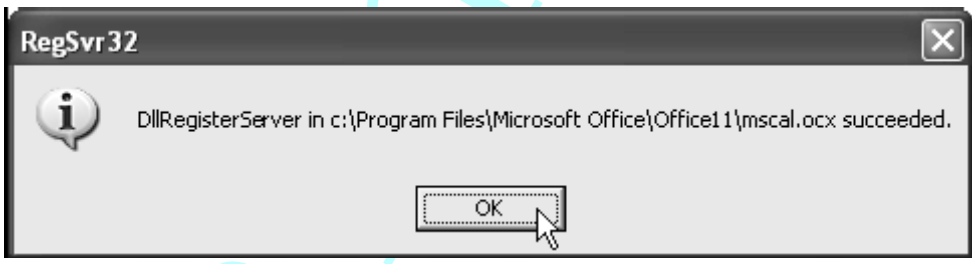
注册一个 ActiveX 控件相当简单，从 Windows 的“开始”按钮中点击“运行”并在窗口中键入“cmd”，然后点击 OK 按钮。

当打开命令窗口后，只需简单地键入：

regsvr32 “c:\program files\Microsoft Office\Office11\mscal.ocx”

并按<Enter>键。如果.ocx 文件所在的路径不同，则应在命令中作相应改变。





如前所提及的，ActiveX 控件应当在以前未安装过的机器上测试，以保证开发的应用程序能成功地部署到他人的机器上。

Internet 上有许多能找到 ActiveX 控件的地方，并且许多网站允许在购买前免费评估控件。

除了购买 ActiveX 控件外，还可以用 Visual Basic 6 来建立自己的控件。

使用现有的 DLL

在程序中有两种基本的使用 DLL 的方法。每种方法都已在本书中使用过。

方法 1：声明 DLL 函数，如 Windows API 那一章所述；

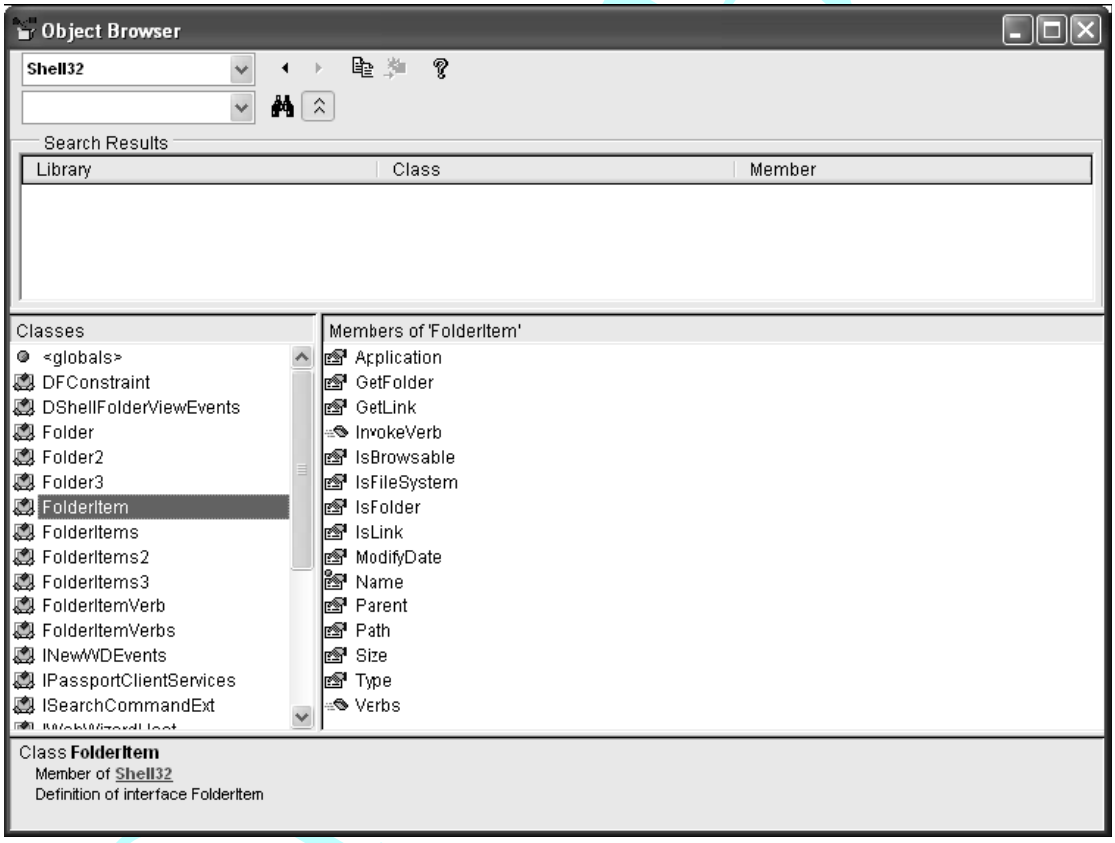
方法 2：添加 DLL 引用并使用它。

因为已经专门有一章来使用 Windows API，所以我们将注意力集中在添加引用上。

我们已经添加过一些不同 DLL 的引用，但当时是在讨论某个特定的主题。现在我们来更详细地查看一些引用。

首先让我们来查看“Microsoft Shell Controls and Automation”引用。

在 VBA 工程中添加新引用。或许以前没有用过它，或许用过一点点。如何发现该引用能做什么呢？第一件要做的事是在对象浏览器中看一眼并过滤到刚添加的引用。



发现某个引用能实现什么功能的最好方法之一是在对象上点击，浙江显示该对象的成员。让我们看一些代码。

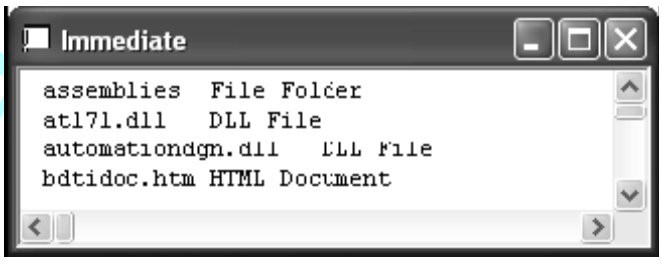
```
Sub TestShellA()
    Dim myShell As New Shell32.Shell
    Dim myFolder As Shell32.Folder3
    Dim myFolder2 As Shell32.Folder3
```

```
    Dim myItem As Shell32.FolderItem
    Set myFolder = myShell.BrowseForFolder(0, "Browse", 0)
    MsgBox myFolder.Self.Path
    For Each myItem In myFolder.Items
        Debug.Print myItem.Name & vbTab & myItem.Type
    Next
End Sub
```

这段代码首先要求我们选择一个文件夹。



在消息框中显示选择的文件夹路径后，在立即窗口中显示了文件夹中的每一项。

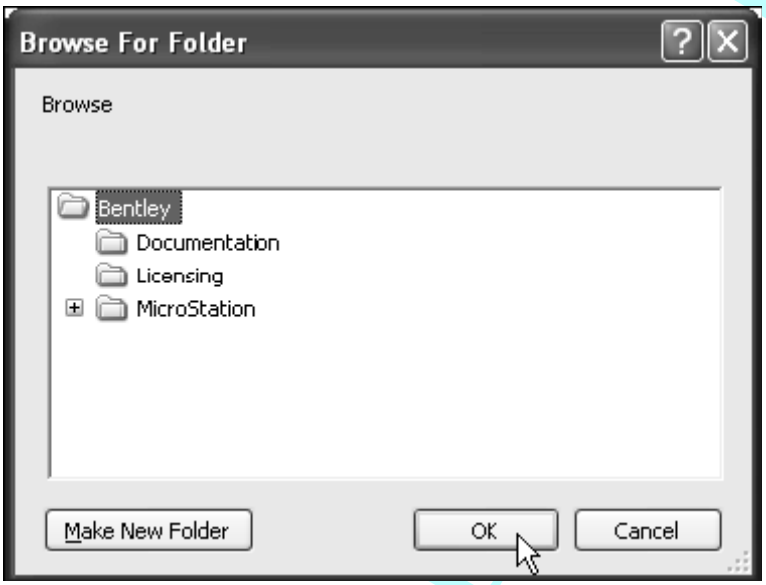


我们看到了对象名及其类型。

Shell32 BrowseForFolder 方法最值得一提的是，调用它时还能指定根文件夹。

```
Sub TestShellB()
    Dim myShell As New Shell32.Shell
    Dim myFolder As Shell32.Folder3
```

```
    Dim myFolder2 As Shell32.Folder3
    Dim myItem As Shell32.FolderItem
    Set myFolder = myShell.BrowseForFolder(0, "Browse", 0, _
                                           "C:\Program Files\Bentley")
    For Each myItem In myFolder.Items
        Select Case myItem.Type
        Case "File Folder"
            Debug.Print "Folder: " & myItem.Name
        End Select
    Next
End Sub
```



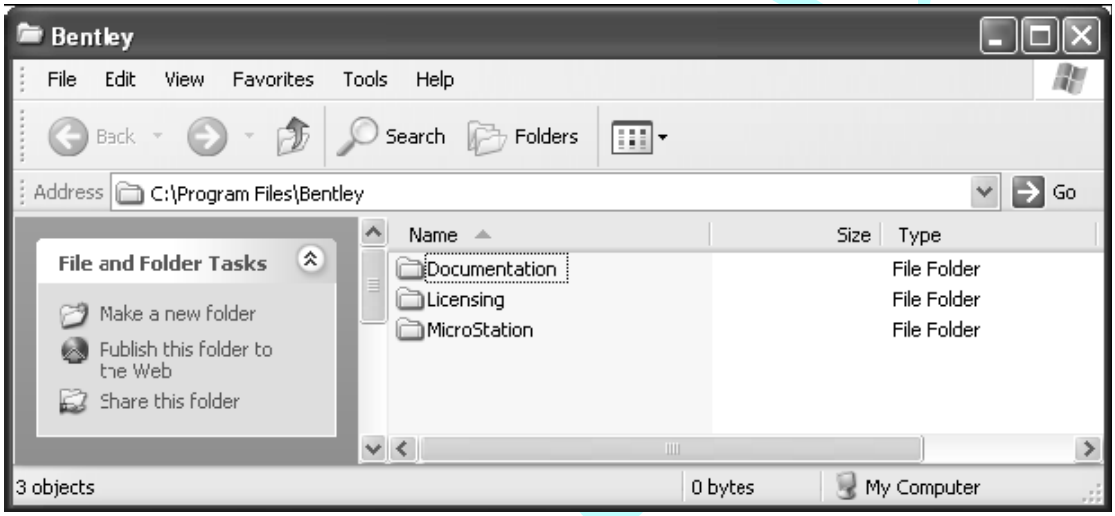
现在，当用户运行该宏时，只能选择根文件夹或根文件夹路径下的一个文件夹。

如果想让用户从工程文件夹下选择东西时这一点很有用。因为选择是从正确的文件夹开始，所以对用户来说很快捷。对于程序设计者来说也很好，因为能知道用户将从根文件夹路径下选择。

下面是更多的一些例子，它们全都没有破坏作用，即使是 ShutdownWindows 调用也是如此。使用 ShutdownWindows 时，将会出现您所期望的提示，只需点击“取消”就一切 OK 了。在使用该调用前保存打开的文档是个好主意，仅仅是为安全起见。

```
Sub TestShellC()
    Dim myShell As New Shell32.Shell
    myShell.FindComputer
End Sub
```

```
Sub TestShellD()
    Dim myShell As New Shell32.Shell
    myShell.ShutdownWindows
End Sub
Sub TestShellE()
    Dim myShell As New Shell32.Shell
    myShell.Open "C:\Program Files\Bentley"
End Sub
```



“Open”在资源管理器窗口中显示所提供路径中的内容。

Microsoft Scripting Runtime

我们已经使用过 Microsoft Scripting Runtime(微软脚本运行时)，让我们更详细地查看它。将要使用的基本对象是文件系统对象(FileSystemObject)。

第一个例子在一特定的文件夹下查找所有的“Bentley MicroStation Design File”文件。

```
Sub TestFSOA()
    Dim myFSO As New Scripting.FileSystemObject
    Dim myFolder As Scripting.Folder
    Dim myFile As Scripting.File
    Set myFolder = myFSO.GetFolder _
                        ("C:\Program Files\Bentley\MicroStation")
```

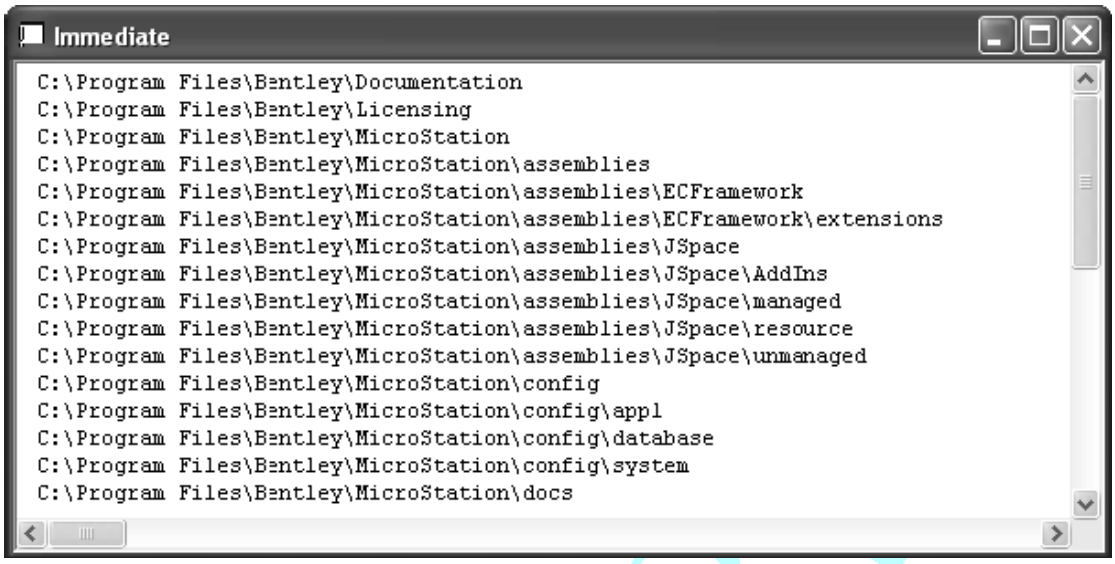
```
        For Each myFile In myFolder.Files
            Select Case myFile.Type
            Case "Bentley MicroStation Design File"
                Debug.Print myFile.Name
            End Select
        Next
    End Sub
```

立即窗口中显示符合指定准则的文件名。

除了查看文件夹下的文件外，还能取得给定文件夹下的子文件夹甚至是所有的子文件夹。下一个例子完整地列出了一个根文件夹下的所有文件夹。这段代码利用了叫做递归执行的技术(参见第 30 章)。

```
    Sub TestFSOB()
        Dim myFSO As New Scripting.FileSystemObject
        Dim myFolder As Scripting.Folder
        Dim mySubFolder As Scripting.Folder
        Set myFolder = myFSO.GetFolder("C:\Program Files\Bentley")
        For Each myFolder In myFolder.SubFolders
            TraverseFolders myFolder
        Next
    End Sub
    Sub TraverseFolders(FolderIn As Scripting.Folder)
        Dim NextSubFolder As Scripting.Folder
        Debug.Print FolderIn.Path
        For Each NextSubFolder In FolderIn.SubFolders
            TraverseFolders NextSubFolder
        Next
    End Sub
```

Chinese BDN



这项技术在执行批处理时很有用。

可以用文件系统对象取得驱动器信息以及文件和文件夹信息。

```
Sub TestFSOC()
    Dim myFSO As New Scripting.FileSystemObject
    Dim myDrive As Scripting.Drive
    Dim I As Long
    For I = Asc("A") To Asc("Z")
        If myFSO.DriveExists(Chr(I)) Then
            Set myDrive = myFSO.GetDrive(Chr(I) & ":\")
            If myDrive.DriveType = Remote Then
                Debug.Print "Drive: " & myDrive.DriveLetter
                Debug.Print "Share Name: " & myDrive.ShareName
                Debug.Print "Volume: " & myDrive.VolumeName
                Debug.Print "Total Space: " & _
                            FormatNumber(myDrive.TotalSize, _
                            0, False, False, True) & " Bytes"
                Debug.Print ""
            End If
        End If
    Next I
```

End Sub

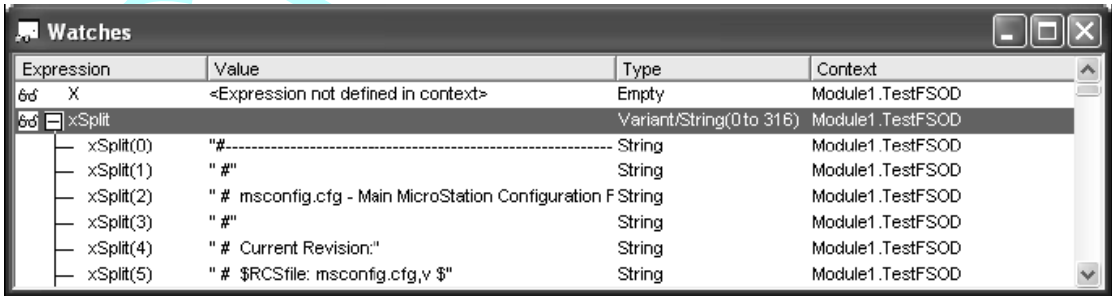


本例取得所有网络驱动器并显示每个驱动器的信息。

下一个例子用文件系统对象读取一个 ASCII 文件。

```
Sub TestFSOD()
    Dim myFSO As New Scripting.FileSystemObject
    Dim myFile As Scripting.File
    Dim myTS As Scripting.TextStream
    Dim strWholeFile As String
    Dim xSplit As Variant
    Set myFile = myFSO.GetFile( _
        "C:\Program Files\Bentley\MicroStation\config\msconfig.cfg")
    Set myTS = myFile.OpenAsTextStream
    strWholeFile = myTS.ReadAll
    xSplit = Split(strWholeFile, vbCr)
    myTS.Close
End Sub
```

TestFSOD 打开 msconfig.cfg 文件。使用 TextStream 对象的“ReadAll”方法读取整个文件，然后通过在文件中查找回车符把文件分割成多个行。最后但不是最不重要的一步是关闭文件。

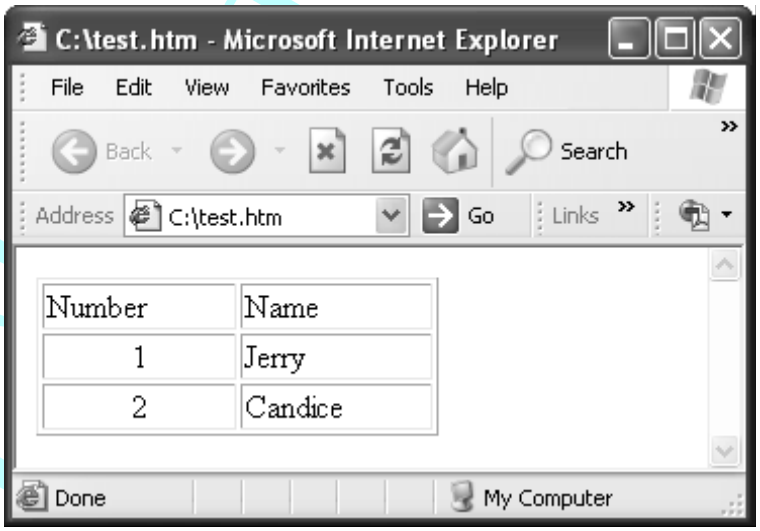


如果对变量 **xSplit** 进行跟踪，将能看到文件被成功地读取并被分割到 **xSplit**

变量中。此时就可以逐行查看文件了。

下面用文件系统对象写一个文件，实际上是建立一个小的 HTML 文件。

```
Sub TestFSOE()
    Dim myFSO As New Scripting.FileSystemObject
    Dim myFile As Scripting.File
    Dim myTS As Scripting.TextStream
    Set myTS = myFSO.CreateTextFile("C:\test.htm", True)
    myTS.WriteLine "<html>"
    myTS.WriteLine vbTab & "<table width=200 border=1>"
    myTS.WriteLine vbTab & "<tr><td>Number</td><td>Name</td></tr>"
    myTS.WriteLine vbTab & "<tr><td align=center>1</td>" & _
                    "<td>Jerry</td></tr>"
    myTS.WriteLine vbTab & "<tr><td align=center>2</td>" & _
                    "<td>Candice</td></tr>"
    myTS.WriteLine vbTab & "</table>"
    myTS.WriteLine vbTab & "</html>"
    myTS.Close
End Sub
```



没什么特别的，这只是一个简单的 HTML 文件。以前写 ASCII 文本文件时，我们使用了标准 VBA 调用，如“Open”和“Print”。

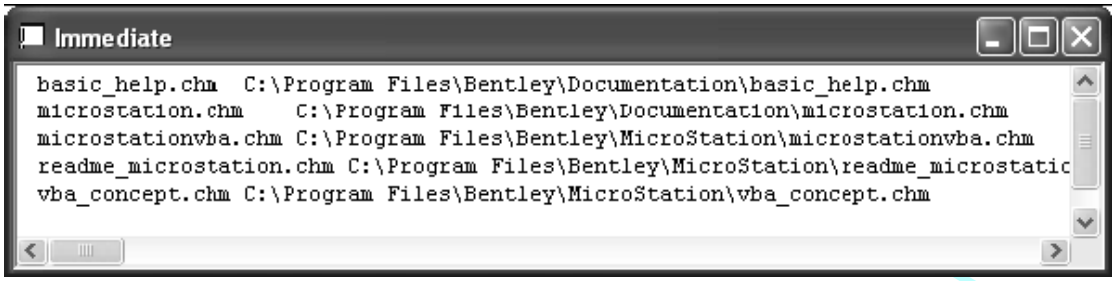
最后一个要查看的文件系统对象是字典对象，它能让我们向一个现成的集合

中添加项目对(关键值和项目)。

```
Sub TestFSOF()
    Dim myFSO As New FileSystemObject
    Dim myDir As Scripting.Folder
    Dim myDictionary As New Dictionary
    Dim I As Long
    Set myDir = myFSO.GetFolder("C:\Program Files\Bentley")
    RecursiveFolder myDir, myDictionary
    For I = 1 To myDictionary.Count
        Debug.Print myDictionary.Keys(I - 1) & vbTab & _
                    myDictionary.Items(I - 1)
    Next I
End Sub
Sub RecursiveFolder(FolderIn As Scripting.Folder, _
                    DictionaryIn As Scripting.Dictionary)
    Dim myFile As Scripting.File
    Dim mySubDir As Scripting.Folder
    For Each myFile In FolderIn.Files
        If UCase(Right(myFile.Name, 4)) = ".CHM" Then
            DictionaryIn.Add myFile.Name, myFile.Path
        End If
    Next
    For Each mySubDir In FolderIn.SubFolders
        RecursiveFolder mySubDir, DictionaryIn
    Next
End Sub
```

TestFSOF 从文件夹 C:\Program Files\Bentley 开始查找所有.chm(帮助)文件。找到一个就添加到字典对象中，待所有文件都找到后，就把字典中的每个关键字和项目打印到调试窗口(立即窗口)中。

China BDN



FileSystemObject 非常有用。使用该对象能取得文件、文件夹和驱动器的信息，读和写 ASCII 文本文件，利用字典对象分类文件。

Microsoft Speech Object Library

Microsoft Speech Object Library(微软语音对象库)是另一个仅需添加一个引用就能获得强大功能的例子。

```
Sub TestSpeechA()
    Dim myVoice As New SpeechLib.SpVoice
    myVoice.Speak "MicroSteyshen V, B, A, Is Great!"
End Sub
```

如果没有这个引用，要向您的软件中添加声音/说话功能将会多么困难？需要承认的是，我们不能总是像通常那样来拼写，我们需要按语音学的要求来拼写单词(或许这很有趣？)。

我们用 Windows API 调用来播放.wav 文件。当使用语音对象时，不需安装大量的.wav 文件就能播放大量的声音指令。甚至还能让语音对象动态地说话，在您的应用程序运行时，向用户提供语音指令和语音反馈。

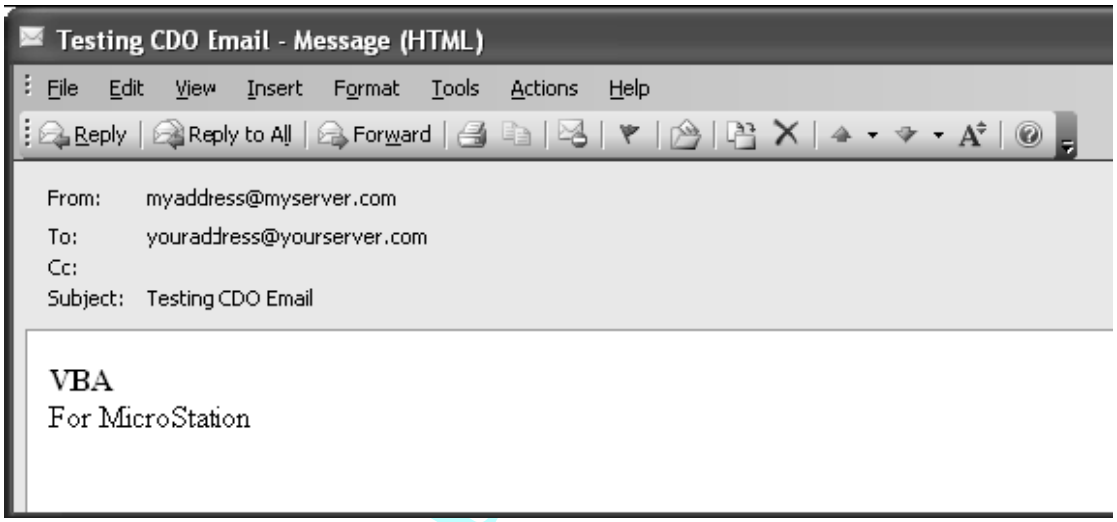
Microsoft CDO for Windows 2000 Library

只要安装了 Windows XP，“CDO for Windows 2000 Library”引用就应该可用。如果引用列表中没有显示该引用，可浏览到 C:\Windows\System32\cdosys.dll 并添加之。

```
Sub TestCDOA()
    Dim myMsg As New CDO.Message
    Dim FieldBase As String
    FieldBase = "http://schemas.microsoft.com/cdo/configuration/"
    myMsg.To = "youraddress@yourserver.com"
```

```
    myMsg.From = "myaddress@myserver.com"
    myMsg.Subject = "Testing CDO Email"
    myMsg.HTMLBody = "<b>VBA</b><br>For MicroStation"
    myMsg.Configuration.Fields.Item(FieldBase & "sendusing") = 2
    myMsg.Configuration.Fields.Item(FieldBase & "smtpserver") = _
                                        "smtp.yourserver.com"
    myMsg.Configuration.Fields.Item(FieldBase & "smtpserverport") = 25
    myMsg.Configuration.Fields.Update
    myMsg.Send
End Sub
```

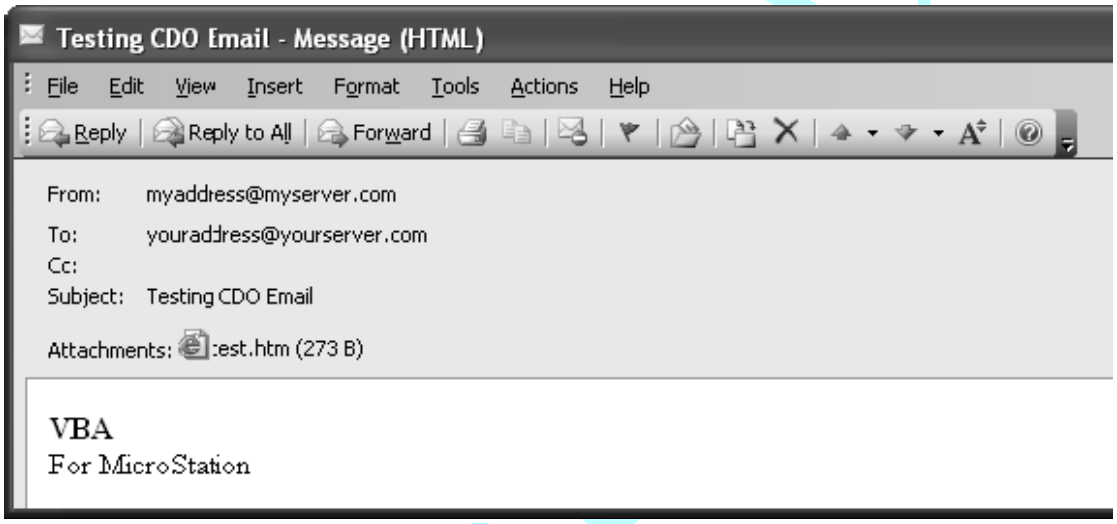
用 VBA 发送邮件很容易。不过还是要用几行代码的，因为需要进行一些设置。一旦设置完成，电子邮件就被发送了。



这是用 CDO 发送的电子邮件。给电子邮件添加附件有多么难？

```
Sub TestCDOB()
    Dim myMsg As New CDO.Message
    Dim FieldBase As String
    FieldBase = "http://schemas.microsoft.com/cdo/configuration/"
    myMsg.To = "youraddress@yourserver.com"
    myMsg.From = "myaddress@myserver.com"
    myMsg.Subject = "Testing CDO Email"
    myMsg.HTMLBody = "<b>VBA</b><br>For MicroStation"
```

```
        myMsg.Configuration.Fields.Item(FieldBase & "sendusing") = 2
        myMsg.Configuration.Fields.Item(FieldBase & "smtpserver") = _
                                                "smtp.yourserver.com"
        myMsg.Configuration.Fields.Item(FieldBase & "smtpserverport") = 25
        myMsg.Configuration.Fields.Update
        myMsg.AddAttachment "C:\test.htm"
        myMsg.Send
    End Sub
```



用 VBA 发送电子邮件有数千种理由。幸运的是发送邮件如此容易。

需要修改 To、From 和 SMTPServer 属性以符合特定的设置和服务器。

DSO OLE Document Properties Reader 2.0

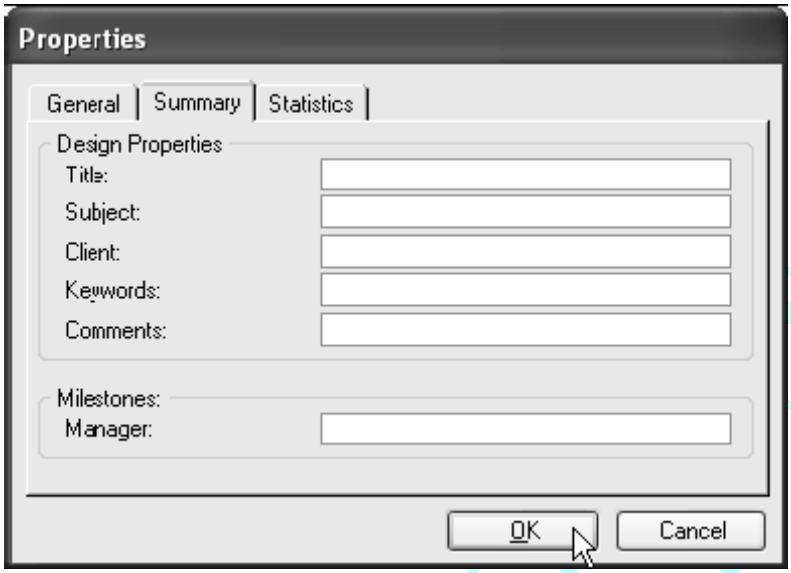
(DSO OLE 文档属性阅读器 2.0)

到目前为止，我们讨论过的所有引用都随 Windows XP 安装，但对 DSO 来说不是这样的。跟随如下链接：

http://support.microsoft.com/kb/224351/

到另一个链接“Download the DsoFileSetup”。这是我们想要的东西，因为 MicroStation 设计文件是 OLE 文档。另一个用于描述这类文件的术语是“结构化存储”文档。该术语是什么意思呢？

在 MicroStation 中，选择 **File > Properties(文件 > 属性)**菜单项，打开文件属性对话框后点击 Summary(摘要)选项卡。



这些属性看起来很像在 Microsoft Word 和 Excel 文件中那样。在设计文件打开时，可以用 VBA 读写在.dgn 文件中的这些属性。DSOFile 引用让我们打开任何结构化存储的文件来读取文件属性或写/建立这些属性，即使建立文件的应用没有安装也可以。让我们看一下。

```
Sub TestDSOA()
    Dim myDSO As New DSOFile.OleDocumentProperties
    myDSO.Open "C:\test.dgn", True
    MsgBox myDSO.SummaryProperties.Author
    myDSO.Close
End Sub
```



文件 **C:\test.dgn** 的作者是“Bentley System,Inc”。不能读或写在应用中打开的文件的属性。

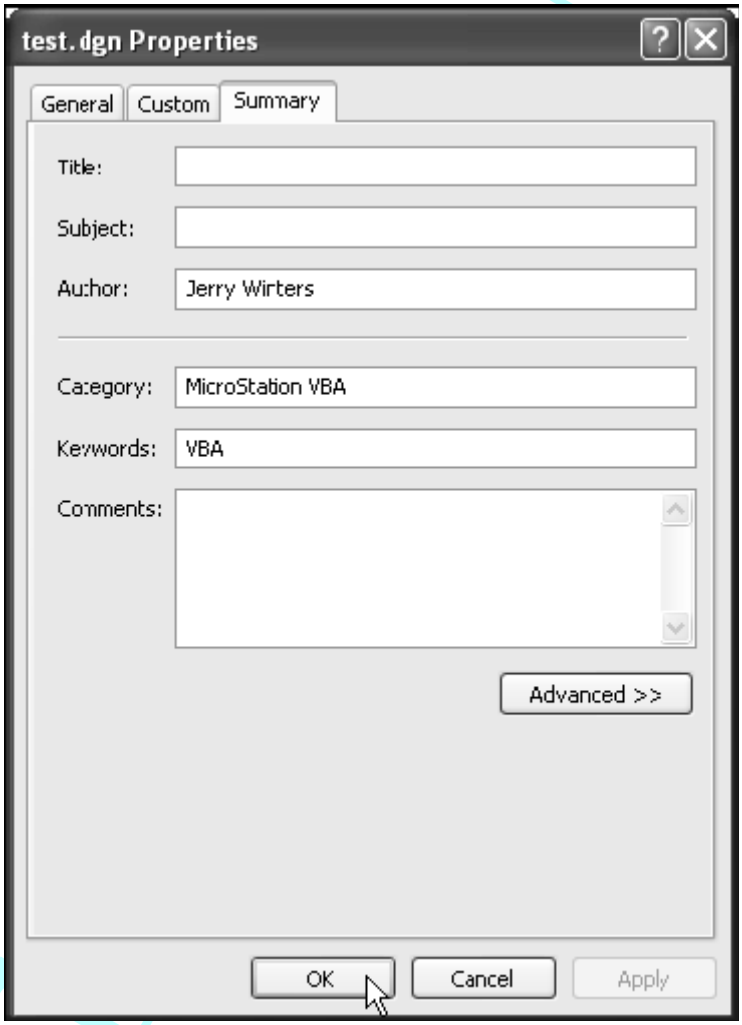
下个例子是写 **test.dgn** 文件的属性。

```
Sub TestDSOB()
    Dim myDSO As New DSOFile.OleDocumentProperties
    myDSO.Open "C:\test.dgn", False
    myDSO.SummaryProperties.Author = "Jerry Winters"
    myDSO.SummaryProperties.Category = "MicroStation VBA"
    myDSO.SummaryProperties.Keywords = "VBA"
```

```
    myDSO.Save
    myDSO.Close
End Sub
```

写属性到一个文件中时，必须在关闭文件前保存。

这是在 Windows 资源管理器属性对话框中看到的结果：



有许多“SummaryProperties(摘要属性)”，比该窗口中显示的多得多。并非所有的属性都用于 MicroStation 的.dgn 文件，MicroStation 不支持有些属性。下一个宏显示每个“SummaryProperties”的值。

```
Sub TestDSOC()
    Dim myDSO As New DSOFile.OleDocumentProperties
    myDSO.Open "C:\test.dgn", True
    Debug.Print myDSO.SummaryProperties.ApplicationName
    Debug.Print myDSO.SummaryProperties.Author
    Debug.Print myDSO.SummaryProperties.ByteCount
```

```
    Debug.Print myDSO.SummaryProperties.Category
    Debug.Print myDSO.SummaryProperties.CharacterCount
    Debug.Print myDSO.SummaryProperties.CharacterCountWithSpaces
    Debug.Print myDSO.SummaryProperties.Comments
    Debug.Print myDSO.SummaryProperties.Company
    Debug.Print myDSO.SummaryProperties.DateCreated
    Debug.Print myDSO.SummaryProperties.DateLastPrinted
    Debug.Print myDSO.SummaryProperties.DateLastSaved
    Debug.Print myDSO.SummaryProperties.HiddenSlideCount
    Debug.Print myDSO.SummaryProperties.Keywords
    Debug.Print myDSO.SummaryProperties.LastSavedBy
    Debug.Print myDSO.SummaryProperties.LineCount
    Debug.Print myDSO.SummaryProperties.Manager
    Debug.Print myDSO.SummaryProperties.MultimediaClipCount
    Debug.Print myDSO.SummaryProperties.NoteCount
    Debug.Print myDSO.SummaryProperties.PageCount
    Debug.Print myDSO.SummaryProperties.ParagraphCount
    Debug.Print myDSO.SummaryProperties.PresentationFormat
    Debug.Print myDSO.SummaryProperties.RevisionNumber
    Debug.Print myDSO.SummaryProperties.SharedDocument
    Debug.Print myDSO.SummaryProperties.SlideCount
    Debug.Print myDSO.SummaryProperties.Subject
    Debug.Print myDSO.SummaryProperties.Template
    Debug.Print myDSO.SummaryProperties.Thumbnail
    Debug.Print myDSO.SummaryProperties.Title
    Debug.Print myDSO.SummaryProperties.TotalEditTime
    Debug.Print myDSO.SummaryProperties.Version
    Debug.Print myDSO.SummaryProperties.WordCount
End Sub
```

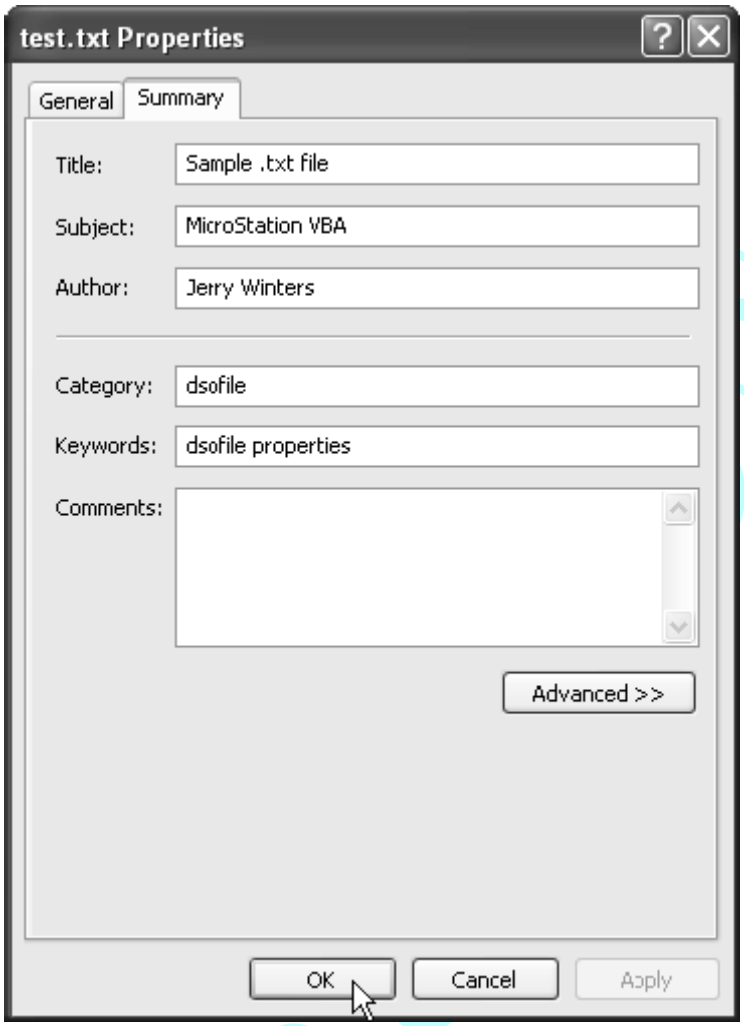
在这里所示的有些摘要属性是只读的。例如，不能修改“DateCreated”属性。

并非所有的文件都是“结构化存储”的文件，如在记事本里建立的.txt 文件。虽然它不是结构化存储的文件，但当其位于 NTFS 硬盘驱动器中时，有些摘要属性仍然可用。要特别留意非结构化存储的文件，它可能看起来已经被输入了文件属性，但通过邮件传送给别人或放入一个非 NTFS 的驱动器上时文件属性没有跟随文件一起过来，因为文件属性没有储存在文件中，而是随文件(实际上在文件之外)一起保存。

```
Sub TestDSOD()
    Dim myDSO As New DSOFile.OleDocumentProperties
    myDSO.Open "C:\test.txt", True
    MsgBox myDSO.SummaryProperties.Author
    myDSO.Close
End Sub
```

TestDSOD 与 TestDSOA 非常相似，只是打开了一个不同的文件。Test.txt 是一个标准的 ASCII 文件。在 Windows 资源管理器中右击该文件并选择“属性”能显示几个选项卡。

China BDN



如果把这个对话框与几页前的相比较，您会发现该对话框中没有一个自定义选项卡。这表明该文件不是一个结构化存储的文件，以上所示的属性是 NTFS 文件系统功能所致。所以能直观地区分出 NTFS 属性与结构化存储属性间的不同。宏 TestDSOD 即使对于非结构化存储的文件也能运行，但如何知道一个文件是 OLE(结构化存储)还是不是呢？

```
Sub TestDSOE()
    Dim myDSO As New DSOFile.OleDocumentProperties
    'First we try a txt file
    myDSO.Open "C:\test.txt", True
    If myDSO.IsOleFile = False Then
        MsgBox "The file is not Structured Storage."
    Else
        MsgBox "Structured Storage File Found."
```

```
        End If
        myDSO.Close
        'Now for the DGN File
        myDSO.Open "C:\test.dgn", True
        If myDSO.IsOleFile = False Then
            MsgBox "The file is not Structured Storage."
        Else
            MsgBox "Structured Storage File Found."
        End If
        myDSO.Close
    End Sub
```

“IsOleFile”属性告诉我们文件是一个 OLE 文档(结构化储存)或一个非 OLE 文档。知道这一点很重要，因为非 OLE 文档在使用属性上有局限性。

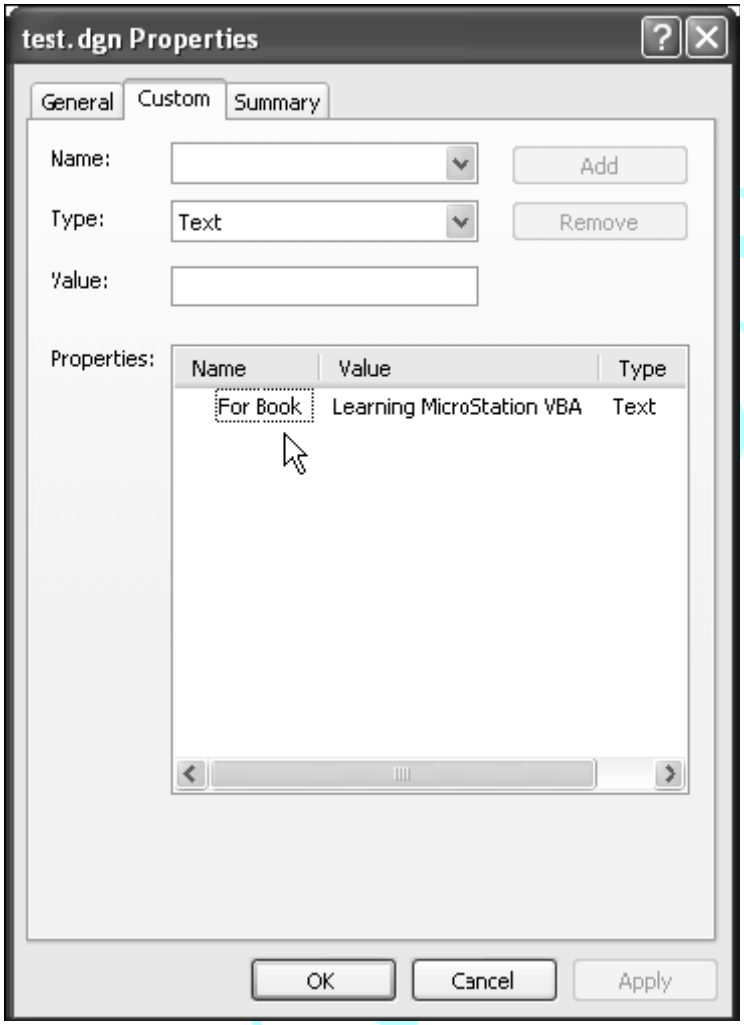
除了摘要属性(标准属性)外，还能建立、读和写自定义属性。

```
    Sub TestDSOF()
        Dim myDSO As New DSOFile.OleDocumentProperties
        myDSO.Open "C:\test.dgn", False
        If myDSO.IsOleFile = True Then
            myDSO.CustomProperties.Add "For Book", _
                                    "Learning MicroStation VBA"
            myDSO.Save
        End If
        myDSO.Close
    End Sub
```

在这个例子中，如果指定的文件是一个 OLE 文件(结构化存储)，就给其添加一个自定义的属性。添加完自定义属性后，该属性就会在 Windows 资源管理器属性对话框的自定义选项卡下显示出来。

添加自定义文件属性的功能很强大，您可能想在设计文件的自定义属性中存储有关单元个数的信息。如果这样做了，VBA 程序(从 MicroStation 内或其它 VBA

环境中)就能读写该属性，即使未安装 MicroStation 也能。



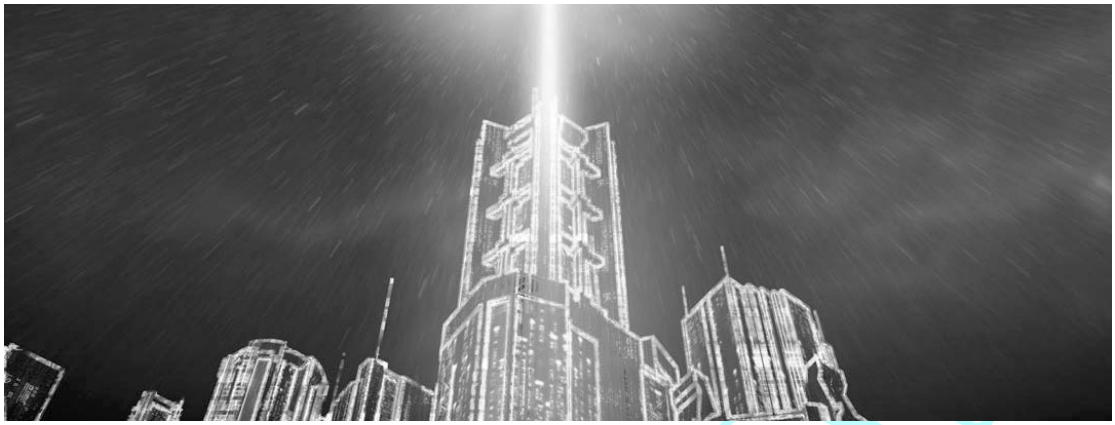
几页前提供有指向 DSOFile.exe 下载的链接，该文件也包含在随本书提供的 CD 中。

有关读写文件属性的更多信息，Microsoft 网站上有其文档和代码样例。在 Internet 上搜索“Dsofile”也能得到大量结果。

本章回顾

VBA 强大功能之一就是，您并不只能使用在 VBA 中直接公开的那些调用，可以使用其他人开发的编程组件来加快开发和增强功能。我们讨论了几个引用，添加它们后能显著增强您开发的软件。

在 VBA 中打开引用对话框添加一个引用，再打开对象浏览器查看引用的内容，这是熟悉您计算机上其它引用所公开的功能的最好方法。



34. 操作 Excel

Microsoft Excel 用途非常广泛，大量的人们在使用它，可能用法各异。许多人用它来作计算，另一些人用它来生成图表，还有一些人用它来平衡开支。

本章的例子是在 MicroStation 的 VBA 环境中编写代码，后面的一章将在 Excel 的环境中编写代码。

本章包括：

- 连接到 Excel
- 工作簿、工作表、范围和单元格
- 提取标签到 Excel 中

连接到 Excel

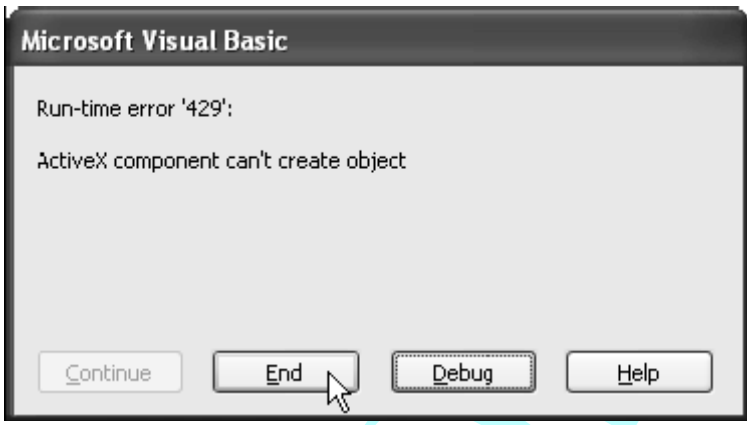
有三种“连接”到 Excel 的方法，从使用“GetObject”开始。

GetObject

```
Sub TestExcelA()
    Dim myExcel As Object
    Set myExcel = GetObject(, "Excel.Application")
End Sub
```

GetObject“取得”一个已经存在的 Microsoft Excel 的实例。

如果 Excel 未运行，当试图“取得”Excel 时将看到一个错误。

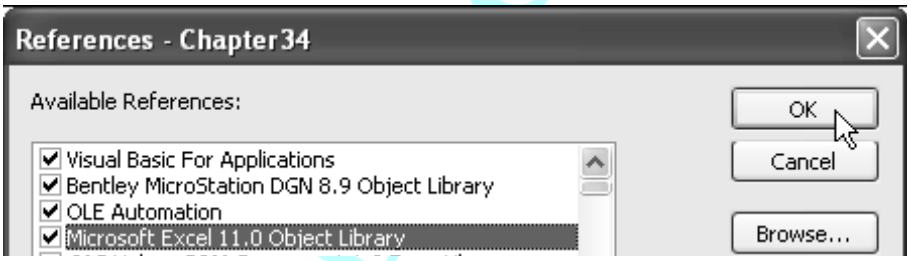


当看到这个错误时，我们知道正试图“取得”Excel 但 Excel 没有运行。如果 Excel 正在运行，宏 TestExcelA 会正常运行，但 TestExcelA 做了什么呢？

我们声明了一个对象变量 **myExcel**，然后把 GetObject 的返回值设置给它。设置完变量 **myExcel** 后，它就是 Microsoft Excel 应用了。对变量 **myExcel** 所做一切动作都将影响 Excel。

当声明一个变量为“Object”时，我们执行的是“后期绑定(Late Binding)”。意思是在设置对象之前，对象不知道它是谁或是什么。当声明一个变量为指定类型的对象时，我们执行的是“前期绑定(Early Binding)”。

添加一个 Microsoft Excel Object Library(对象库)引用确实能降低编程难度。

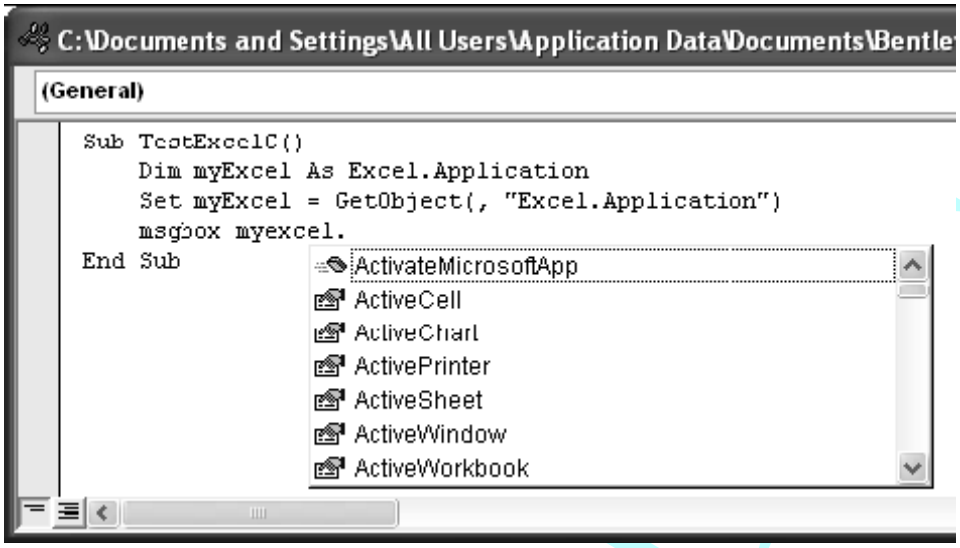


一旦引入一个引用，就能声明变量为特定类型的对象了。

```
Sub TestExcelB()
    Dim myExcel As Excel.Application
    Set myExcel = GetObject(, "Excel.Application")
End Sub
```

这段代码看上去与 TestExcelA 非常相似，其实有巨大的不同，TestExcelB 执行了“前期绑定”。变量 **myExcel** 知道它是一个 Excel.Application，因为它被声明为一个 Excel.Application。除了知道它是什么外，变量 **myExcel** 还知道能做什么。例如，当键入“myexcel”然后按句点键时，将弹出一个列表告诉我们

“myexcel”对象能做什么。从这个变量可以获得当前工作表、当前窗口、当前工作簿以及其它一些 Excel 的内容。



前期绑定不仅因其具有智能性提高了软件开发的速度而且也提高了程序的性能。

CreateObject

假如 Microsoft Excel 没有运行或者我们想建立 Excel.Application 的新实例，可以用 CreateObject。

```
Sub TestExcelD()
    Dim myExcel As Excel.Application
    Set myExcel = CreateObject("Excel.Application")
    myExcel.Visible = True
    myExcel.Workbooks.Add
End Sub
```

TestExcelD 建立了一个 Excel 的新实例，无论 Excel 运行与否，它都建立一个新的 Excel 实例。

在多个 Excel 实例运行的情况下，CreateObject 很有用。使用 GetObject 时，不能事先知道要“取得”的是哪个 Excel 实例。但使用 CreateObject 时，能准确地知道正在使用的是哪个 Excel.Application，因为它是一个新建立的实例。

New

在 VBA 工程中添加了“Excel 库”引用后，VBA 就知道 Excel.Application 对象是什么。如果添加了一个引用，就能用 New 关键字来“建立”一个对象。

```
Sub TestExcelD2()
    Dim myExcel As New Excel.Application
    myExcel.Visible = True
    myExcel.Workbooks.Add
End Sub
```

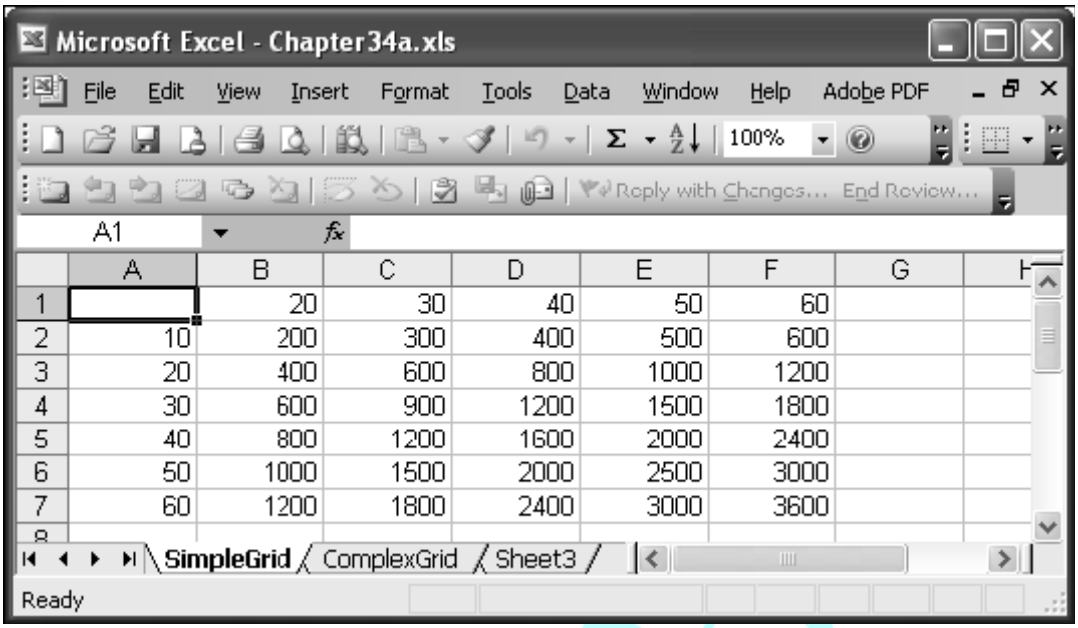
用 New 取代 CreateObject 很有用，因为不需要像在 CreateObject 中那样来提供类“Excel.Application”。不需提供类为什么有用呢？因为公开给 VBA 的应用可能有多个版本，使用 New 关键字将“建立”引用的那个版本。

GetObject、CreateObject 和 New 是用于“连接”到 Microsoft Excel 的方法。当添加了 Microsoft Excel 对象库引用后，就可以声明一个特定类型的变量(前期绑定)。在操作这些变量时，VBA 能为我们执行语法检查并能帮助我们知道可用的属性、方法和事件。

工作簿、工作表、范围和单元格

我们在使用 GetObject、CreateObject 或 New 时，其实是在取得 Excel.Application 对象。该对象下面直接就是工作簿集合(还有其它的内容)。工作簿由工作表组成，工作表由范围和单元格组成。

China BDN



	A	B	C	D	E	F	G
1		20	30	40	50	60	
2	10	200	300	400	500	600	
3	20	400	600	800	1000	1200	
4	30	600	900	1200	1500	1800	
5	40	800	1200	1600	2000	2400	
6	50	1000	1500	2000	2500	3000	
7	60	1200	1800	2400	3000	3600	

这个工作簿有三个工作表，分别叫做 SimpleGrid、ComplexGrid 和 Sheet3。当前正在查看 SimpleGrid。

该 Excel.Application 的 ActiveSheet(当前工作表)是“SimpleGrid”，ActiveCell(当前单元格)具有地址“A1”，其行和列分别是(1,1)。

让我们看一下工作表集合和工作表对象。

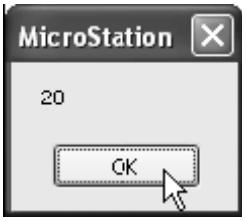
```
Sub TestExcelE()
    Dim myExcel As Excel.Application
    Dim mySheetA As Worksheet
    Dim mySheetB As Worksheet
    Dim mySheetC As Worksheet
    Dim mySheetD As Worksheet
    Set myExcel = GetObject(, "Excel.Application")
    Set mySheetA = myExcel.ActiveSheet
    Set mySheetB = myExcel.ActiveWorkbook.Worksheets("SimpleGrid")
    Set mySheetC = myExcel.ActiveWorkbook.Worksheets("ComplexGrid")
    Set mySheetD = myExcel.ActiveWorkbook.Worksheets("Sheet3")
    Debug.Print mySheetA.Name
    Debug.Print mySheetB.Name
    Debug.Print mySheetC.Name
    Debug.Print mySheetD.Name
```

```
    Set mySheetB = myExcel.ActiveWorkbook.Worksheets(1)
    Set mySheetC = myExcel.ActiveWorkbook.Worksheets(2)
    Set mySheetD = myExcel.ActiveWorkbook.Worksheets(3)
    Debug.Print mySheetB.Name
    Debug.Print mySheetC.Name
    Debug.Print mySheetD.Name
End Sub
```

可以通过取得 ActiveSheet(当前工作表)或按名字或按索引通过工作表集合来定位到工作表。

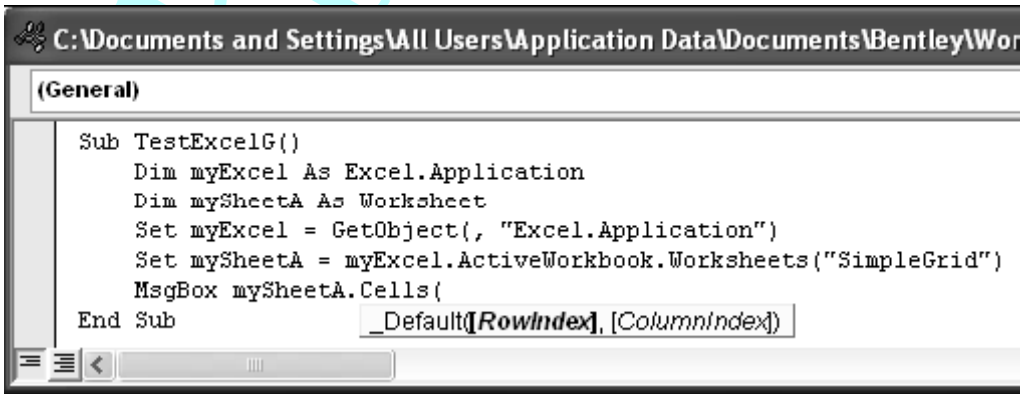
既然能取得工作表了，让我们看一下如何取得单元格。

```
Sub TestExcelF()
    Dim myExcel As Excel.Application
    Dim mySheetA As Worksheet
    Set myExcel = GetObject(, "Excel.Application")
    Set mySheetA = myExcel.ActiveWorkbook.Worksheets("SimpleGrid")
    MsgBox mySheetA.Range("B1").Text
End Sub
```



定位某些单元格的方法之一是用 Range 对象来取得它们。使用 Range 对象时可通过单元格的地址来取得它，如“A1”、“C6”、“F9”等。使用一个像“A4”这样的地址很有用，因为能直接关联到在 Excel 中看到的对应单元格。

另一种操作 Excel 中单元格的方法是使用单元格集合。



在使用单元格集合时，先指定行索引再指定列索引。Excel 中的第 1 行具有

行索引 1，Excel 中的“A”列具有列索引 1。在使用单元格集合时，需要确保行在列之前指定。

```
Sub TestExcelG()
    Dim myExcel As Excel.Application
    Dim mySheetA As Worksheet
    Set myExcel = GetObject(, "Excel.Application")
    Set mySheetA = myExcel.ActiveWorkbook.Worksheets("SimpleGrid")
    MsgBox mySheetA.Cells(4, 6).Text
End Sub
```

这段代码运行后，一个消息框显示了在第 4 行和第 6 列单元格中的文本。

基于行和列来取得单元格不像基于其地址那样容易，那为什么还要麻烦地使用行和列呢？

```
Sub TestExcelH()
    Dim myExcel As Excel.Application
    Dim mySheetA As Worksheet
    Dim CurRow As Long
    Dim CurCol As Long
    Set myExcel = GetObject(, "Excel.Application")
    Set mySheetA = myExcel.ActiveWorkbook.Worksheets("SimpleGrid")
    For CurRow = 1 To 7
        For CurCol = Asc("A") To Asc("F")
            Debug.Print mySheetA.Range(Chr(CurCol) & CurRow)
        Next CurCol
    Next CurRow
End Sub
Sub TestExcelJ()
    Dim myExcel As Excel.Application
    Dim mySheetA As Worksheet
    Dim CurRow As Long
    Dim CurCol As Long
```


China BDN

```
    Set myExcel = GetObject(, "Excel.Application")
    Set mySheetA = myExcel.ActiveWorkbook.Worksheets("SimpleGrid")
    For CurRow = 1 To 7
        For CurCol = 1 To 6
            Debug.Print mySheetA.Cells(CurRow, CurCol)
        Next CurCol
    Next CurRow
End Sub
```

TestExcelH 和 TestExcelJ 都能将一组网格内的单元格值打印到立即窗口中。TestExcelH 能轻易地完成这项任务，因为我们只处理 A 列到 F 列。同样的代码也能操作 A 列到 Z 列，但当处理到“AA”列时会发生什么呢？使用 Range 对象时，用从“A”到“IV”间的任意字母名来指定列。编写从“Z”到“AA”的代码虽然不难但很麻烦。当使用单元格集合时，只需在 26 列完成后简单地指定 27 列而不用担心是否从 Z 列到了 AA 列或 AB 列等等。

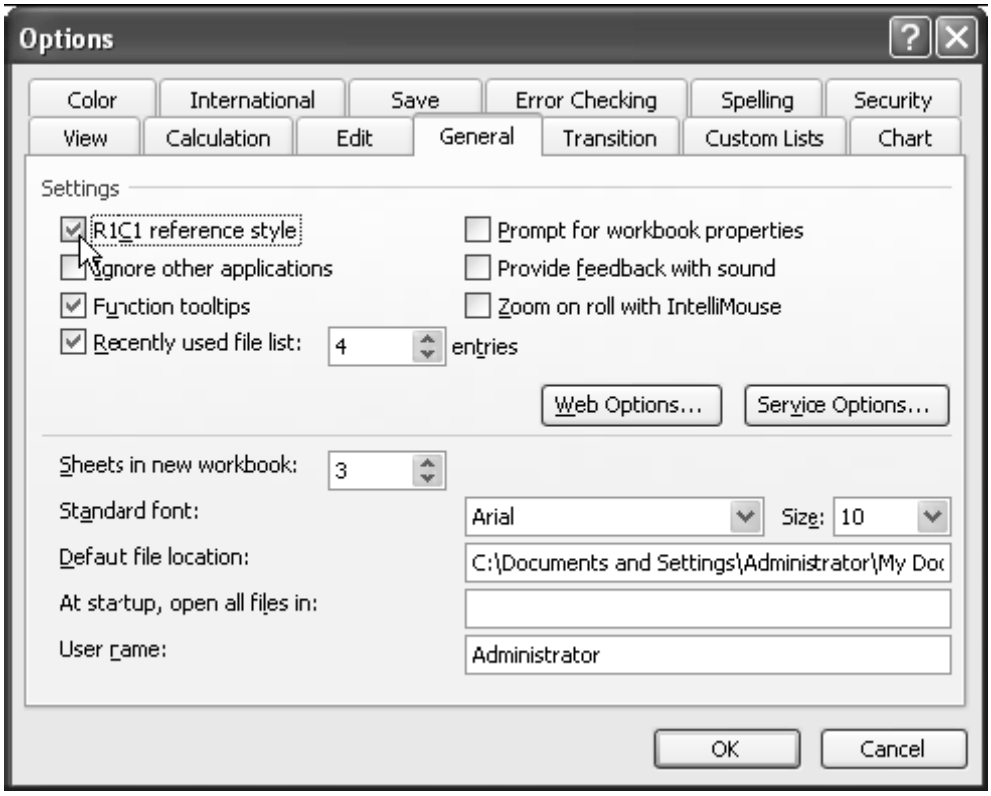
那么，哪种方法最好呢？单元格集合还是范围集合？

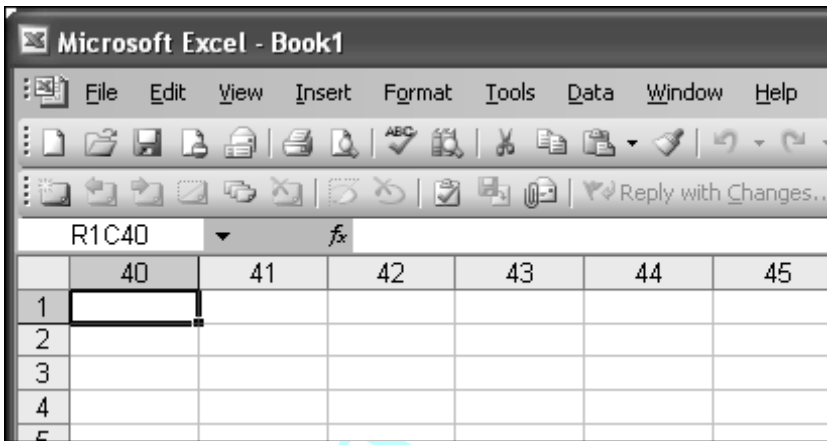
正如我们讨论过的，每种方法都有其强项和弱点。以数字提供行和列很容易，但难于把单元列“翻译”为 Excel 中字母列。设想一下 211 列对应于什么字母列呢？

尤其在处理相对小的数据集(特别是 A 列到 Z 列之间) 时使用范围的方法最好，但这种方法在遇到 AA 列到 IV 列时变得很困难。然而，范围还能由多个单元格(例如从 A1 到 D4)组成，这是使用范围方法的一个明确的优点。

如果使用单元格(对列和行都提供以数字)，通过改变 Microsoft Excel 中的一个设置能对我们有点帮助。

Excel 中的 **Tools > Options(工具 > 选项)**显示选项对话框。点击“常规”选项卡能让我们打开“R1C1 引用样式”。当“R1C1 引用样式”被打开后，Excel 中的列以数字而不是以字母显示。单元格中的公式也被修改成 R1C1 样式，这会使公式看起来很古怪。但对于 Microsoft Excel 编程来说，数字列看起来比字母列要容易得多，使用字母列需要计数“AA”、“AB”、“AC”等才能找到“DC”列对应的列索引。





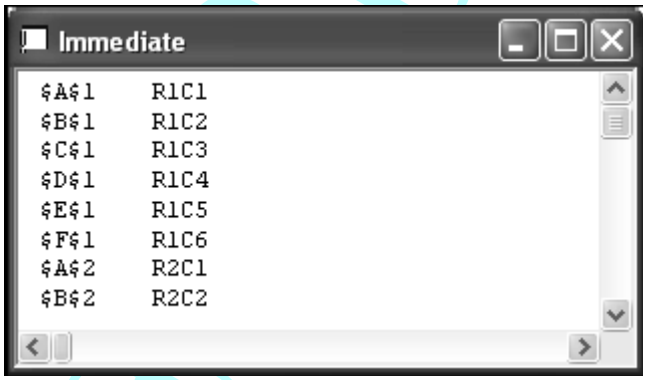
在选择了“R1C1 引用样式”后，列以数字而不是字母出现。

单元格和范围的地址

Excel 中的每个单元格都具有大量的属性。本书仅关心这些属性中的几个，我们建立的查看的第一个属性是 Address。

```
Sub TestExcelL()
    Dim myExcel As Excel.Application
    Dim mySheetA As Worksheet
    Dim CurRow As Long
    Dim CurCol As Long
```

```
    Dim myCell As Excel.Range
    Set myExcel = GetObject(, "Excel.Application")
    Set mySheetA = myExcel.ActiveWorkbook.Worksheets("SimpleGrid")
    For CurRow = 1 To 7
        For CurCol = 1 To 6
            Set myCell = mySheetA.Cells(CurRow, CurCol)
            Debug.Print myCell.Address(True, True, xlA1) & vbTab & _
                        myCell.Address(True, True, xlR1C1)
        Next CurCol
    Next CurRow
End Sub
```



我们能以“xlA1”或“xlR1C1”格式获得单元格(被声明为 Range 对象)的地址。我们刚刚讨论过，“xlA1”格式以字母为列、以数字为行来表示单元格地址。“xlR1C1”格式都以数字为行和列来表示单元格地址，正如运行 TestExcelL 后立即窗口中所示的那样。

正如这里看到的那样，单元格地址以两种方式之一返回。“D2”和“R2C2”指向的是 Excel 中的同一个单元。

了解地址是重要的，因为需要知道用户选择了哪些单元格。例如，如何知道用户已经选择了范围“B2”到“B7”或“D2”到“D5”？

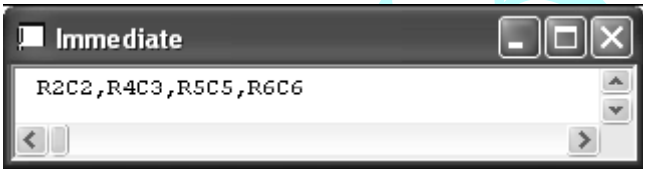
```
Sub TestExcelM()
    Dim myExcel As Excel.Application
    Dim mySelection As Excel.Range
    Set myExcel = GetObject(, "Excel.Application")
    Set mySelection = myExcel.Selection
    Debug.Print mySelection.Address(True, True, xlR1C1)
End Sub
```



在这个例子中，选择集是“B2”到“B7”(2,2 到 7,2)。

冒号(:)告诉我们选择了“B2”到“B7”。在 Excel 中还有其它选择单元格的方法。

通常在 Excel 中选择单元格时，都选择的是相邻的单元格，例如，可能选择“D4”到“G12”。然而，按下**<Ctrl>**键的同时进行选择也能选中多个不相邻的单元格。从下图中能看出，现在选择了四个不同的单元格，B2、C4、E5 和 F6。逗号表明给出的是独立的单元格地址。



下面看到的是选择了范围“B2”到“B5”与“D2”与“D5”。



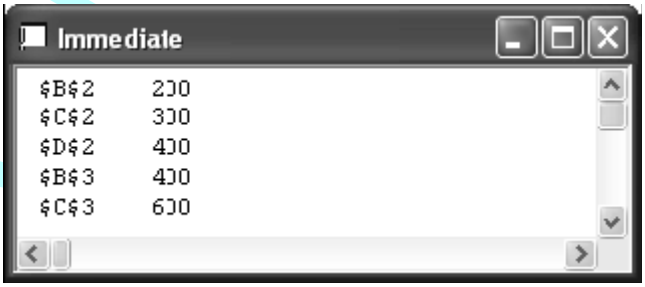
以前的例子中，我们准确地知道要从哪些单元格中取值。下面这个例子用来显示被选择单元格的值。我们假定用户选择了一个单元格范围而不是多个独立的单元格。

```
Sub TestExcelN()
    Dim myExcel As Excel.Application
    Dim mySelection As Excel.Range
    Dim mySheet As Excel.Worksheet
    Dim StartRow As Long: Dim StartCol As Long
    Dim EndRow As Long: Dim EndCol As Long
    Dim X As Long: Dim Y As Long
    Dim XSplitA As Variant: Dim XSplitB As Variant: Dim XSplitC As Variant
    Set myExcel = GetObject(, "Excel.Application")
    Set mySelection = myExcel.Selection
    Set mySheet = myExcel.ActiveSheet
```

```
    XSplitA = Split(mySelection.Address(True, True, xlR1C1), ":")
    XSplitB = Split(XSplitA(0), "C")
    XSplitC = Split(XSplitA(1), "C")
    StartRow = Replace(XSplitB(0), "R", "")
    StartCol = XSplitB(1)
    EndRow = Replace(XSplitC(0), "R", "")
    EndCol = XSplitC(1)
    For Y = StartRow To EndRow
        For X = StartCol To EndCol
            Debug.Print mySheet.Cells(Y, X).Text
        Next X
    Next Y
End Sub
```

在这个例子中，我们找到开始行和开始列以及结束行和结束列。然后查看范围内的每个单元格并将单元格的 Text 属性打印到立即窗口中。

```
Sub TestExcelP()
    Dim myExcel As Excel.Application
    Dim mySelection As Excel.Range
    Set myExcel = GetObject(, "Excel.Application")
    Set mySelection = myExcel.Selection
    Dim myCell As Range
    For Each myCell In mySelection
        Debug.Print myCell.Address & vbTab & myCell.Text
    Next
End Sub
```

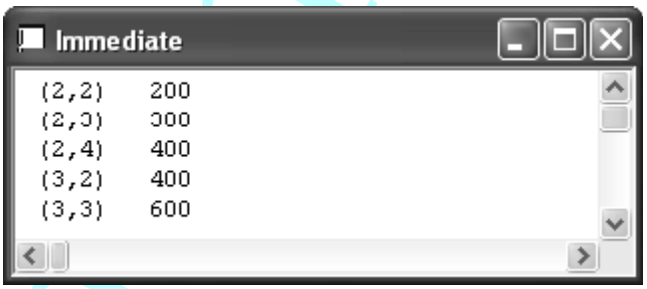


TestExcelN 和 TestExcelP 非常相像，它们完成了同样的任务。在 TestExcelP

中，我们也“提取”了地址。那么，如果两者都做了同样的事情，哪个更好一些呢？行数少的那个不错，知道如何分解地址的那个也不错。每个都有其长处，不一定非要比另一个好，它们各有不同。

下面是另一个宏，它不是把地址提取为“A1”样式，而是提取为(行,列)样式。

```
Sub TestExcelQ()
    Dim myExcel As Excel.Application
    Dim mySelection As Excel.Range
    Dim myCell As Range
    Dim strAdd As String
    Set myExcel = GetObject(, "Excel.Application")
    Set mySelection = myExcel.Selection
    For Each myCell In mySelection
        strAdd = myCell.Address(True, True, xlR1C1)
        strAdd = Replace(strAdd, "R", "")
        strAdd = Replace(strAdd, "C", ",")
        Debug.Print "(" & strAdd & ")" & vbTab & myCell.Text
    Next
End Sub
```



需要记住的是，在处理 Excel 单元格时，我们以(行,列)来查看，行在列的前面。当我们习惯于处理(X,Y)时，需要花一点时间来习惯(Y,X)的形式，因为 Excel 是按(Y,X)来考虑的。

到目前为止，我们从 Excel 中“取出”了许多内容，现在让我们做一些“放入”。我们将改变单元格的值、单元格的公式，然后执行复制和粘贴操作。

```
Sub TestExcelR()
    Dim myExcel As Excel.Application
    Dim mySheet As Worksheet
```

```
    Set myExcel = GetObject(, "Excel.Application")
    Set mySheet = myExcel.ActiveSheet
    'Give B1 a new Value
    mySheet.Range("B1").Value = 70
    'Give B2 a new Formula
    mySheet.Range("B2").Formula = "=B$1*$A2*52"
    'Copy B2 to the Windows Clipboard
    mySheet.Range("B2").Copy
    'Select B2 through F7
    mySheet.Range("B2", "F7").Select
    'Paste copied formula to selected cells
    mySheet.Paste
    'Select B2
    mySheet.Range("B2").Select
    'Reset Cut/Copy Mode
    myExcel.CutCopyMode = False
End Sub
```

这段代码很直接而简单。在对 Excel 文件进行修改的每行代码前都有注释。TestExcelR 对“ActiveSheet(当前工作表)”进行改变。

操作工作表

我们知道，一个工作簿中能有多个工作表。让我们看一个操作多个工作表的例子。

```
Sub TestExcelS()
    Dim myExcel As Excel.Application
    Dim myWB As Excel.Workbook
    Dim myWSA As Excel.Worksheet
    Dim myWSB As Excel.Worksheet
    Dim myWSC As Excel.Worksheet
    Dim RadValue As Double
```

China BDN

```
Dim InchValue As Double
Dim FeetValue As Double
Dim CurRow As Long
Set myExcel = GetObject(, "Excel.Application")
Set myWB = myExcel.ActiveWorkbook
Set myWSA = myWB.Worksheets("Sheet1")
Set myWSB = myWB.Worksheets("Sheet2")
Set myWSC = myWB.Worksheets("Sheet3")
'Merge Header Rows
myWSA.Range("A1", "D1").Merge True
myWSB.Range("A1", "D1").Merge True
myWSC.Range("A1", "D1").Merge True
'Add Titles
myWSA.Range("A1").Value = "Degrees"
myWSA.Range("A1").HorizontalAlignment = xlCenter
myWSB.Range("A1").Value = "Inches"
myWSB.Range("A1").HorizontalAlignment = xlCenter
myWSC.Range("A1").Value = "Feet"
myWSC.Range("A1").HorizontalAlignment = xlCenter
'Add Values
myWSA.Range("A2").Value = "Degrees"
myWSA.Range("B2").Value = "Radians"
CurRow = 3
For RadValue = 0 To 360 Step 5
    myWSA.Range("A" & CurRow) = RadValue
    myWSA.Range("B" & CurRow).Value = RadValue * Atn(1) * 4 / 180
    CurRow = CurRow + 1
Next RadValue
CurRow = 3
myWSB.Range("A2") = "Inch"
```

```
    myWSB.Range("B2") = "Centimeter"
    For InchValue = 1 To 36
        myWSB.Range("A" & CurRow) = InchValue
        myWSB.Range("B" & CurRow).Formula = "=A" & CurRow & " * 2.54"
        CurRow = CurRow + 1
    Next InchValue
    CurRow = 3
    myWSC.Range("A2") = "Feet"
    myWSC.Range("B2") = "Miles"
    For FeetValue = 0 To 20000 Step 1000
        myWSC.Range("A" & CurRow) = FeetValue
        myWSC.Range("B" & CurRow).Formula = "=A" & CurRow & " / 5280"
        CurRow = CurRow + 1
    Next FeetValue
End Sub
```

TestExcelS 使用同一个工作簿中的三个不同的工作表。在使用实际的工作表之前不需要切换到它，只需简单地按名字(Sheet1、Sheet2、Sheet3)定位即可。

Excel 用于保存大量的信息，其中之一是用来保存点的 X、Y 和 Z 坐标。让我们看第一个例子。

我们使用 Excel 函数 RAND 来生成随机数，它能帮助我们生成在一定区域内的点。

下面是其公式。

China BDN

	A	B	C	D
1	X	Y	Z	Level
2	=RAND()*(-100)+50	=RAND()*(-100)+50	=RAND()*(-50)+25	1
3	=RAND()*(-100)+50	=RAND()*(-100)+50	=RAND()*(-50)+25	2
4	=RAND()*(-100)+50	=RAND()*(-100)+50	=RAND()*(-50)+25	3
5	=RAND()*(-100)+50	=RAND()*(-100)+50	=RAND()*(-50)+25	2
6	=RAND()*(-100)+50	=RAND()*(-100)+50	=RAND()*(-50)+25	1
7	=RAND()*(-100)+50	=RAND()*(-100)+50	=RAND()*(-50)+25	2
8	=RAND()*(-100)+50	=RAND()*(-100)+50	=RAND()*(-50)+25	3
9	=RAND()*(-100)+50	=RAND()*(-100)+50	=RAND()*(-50)+25	2
10	=RAND()*(-100)+50	=RAND()*(-100)+50	=RAND()*(-50)+25	1
11	=RAND()*(-100)+50	=RAND()*(-100)+50	=RAND()*(-50)+25	2
12	=RAND()*(-100)+50	=RAND()*(-100)+50	=RAND()*(-50)+25	3
13	=RAND()*(-100)+50	=RAND()*(-100)+50	=RAND()*(-50)+25	2
14	=RAND()*(-100)+50	=RAND()*(-100)+50	=RAND()*(-50)+25	1
15	=RAND()*(-100)+50	=RAND()*(-100)+50	=RAND()*(-50)+25	2
16	=RAND()*(-100)+50	=RAND()*(-100)+50	=RAND()*(-50)+25	3
17	=RAND()*(-100)+50	=RAND()*(-100)+50	=RAND()*(-50)+25	2
18	=RAND()*(-100)+50	=RAND()*(-100)+50	=RAND()*(-50)+25	1
19	=RAND()*(-100)+50	=RAND()*(-100)+50	=RAND()*(-50)+25	2
20	=RAND()*(-100)+50	=RAND()*(-100)+50	=RAND()*(-50)+25	3
21	=RAND()*(-100)+50	=RAND()*(-100)+50	=RAND()*(-50)+25	2
22	=RAND()*(-100)+50	=RAND()*(-100)+50	=RAND()*(-50)+25	1
23	=RAND()*(-100)+50	=RAND()*(-100)+50	=RAND()*(-50)+25	2
24	=RAND()*(-100)+50	=RAND()*(-100)+50	=RAND()*(-50)+25	3
25	=RAND()*(-100)+50	=RAND()*(-100)+50	=RAND()*(-50)+25	2
26	=RAND()*(-100)+50	=RAND()*(-100)+50	=RAND()*(-50)+25	1

X 和 Y 值设置为在-50 和 50 之间的随机数，Z 值为-25 和 25 之间的随机数。

每当工作表重新计算时，RAND 函数就重新计算这些值。这些值不是固定的，一个人的值与另一个人的值不一样。

既然看过了公式，让我们看一下生成的值。

China

	A	B	C	D	E
1	X	Y	Z	Level	
2	-2.208716598	33.06775051	10.80048308	1	
3	-6.328729698	-33.55681358	-15.12641372	2	
4	34.56159917	-10.503424	6.740530447	3	
5	-4.622206053	28.07424377	11.58952008	2	
6	-29.87002472	-42.94468857	-18.03490077	1	
7	41.69913054	-25.40802339	17.79778593	2	
8	18.51114764	0.302771807	-12.71775582	3	
9	-43.30360124	-4.219122549	17.52240842	2	
10	27.57192593	36.50908479	5.833780353	1	
11	27.82085713	-1.916510652	16.59443405	2	
12	47.68307758	22.11381557	-8.423401139	3	
13	-49.79771456	40.82086309	-1.097469319	2	
14	-28.44507751	11.91563398	23.67553777	1	
15	15.35135842	17.71578675	22.46271813	2	
16	16.21397367	1.407118929	-5.470716498	3	
17	49.45543868	45.61203874	16.87426187	2	
18	-36.01242886	24.27163771	-5.996325708	1	
19	34.43815527	14.51513287	-12.31695913	2	
20	-25.49842919	12.18440165	-4.6610566	3	
21	-13.442982	-39.63885911	-14.34005614	2	
22	-4.035505243	-15.54321707	-7.858884602	1	
23	-18.46141973	38.73125351	-19.68034631	2	
24	37.29410466	-33.84766375	-11.24281818	3	
25	-6.704128391	-18.2653331	-2.96787557	2	
26	1.553724653	2.912092628	4.542273424	1	

对于这个例子，行数不固定，可以从 1 行到 65536 行。我们将要用的代码从第 2 行开始查看直到找到一个 A 列为空的行为止。

```
Sub TestExcelT()
    Dim myExcel As Excel.Application
    Dim myWSA As Excel.Worksheet
    Dim CurRow As Long
    CurRow = 2
    Set myExcel = GetObject(, "Excel.Application")
    Set myWSA = myExcel.ActiveWorkbook.Worksheets("Sheet1")
    While myWSA.Cells(CurRow, 1) <> ""
        Debug.Print myWSA.Cells(CurRow, 1) & ", " & _
                    myWSA.Cells(CurRow, 2) & ", " & _
                    myWSA.Cells(CurRow, 3)
        CurRow = CurRow + 1
```

```
    Wend
    myWSA.Calculate
End Sub
```

如果想提取特定的行数，可以使用一个 For I … Next 结构。但由于行数可能会变化，我们使用一个 While … Wend 结构。

代码的最后一行强制工作表重新计算，它能生成新的随机数。

让我们建立 TestExcelT，该例子在 MicroStation 中绘制生成的点。

```
Sub TestExcelU()
    Dim myExcel As Excel.Application
    Dim myWSA As Excel.Worksheet
    Dim CurRow As Long
    Dim myPoint As Point3d
    Dim myTextNode As TextNodeElement
    Dim myRotMatrix As Matrix3d
    CurRow = 2
    Set myExcel = GetObject(, "Excel.Application")
    Set myWSA = myExcel.ActiveWorkbook.Worksheets("Sheet1")
    While myWSA.Cells(CurRow, 1) <> ""
        myPoint.X = myWSA.Cells(CurRow, 1)
        myPoint.Y = myWSA.Cells(CurRow, 2)
        myPoint.Z = myWSA.Cells(CurRow, 3)
        Set myTextNode = CreateTextNodeElement1(Nothing, myPoint, _
                                                myRotMatrix)
        myTextNode.AddTextLine CurRow - 2
        ActiveModelReference.AddElement myTextNode
        CurRow = CurRow + 1
    Wend
    myWSA.Calculate
End Sub
```

China BDN

工作表中有一列“Level”，上例中没有用到它。现在让我们改进 TestExcelU，把文件节点放置到特定的层上。

如果想在不存在的层上放置元素将会得到一个错误。让我们建立一个新函数叫做 CheckLevel，它在层不存在的情况下建立该层。我们将在 TestExcelV 中使用这个函数。

```
Function CheckLevel(LevelName As String) As Level
    On Error Resume Next
    Set CheckLevel = ActiveDesignFile.Levels(LevelName)
    If Err.Number <> 0 Then
        Set CheckLevel = ActiveDesignFile.AddNewLevel(LevelName)
    End If
    Err.Clear
End Function
Sub TestExcelV()
    Dim myExcel As Excel.Application
    Dim myWSA As Excel.Worksheet
    Dim CurRow As Long
    Dim myPoint As Point3d
    Dim myTextNode As TextNodeElement
    Dim myRotMatrix As Matrix3d
    CurRow = 2
    Set myExcel = GetObject(, "Excel.Application")
    Set myWSA = myExcel.ActiveWorkbook.Worksheets("Sheet1")
    While myWSA.Cells(CurRow, 1) <> ""
        myPoint.X = myWSA.Cells(CurRow, 1)
        myPoint.Y = myWSA.Cells(CurRow, 2)
        myPoint.Z = myWSA.Cells(CurRow, 3)
        Set myTextNode = CreateTextNodeElement1(Nothing, _
                                                myPoint, myRotMatrix)
        myTextNode.AddTextLine CurRow - 2
```

```
        CheckLevel CStr(myWSA.Cells(CurRow, 4))
        myTextNode.Level = CheckLevel(myWSA.Cells(CurRow, 4))
        ActiveModelReference.AddElement myTextNode
        CurRow = CurRow + 1
    Wend
    myWSA.Calculate
End Sub
```

注意函数 CheckLevel 是如何返回一个层对象的。建立完文本节点后使用了这个层对象，所以文本节点出现在了正确的层上。

标签提取

在前面一章讨论标签时，曾说过要看一个将标签信息提取到 Microsoft Excel 中的例子。我们将从修改宏 ExportFolderTagsToHTML 开始，这个宏建立了一个 HTML 文档，该文档显示指定文件夹下文件的标签信息。

```
Sub TestExcelW()
    Dim myDGN As DesignFile
    Dim myFSO As New Scripting.FileSystemObject
    Dim myFolder As Scripting.Folder
    Dim myFile As Scripting.File
    Dim myTagSet As TagSet
    Dim myTagDef As TagDefinition
    Dim TargetTagset As String
    Dim myTag As TagElement
    Dim myElemEnum As ElementEnumerator
    Dim myFilter As New ElementScanCriteria
    'New Declarations
    Dim myExcel As Excel.Application
    Dim myWS As Excel.Worksheet
    Dim CurRow As Long
    'New Code
```

China BDN

```
Set myExcel = New Excel.Application
myExcel.Visible = True
myExcel.Workbooks.Add
Set myWS = myExcel.ActiveSheet
CurRow = 2
TargetTagset = "TitleBlock"
Set myFolder = myFSO.GetFolder("C:\Documents and Settings\" & _
                               "All Users\Application Data\" & _
                               "Bentley\WorkSpace\Projects\" & _
                               "Examples\Building\Dgn")
For Each myFile In myFolder.Files
    Select Case myFile.Type
    Case "Bentley MicroStation Design File"
        'File Name and Merge Cells
        myWS.Cells(CurRow, 1) = myFile.Path
        myWS.Range("A" & CurRow & ":F" & CurRow).MergeCells = True
        myWS.Range("A" & CurRow, "F" & CurRow).BorderAround , xlThick
        'Header
        CurRow = CurRow + 1
        myWS.Range("B" & CurRow) = "Tag Set Name"
        myWS.Range("B" & CurRow).Font.Bold = True
        myWS.Range("C" & CurRow) = "Tag Name"
        myWS.Range("C" & CurRow).Font.Bold = True
        myWS.Range("D" & CurRow) = "Tag Value"
        myWS.Range("D" & CurRow).Font.Bold = True
        myWS.Range("E" & CurRow) = "ID High"
        myWS.Range("E" & CurRow).Font.Bold = True
        myWS.Range("F" & CurRow) = "ID Low"
        myWS.Range("F" & CurRow).Font.Bold = True
        CurRow = CurRow + 1
```


China BDN

```
            'Open the File
            Set myDGN = Application.OpenDesignFileForProgram( _
                                                myFile.Path, True)
            For Each myTagSet In myDGN.TagSets
                Select Case UCase(myTagSet.Name)
                Case UCase(TargetTagset)
                    myFilter.ExcludeAllTypes
                    myFilter.IncludeType msdElementTypeTag
                    Set myElemEnum = myDGN.Models(1).Scan(myFilter)
                    While myElemEnum.MoveNext
                        Set myTag = myElemEnum.Current
                        'Write to Excel
                        myWS.Cells(CurRow, 2) = TargetTagset
                        myWS.Cells(CurRow, 3) = myTag.TagDefinitionName
                        myWS.Cells(CurRow, 4) = myTag.Value
                        myWS.Cells(CurRow, 5) = myTag.ID.High
                        myWS.Cells(CurRow, 6) = myTag.ID.Low
                        CurRow = CurRow + 1
                    Wend
                End Select
            Next
            myDGN.Close
        End Select
    Next
End Sub
```

这段代码与 28 章的那段非常相似，以“ForProgram”打开每个文件，从文件的标签中提取标题框信息。我们还做了一点格式化的工作，把保存文件名的几个单元格合并起来并在其周围绘制一个边框，还把“标题”的 font.Bold 属性改为 True。

China BDN

	A	B	C	D	E	F
1						
2	C:\Documents and Settings\All Users\Application Data\Bentley\WorkSpace\Projects\Examples\Building\Dgn\BSI300AE101-Plan.dgn					
3		Tag Set Name	Tag Name	Tag Value	ID High	ID Low
4	C:\Documents and Settings\All Users\Application Data\Bentley\WorkSpace\Projects\Examples\Building\Dgn\BSI300AE101-Plan_Kitchen_a					
5		Tag Set Name	Tag Name	Tag Value	ID High	ID Low
6	C:\Documents and Settings\All Users\Application Data\Bentley\WorkSpace\Projects\Examples\Building\Dgn\BSI300AE101-Plan_Training.d					
7		Tag Set Name	Tag Name	Tag Value	ID High	ID Low
8	C:\Documents and Settings\All Users\Application Data\Bentley\WorkSpace\Projects\Examples\Building\Dgn\BSI300AE201-Elevations.dgn					
9		Tag Set Name	Tag Name	Tag Value	ID High	ID Low
10		TitleBlock	Design File Ref		0	36089
11		TitleBlock	Project File Ref		0	36090
12		itleBlock	Date		0	36091
13		TitleBlock	Checked By		0	36092
14		TitleBlock	DRG. NO.	AE201	0	36094
15		TitleBlock	Rev.	1	0	36095
16		TitleBlock	Job No	BSI300	0	36096
17		TitleBlock	Scale	0.111111111	0	36097
18		TitleBlock	Title2	Facade Treatment Options	0	36151
19		TitleBlock	Title	South Elevation and	0	36152
20	C:\Documents and Settings\All Users\Application Data\Bentley\WorkSpace\Projects\Examples\Building\Dgn\BSI300AE301-Sections.dgn					

取得标签名及其值在很多方面都很有用，但取得 ID 值(High 和 Low 两个值)也很有用。既然标签信息存在于 Excel 中，我们就能改变 Excel 中的标签然后运行一个宏来更新.dgn 文件。

	A	B	C	D	E	F
8	C:\Documents and Settings\All Users\Application Data\Bentley\WorkSpace\Projects\Examples\Building\Dgn\BSI300AE201-Elevations.dgn					
9		Tag Set Name	Tag Name	Tag Value	ID High	ID Low
10		TitleBlock	Design File Ref		0	36089
11		TitleBlock	Project File Ref		0	36090
12		TitleBlock	Date		0	36091
13		TitleBlock	Checked By		0	36092
14		TitleBlock	DRG. NO.	AE201	0	36094
15		TitleBlock	Rev.	1	0	36095
16		TitleBlock	Job No	BSI300A	0	36096
17		TitleBlock	Scale	0.111111111	0	36097
18		TitleBlock	Title2	Facade Treatment Options	0	36151
19		TitleBlock	Title	South Elevation and	0	36152

例如，如果工作号从“BSI300”变成了“BSI300A”，我们可以在 Excel 中作出改变，然后运行宏 TestExcelX 来更新 BSI300AE301-Elevations.dgn 文件。

```
Sub TestExcelX()
    Dim myDGN As DesignFile
    Dim myTag As TagElement
    Dim myExcel As Excel.Application
    Dim myWS As Excel.Worksheet
    Dim CurRow As Long
    Dim FileRow As Long
    Set myExcel = GetObject(, "Excel.Application")
```

```
    Set myWS = myExcel.ActiveSheet
    CurRow = myExcel.ActiveCell.Row
    FileRow = CurRow
    While myWS.Cells(FileRow, 1).MergeCells = False
        FileRow = FileRow - 1
    Wend
    Set myDGN = Application.OpenDesignFileForProgram( _
                                        myWS.Cells(FileRow, 1), False)
    Dim myID As DLong
    myID.High = myWS.Cells(CurRow, 5)
    myID.Low = myWS.Cells(CurRow, 6)
    Set myTag = myDGN.GetElementByID(myID)
    myTag.Value = myWS.Cells(CurRow, 4)
    myTag.Rewrite
    myDGN.Save
    myDGN.Close
End Sub
```

一旦从 MicroStation 中提取了标签，就想让用户在 Excel 中改变其值。Excel 中当前选中的单元格很重要，因为它告诉我们要修改哪个标签。

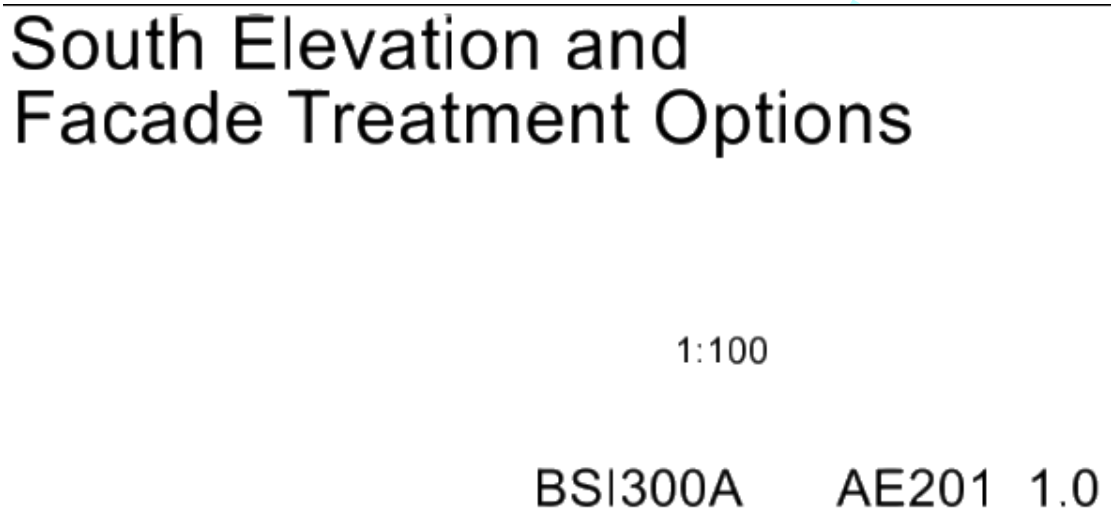
首先检索到 Excel 中当前被选单元格所在的行，一直往上找直到发现 A 列(第 1 列)被合并的行。当找到合并的列时就等于是找到了存储文件名的行，这就是选中的标签(实际上是在 Excel 中选中的单元格)所在的文件。我们打开这个文件，用 ID.High 和 ID.Low 值取得标签元素，按 Excel 中的内容改变标签值，重写标签元素到 DGN 文件中，然后保存并关闭 DGN 文件。所有这一切都在很短一段时间内完成，用户看不到 DGN 文件的打开，因为使用了 OpenDesignFileForProgram。

假如打开 DGN 文件来看的话，能看到标签元素已经按 Excel 中的值被修改。

这些宏功能很强大，因为它们能让我们从 MicroStation 文件中提取数据，不使用 MicroStation 而能随时“修改”这些数据，然后又把修改“上载”回

MicroStation 文件中。

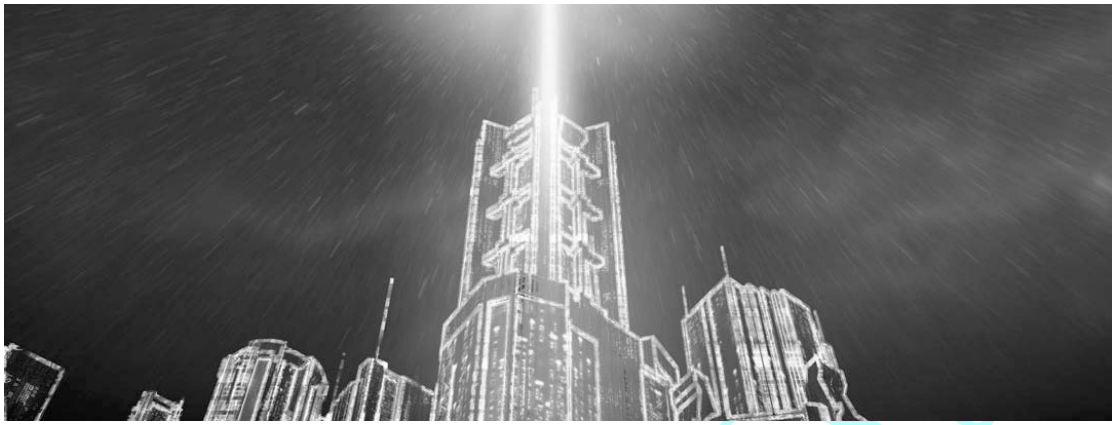
我们能改进这个宏来上载 Excel 文件中的所有标签。但这个任务就留给读者来完成吧。



本章回顾

我们讨论过的关于元素建立、数据提取等任何内容都能与 Microsoft Excel 相关联。我们已经使用 Excel 从 MicroStation 中提取数据、在 MicroStation 中建立数据以及修改 MicroStation 中的数据。那些把 Microsoft Excel 与 MicroStation 关联起来使用的人们将会发现，许多手工的、耗时的、乏味的、易错的任务都能通过这两项卓越技术的完美结合来完成。

China



35. 操作数据库(ActiveX 数据对象)

Access、Oracle、DB2、SQL Server。只要有要存储的数据，用数据库平台来存储它保准没有坏处。与其在每种数据库上花费时间，不如把时间都花在 ActiveX 数据对象的学习上，因为用 ActiveX 数据对象能与这些数据库产品中的每一种进行“对话”。

本章包括：

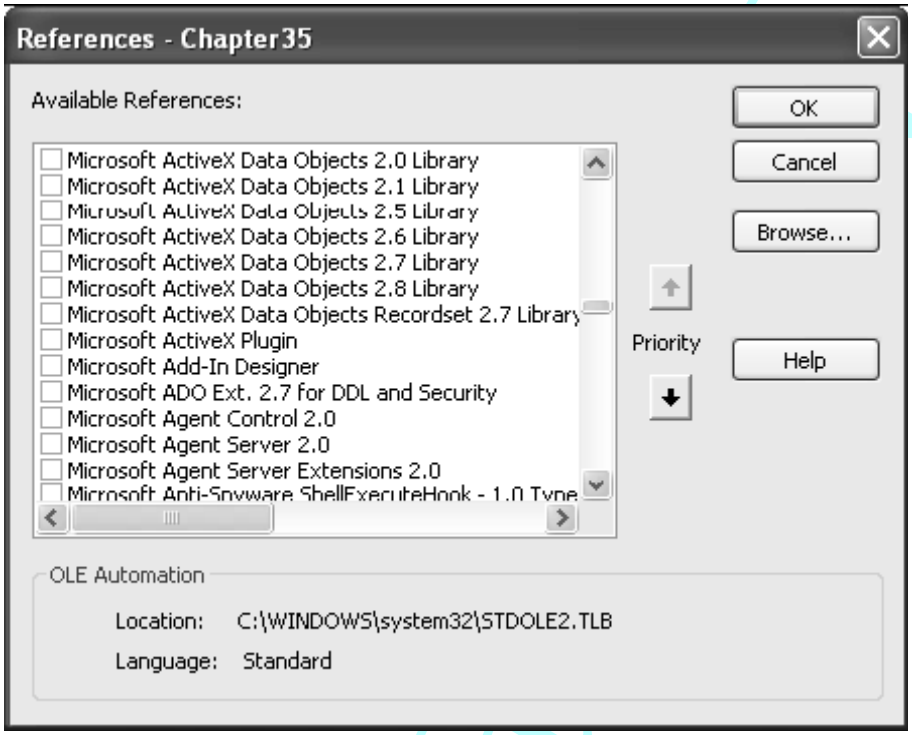
- ActiveX 数据对象入门
- UDL 文件基础
- 连接和记录集
- 结构化查询语言要素
- 扩展 ActiveX 数据对象
- 分析数据库架构(Schema)
- Excel 文件作为数据库

ActiveX 数据对象入门

ActiveX 数据对象是由微软创立的一项技术，它能既简单又强大地与数据库通讯。哪种数据库？事实上是所有“开放”的数据库。“开放”又是什么意思？Microsoft Access 是“开放”的，SQL Server 是“开放”的，Oracle 是“开放”的，主流的数据库都是“开放”的。我们能通过标准的 VBA/ADO(ActiveX 数据对象)调用来操作它们。然而，有些数据库是私有的，其设计之初就不允许软件开发者随意来操作它们。总之，大多数可用的数据库都能通过 ActiveX 数据对象来

访问。

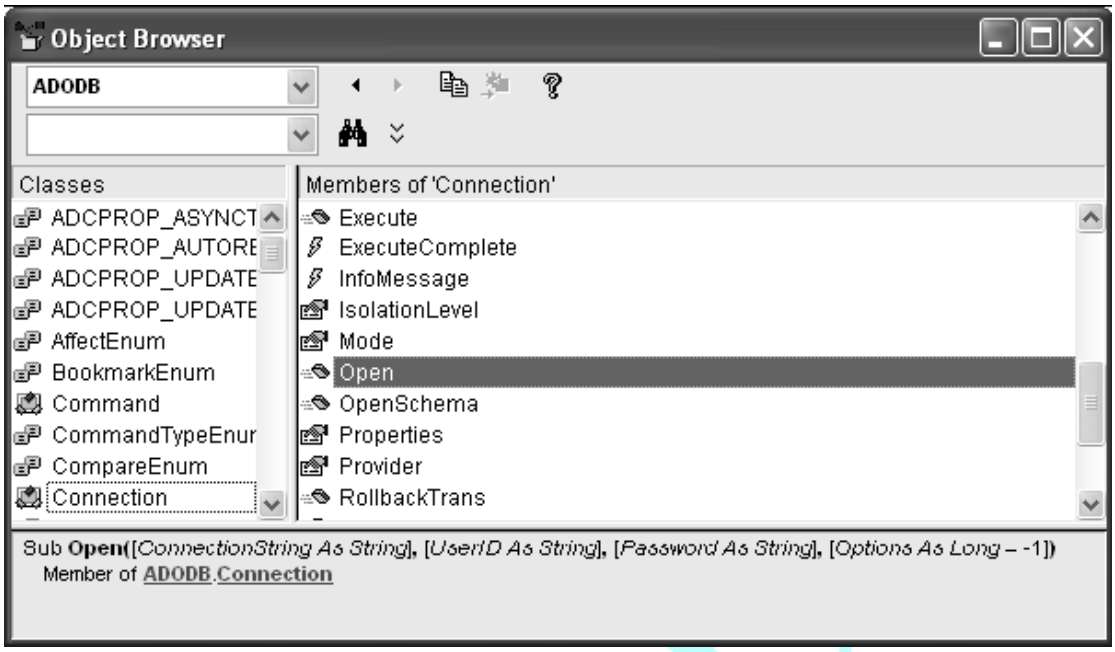
ActiveX 数据对象能让我们用几个关键的对象来操作数据库。在使用“ActiveX 数据对象”前，必须添加其引用，让我们现在就这样做。



当在 VBA 中查看引用对话框时，可能会看到多个不同的“ActiveX 数据对象”库。通常选择最高的库版本最好，本例中 2.8 版是最高的版本。

添加完““Microsoft ActiveX Data Objects 2.8 Library”引用后，新的对象和常数被公开并可为我们所用。可以在对象浏览器中查看它们，过滤条件为“ADODB”库。

选择“Connection”对象显示其方法、属性和事件。



选择“Open”方法时能看到 Open 的声明。

几个值得分析一会儿的对象是“Connection”、“RecordSet”和“Field”。我们将使用这些对象及其方法和属性。查找像 Open、Update、Execute 和 AddNew 这样的方法。

简单地说，ActiveX 数据对象让我们打开数据库、查询记录、修改记录、添加新记录等。然而，在用 ActiveX 数据对象做更多事情之前，需要有一个数据库。

USGS(美国地质勘探局)维护着一个叫做“地理名称信息系统(GNIS)”的系统。来自几个州的地理名称已经被导入到一个叫做 PlacePoints.mdb 的 Microsoft Access 数据库中，该文件位于随本书附送的 CD 上。这是本章要用到的第一个数据库。

下面是这个数据库中一些数据的屏幕拷贝，注意其字段(列)和每一记录(行)。

Points : Table

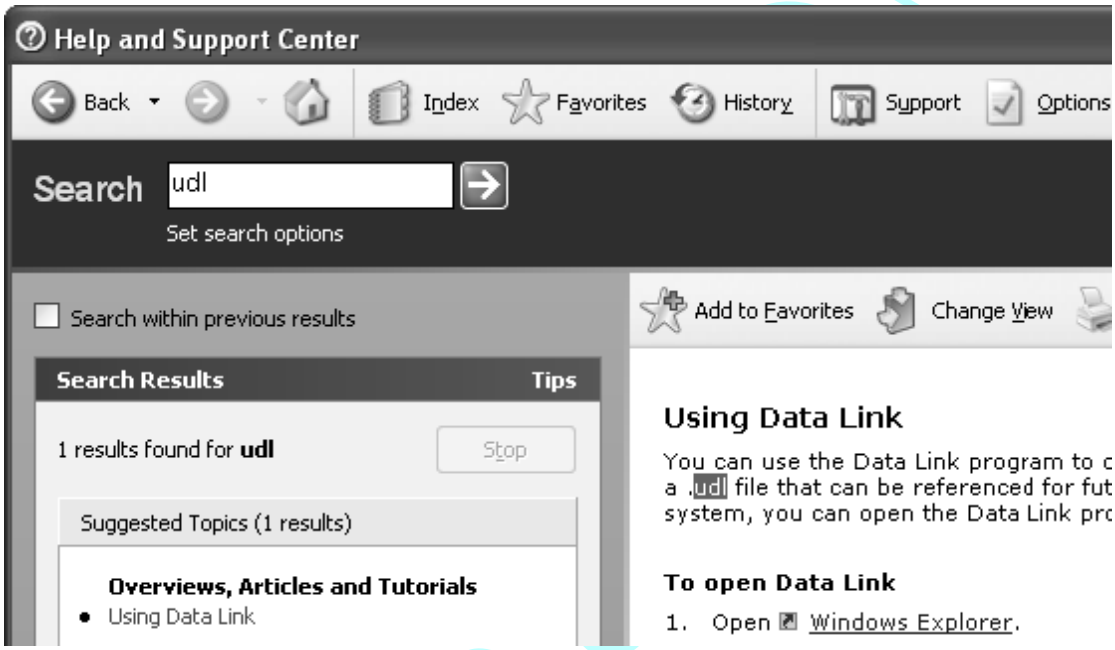
UniqueID	USGS_ID	State	Description	PointType	County	TypeA	TypeB	LatDMS	LonDMS	LatDec	LonDec
14021	1429436	UT	Lake Point	cape	Tooele	49	045	404205N	1121449W	40.70139	-112.24694
14022	1429437	UT	Lake Point	ppl	Tooele	49	045	404051N	1121544W	40.68083	-112.26222
14023	1429438	UT	Lake Point Cemetery	cemetery	Tooele	49	045	404017N	1121545W	40.67139	-112.2625
14024	1454454	UT	Lake Point Interchange	crossing	Tooele	49	045	404136N	1121547W	40.69333	-112.26306
14025	1437611	UT	Lake Point Junction	ppl	Tooele	49	045	404323N	1121331W	40.72306	-112.22528
14026	1435839	UT	Lake Powell Overlook	locale	Garfield	49	017	375302N	1102508W	37.88389	-110.41889
14028	1429440	UT	Lake Ridge	ridge	Rich	49	033	415703N	1110527W	41.95083	-111.09083
14030	1451607	UT	Lake Ridge	ppl	Salt Lake	49	035	404156N	1120418W	40.69889	-112.07167
14031	1442420	UT	Lake Ridge	ridge	Weber	49	057	411718N	1112948W	41.28833	-111.49667
14027	1429439	UT	Lake Ridge	ridge	Cache	49	005	413923N	1124112W	41.65639	-112.68667
14029	1442419	UT	Lake Ridge	ridge	Rich	49	033	411507N	1111322W	41.25194	-111.22278

UDL 文件基础

ActiveX 数据对象为我们提供了与数据库交互的框架。我们用连接串来指定

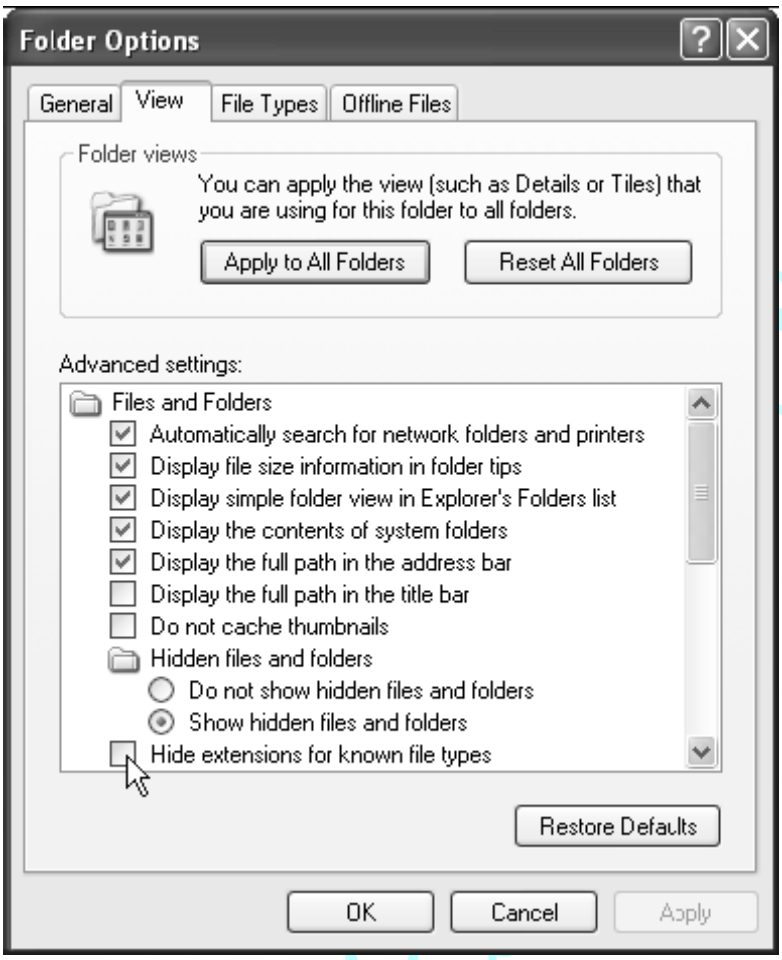
要操作的数据库位置以及连接数据库所用的驱动程序。连接串经常是硬编码的，意思是连接串出现在代码中，如果改变连接串的任何部分就必须改变代码。连接串也能从配置文件或 Windows 注册表中读取。指定驱动程序和数据库位置的另一种方法是使用 UDL 文件。

建立 UDL 文件的步骤非常简单。Windows 帮助文件中描述了这些步骤，在帮助文件的搜索框中键入“udl”能找到它们。



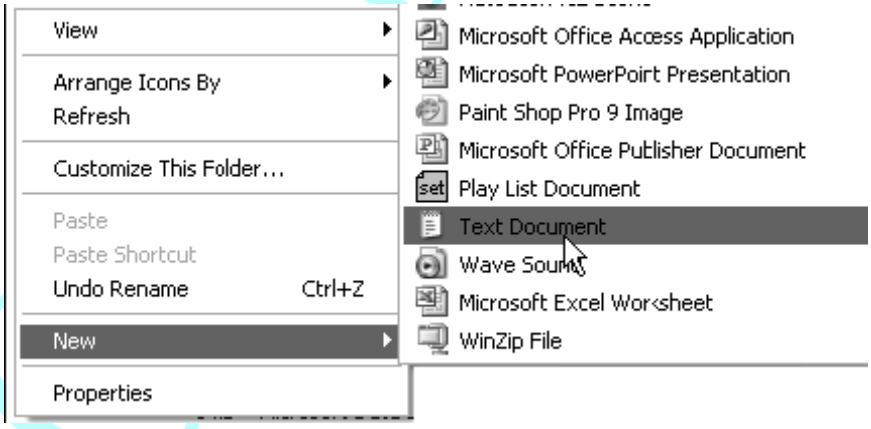
1、打开 Windows 资源管理器；

2、选择 **Tools > Folder Options(工具 > 文件夹选项)**；



3、在“查看”选项卡上，清除“隐藏已知文件类型的文件扩展名”；

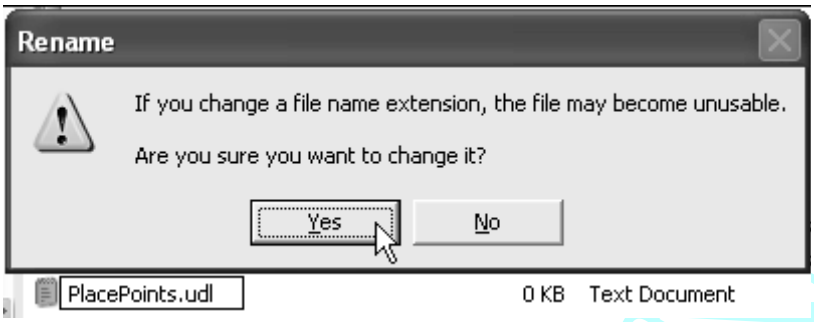
4、在要建立 UDL 文件的文件夹中右击并选择 **New > Text Document(新建 > 文本文档)**；



5、一个像 New Text Document.txt(新建 文本文档.txt)这样的文件显示出来并等待重命名；



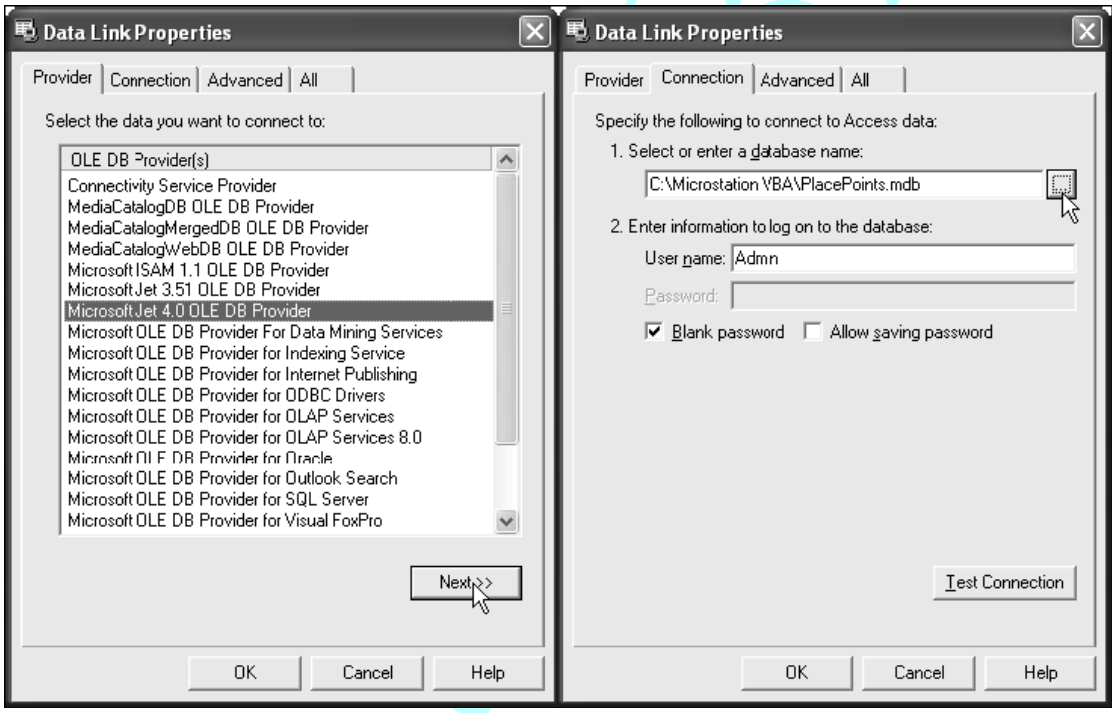
6、键入 UDL 文件名后跟扩展名“.udl”并按**<Enter>**；



7、当提问是否确认应该改变文件扩展名时，点击 Yes 按钮。

此时新的 UDL 文件被建立。现在需要打开 UDL 文件并设置要使用哪个驱动程序(根据要连接的数据库)以及数据库的位置；

8、在 Windows 资源管理器中双击 UDL 文件打开它；



9、在 Provider(提供程序)选项卡中选择“Microsoft Jet X.X OLE DB Provider”；

10、在 Connection(连接)选项卡中浏览到 PlacePoints.mdb 文件；

11、点击 OK 按钮。

祝贺您！一旦理解了 UDL 文件建立的基础，建立 UDL 文件是非常快的。

在 Provider(提供程序)选项卡中列出的“提供程序”会随计算机的不同而不同。对提供程序列表的查看能显示出我们可用的数据库。在上图所示的屏幕拷贝中，我们能看到有 Oracle、SQL Server、Visual Fox Pro、Jet 和 ODBC 等驱动程序。任何名字中含有“Jet”的提供程序都是指用于打开 Microsoft Access 数

据库的驱动程序。

连接、记录集以及更多内容

既然有了数据库(PlacePoints.mdb)和 UDL 文件(PlacePoints.udl)，就能在代码中使用连接和记录集了。

Connection 对象用于“连接”数据库，它是使用数据库中数据的第一步。让我们看第一个例子：

```
Sub TestConnectionA()
    Dim myDB As New ADODB.Connection
    myDB.Open "file name=C:\MicroStation VBA\PlacePoints.udl"
    myDB.Close
End Sub
```

这第一个例子使用刚建立的 UDL 文件打开一个数据库连接，然后立即关闭这个连接。注意是如何声明变量 **myDB** 的，它被声明为“New ADODB.Connection”。

除了把连接声明为“New ADODB.Connection”外，还可以把连接先声明为“ADODB.Connection”然后设置变量为“New ADODB.Connection”，如 TestConnectionB 所示。

```
Sub TestConnectionB()
    Dim myDB As ADODB.Connection
    Set myDB = New ADODB.Connection
    myDB.Open "file name=C:\MicroStation VBA\PlacePoints.udl"
    myDB.Close
End Sub
```

不是把连接声明为 New，而是在下一行设置它为 New ADODB.Connection。除了第二个例子中多用了一行代码外，这两种方法的区别可以被忽略。

打开并立即关闭连接对我们毫无用处。现在让我们在关闭连接前做点事情。

```
Sub TestConnectionC()
    Dim myDB As ADODB.Connection
    Set myDB = New ADODB.Connection
```

```
    myDB.Open "file name=C:\MicroStation VBA\PlacePoints.udl"
    myDB.Execute "Create Table PlaceNotes " & _
                 "(UniqueID Counter, PlaceID Long)"
    myDB.Close
End Sub
```

Execute 方法能让我们在 Connection 对象上执行 SQL 语句。上例建立了一个具有两个字段的表“PlaceNotes”。下面是另外一些使用 Execute 方法的例子。

TestConnectionD 给 PlaceNotes 表又增加了两个字段。

```
Sub TestConnectionD()
    Dim myDB As ADODB.Connection
    Set myDB = New ADODB.Connection
    myDB.Open "file name=C:\MicroStation VBA\PlacePoints.udl"
    myDB.Execute "Alter Table PlaceNotes " & _
                 "Add NoteBy Char(50), NoteDate Date"
    myDB.Close
End Sub
```

TestConnectionE 给 PlaceNotes 表添加了一条记录。

```
Sub TestConnectionE()
    Dim myDB As ADODB.Connection
    Set myDB = New ADODB.Connection
    myDB.Open "file name=C:\MicroStation VBA\PlacePoints.udl"
    myDB.Execute "Insert Into PlaceNotes " & _
                 "(PlaceID, NoteBy, NoteDate) VALUES " & _
                 "(1, 'JKW', '" & Now & "')"
    myDB.Close
End Sub
```

TestConnectionF 在 PlaceNotes 表中建立另一个字段。

```
Sub TestConnectionF()
    Dim myDB As ADODB.Connection
    Set myDB = New ADODB.Connection
```


China BDN

```
    myDB.Open "file name=C:\MicroStation VBA\PlacePoints.udl"
    myDB.Execute "Alter Table PlaceNotes Add TheNote Memo"
    myDB.Close
End Sub
```

TestConnectionG 更新 PlaceNotes 中的所有记录，对 NoteBy 字段为“JKW”的记录设置 TheNote 字段值为“Reviewed”。

```
Sub TestConnectionG()
    Dim myDB As ADODB.Connection
    Set myDB = New ADODB.Connection
    myDB.Open "file name=C:\MicroStation VBA\PlacePoints.udl"
    myDB.Execute "Update PlaceNotes Set TheNote = 'Reviewed' " & _
                 " Where NoteBy = 'JKW'"
    myDB.Close
End Sub
```

本章并不是关于 SQL(结构化查询语言)的全面指南。然而，介绍一些通过使用 Connection 对象的 Execute 方法中的标准 SQL 语句所能实现的功能也不错。

打开连接比较耗时。当然了，它不会花上数周、数日、数小时或数分钟，但可能会花上一或二秒钟。如果每次想使用数据库时就打开并关闭一个连接，程序性能将有很大影响。在一些情况下，我们可能只想打开连接一次并一直开着直到应用程序结束。我们能查看连接的 State 属性来确定其是打开还是关闭。

```
Sub TestConnectionH()
    Dim myDB As ADODB.Connection
    Set myDB = New ADODB.Connection
    myDB.Open "file name=C:\MicroStation VBA\PlacePoints.udl"
    Select Case myDB.State
    Case adStateClosed
        MsgBox "Connection is Closed."
    Case adStateConnecting
        MsgBox "Connection is Connecting."
    Case adStateExecuting
```

```
        MsgBox "Connection is Executing."
    Case adStateFetching
        MsgBox "Connection is Fetching."
    Case adStateOpen
        MsgBox "Connection is Open."
    End Select
    myDB.Close
End Sub
```

TestConnectionH 打开连接，然后用一个 Select Case 语句查看每种可能的状态。

在 TestConnectionJ 中，我们查看一个现实的如何使用 State 属性的例子。首先查看变量 **myDB**(应当在代码模块或用户窗体的通用声明区中声明过)是否关闭，如果是关闭的，就用 UDL 文件打开它。

```
Sub TestConnectionJ()
    If myDB.State = adStateClosed Then
        myDB.Open "file name=C:\MicroStation VBA\PlacePoints.udl"
    End If
    MsgBox "Use the Connection Object Here"
End Sub
```

这里应当注意到，我们没有在 TestConnectionJ 中声明变量 **myDB** 或设置它到 New ADODB.Connection。我们只是简单地检查它是否关闭，如果关闭就打开它。要使该过程能正确运行，变量 **myDB** 必须以这样的方式声明：它对该过程来说可用(在代码模块中声明为 Public，或位于该过程所在模块的通用声明区中)且被实例化(设置为 New ADODB.Connection)。例如，

```
    'General Declarations
    Dim myDB as New ADODB.Connection
```

既然能用 UDL 文件“连接”数据库，那么让我们看一看连接的 ConnectionString(连接串)属性。

```
Sub TestConnectionK()
    Dim myDB As ADODB.Connection
```

```
    Set myDB = New ADODB.Connection
    myDB.Open "file name=C:\MicroStation VBA\PlacePoints.udl"
    MsgBox Replace(myDB.ConnectionString, ";", vbCr)
    myDB.Close
End Sub
```

连接串相当长，以分号分隔。上例用回车来替换分号(;)，这样能更清晰地看连接串的内容。



使用 UDL 文件时，连接串反映了 UDL 文件的设置。虽然我们一直是依赖于 UDL 文件，但也能在打开连接时提供一个连接串，这样就能不用 UDL 文件而打开数据库并使用之。

```
Sub TestConnectionL()
    Dim myDB As ADODB.Connection
    Dim ConnectionStringVals(0 To 2) As String
    Set myDB = New ADODB.Connection
    ConnectionStringVals(0) = "Provider=Microsoft.Jet.OLEDB.4.0"
    ConnectionStringVals(1) = "User ID=Admin"
    ConnectionStringVals(2) = "Data Source=" & _
                              "C:\Microstation VBA\PlacePoints.mdb"
    myDB.Open Join(ConnectionStringVals, ";")
```

```
    MsgBox myDB.State
    myDB.Close
End Sub
```

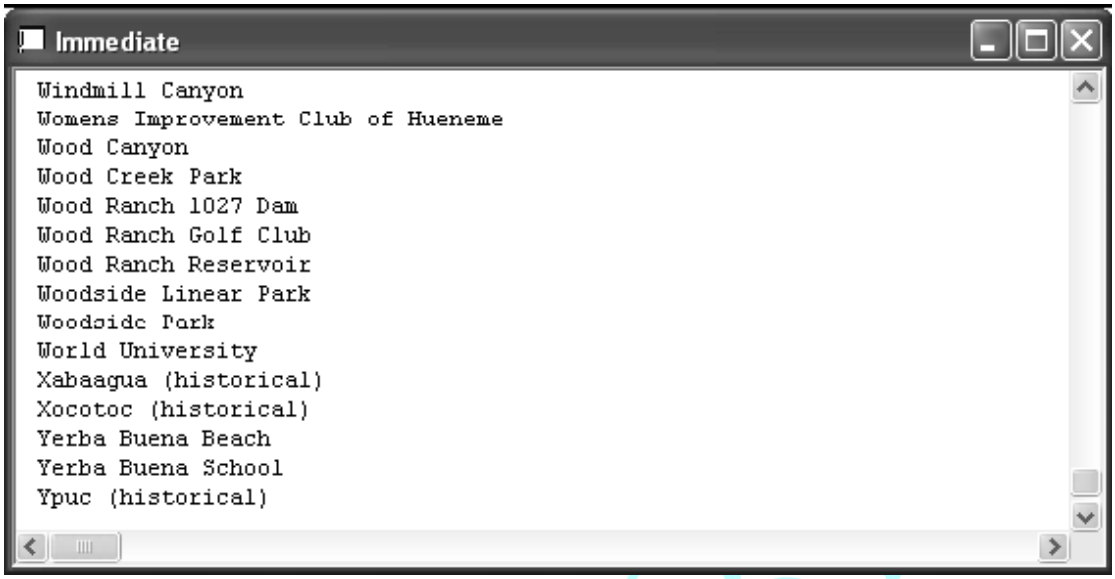
TestConnectionL 打开的数据库与使用 UDL 文件 PlacePoints.udl 所打开的是同一个数据库，但我们用连接串打开了连接，而没有用 UDL 文件。

记录集

Connection 对象用于“连接”数据库。Recordset(记录集)用于“连接”数据库中的记录。

```
Sub TestRecordsetA()
    Dim myDB As ADODB.Connection
    Dim myRS As New Recordset
    Set myDB = New ADODB.Connection
    myDB.Open "file name=C:\MicroStation VBA\PlacePoints.udl"
    myRS.Open "Select * from Points Where County = 'Ventura'", _
                    myDB, adOpenDynamic, adLockOptimistic
    While myRS.EOF = False
        Debug.Print myRS("Description")
        myRS.MoveNext
    Wend
    myRS.Close
    myDB.Close
End Sub
```

China BDN



在这第一个例子中，使用 Recordset 对象返回所有字段 County 为“Ventura”的记录的各个字段。虽然取得了所有字段(在 SQL Select 语句中使用星号(*))，但只将每个记录的描述(Description)显示到立即窗口中。

本章后续部分将详细讨论 SQL 语句。现在还是把注意力集中到 Recordset 对象上。

过程 TestRecordsetA 使用了一个 While … Wend 语句并查看 EOF(文件结束)属性。只要 EOF 属性为 False，我们就继续打印当前记录的 Description 字段然后移动到下一记录。MoveNext 就是移动记录集到下一记录的方法。

Open 方法用于指定要返回的数据、从哪个连接对象取数据、使用的游标类型以及检索数据时的记录锁定类型。

想要哪些数据？用一个 SQL 语句来指定想要的数据。记录集对象的 Open 方法中的连接对象指向当前连接。当谈及游标类型和锁定方法时，我们应该明白它们干什么用以及什么时候可能要用到它们。下面所示的描述是直接从微软网站上拿来的。

游标类型常量：

- adOpenDynamic = 2：使用动态游标。由其他用户所做的添加、改变和删除都可见，如果提供程序不支持书签功能的话，除书签以外，在记录集中允许所有类型的移动。
- adOpenForwardOnly = 0：缺省方式。使用只进游标。除了只能在记录集中向前滚动外与静态游标一样。当只需要在记录集中扫描一遍时，使用这种方式

能提高性能。

➢ adOpenKeyset = 1：使用键集游标。除不能看到其他用户添加的记录外很像动态游标，其他用户删除的记录也不能从您的记录集中访问。其他用户改变的数据仍然可见。

➢ adOpenStatic = 3：静态游标，它是记录集合的一个静态复制，可用于查找数据或生成报表。其他用户的添加、改变或删除都不可见。

锁定类型常量：

➢ adLockBatchOptimistic = 4：乐观批更新。为批更新模式所需。

➢ adLockOptimistic = 3：乐观锁定，逐条记录设置。提供程序使用乐观锁定时，仅当调用 Update 方法时才锁定记录。

➢ adLockPessimistic = 2：悲观锁定，逐条记录设置。提供程序为保证记录的成功编辑做一些必需的工作，通常是在记录被编辑后立即锁定记录。

➢ adLockReadOnly = 1：只读记录。不能改变数据。

游标类型对记录集的好几个属性都有影响。

```
Sub TestRecordsetB()
    Dim myDB As ADODB.Connection
    Dim myRS As New Recordset
    Set myDB = New ADODB.Connection
    myDB.Open "file name=C:\MicroStation VBA\PlacePoints.udl"
    myRS.Open "Select * from Points Where County = 'Ventura'", _
                myDB, adOpenForwardOnly, adLockReadOnly
    MsgBox myRS.RecordCount
    myRS.Close
    myDB.Close
End Sub
```

RecordCount(记录计数)是一个非常有用的属性，然而，与 adOpenDynamic 和 adOpenForwardOnly 一起使用时总是返回-1。使用 adOpenKeyset 和 adOpenStatic 时，RecordCount 属性给出记录集中检索到的记录个数。例如，运行 TestRecordsetB 时，不管检索到的记录有多少条，由于记录集 Open 语句中指

定的游标类型而总使 RecordCount 为-1。

```
Sub TestRecordsetC()
    Dim myDB As ADODB.Connection
    Dim myRS As New Recordset
    Set myDB = New ADODB.Connection
    myDB.Open "file name=C:\MicroStation VBA\PlacePoints.udl"
    myRS.Open "Select * from Points Where County = 'Ventura'", _
              myDB, adOpenStatic, adLockReadOnly
    MsgBox myRS.RecordCount
    myRS.Close
    myDB.Close
End Sub
```

TestRecordsetC 显示 RecordCount 为 2253。

```
Sub TestRecordsetD()
    Dim myDB As ADODB.Connection
    Dim myRS As New Recordset
    Set myDB = New ADODB.Connection
    myDB.Open "file name=C:\MicroStation VBA\PlacePoints.udl"
    myRS.Open "Select * from Points Where County = 'Ventura'", _
              myDB, adOpenDynamic, adLockOptimistic
    While myRS.EOF = False
        Debug.Print myRS("Description")
        myRS.MoveNext
    Wend
    myRS.MoveFirst
    While myRS.EOF = False
        Debug.Print myRS("CellName")
        myRS.MoveNext
    Wend
```

China BDN

```
    myRS.Close
    myDB.Close
End Sub
```

TestRecordsetD 使用了记录集对象的 MoveFirst 方法。该方法能让我们从记录集头开始再次从头至尾查看记录集，或者查找别的信息。

Find 方法

现在我们转向记录集对象的 Find(查找)方法。该方法把当前记录集游标移动到当前记录以下符合查找准则的第一条记录。它能让我们在用 Select 语句形成的记录集中搜索。由于记录集游标可能位于记录集中间的某个位置，所以使用记录集的 MoveFirst 方法是个好主意，这样总能从记录集头开始“查找”。为了能更好地查找，我们还对记录集按 CellName(单元名)字段排序。这能把记录集中 CellName 相同的所有记录放在一起。

```
Sub TestRecordsetE()
    Dim myDB As ADODB.Connection
    Dim myRS As New Recordset
    Set myDB = New ADODB.Connection
    myDB.Open "file name=C:\MicroStation VBA\PlacePoints.udl"
    myRS.Open "Select * from Points Where County = 'Ventura' " & _
            "Order by CellName", myDB, adOpenDynamic, adLockOptimistic
    myRS.Find "CellName = 'Lion Canyon'"
    While myRS("CellName") = "Lion Canyon"
        Debug.Print "Lion Canyon: " & myRS("Description")
        myRS.MoveNext
    Wend
    myRS.MoveFirst
    myRS.Find "CellName = 'Oxnard'"
    While myRS("CellName") = "Oxnard"
        Debug.Print "Oxnard: " & myRS("Description")
        myRS.MoveNext
```

```
    Wend
    myRS.MoveFirst
    myRS.Find "CellName = 'Fillmore'"
    While myRS("CellName") = "Fillmore"
        Debug.Print "Fillmore: " & myRS("Description")
        myRS.MoveNext
    Wend
    myRS.Close
    myDB.Close
End Sub
```

在这个例子中，我们只查找 County='Ventura'的记录。使用“Order by CellName”后记录集就按 CellName 字段排序。然后我们使用 Find 方法查找 CellName 为'Lion Canyon'的第一条记录。在查看完每个'Lion Canyon'单元后，我们继续查找第一个'Oxnard'单元，随后又是'Fillmore'。

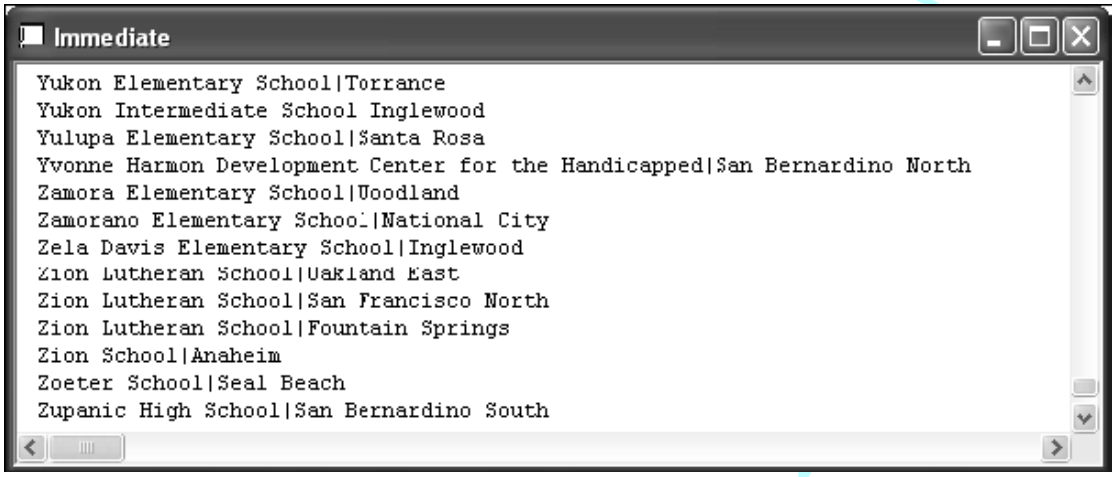
GetString 方法

下一个例子演示记录集对象的 GetString 方法。

```
Sub TestRecordsetF()
    Dim myDB As ADODB.Connection
    Dim myRS As New Recordset
    Set myDB = New ADODB.Connection
    myDB.Open "file name=C:\MicroStation VBA\PlacePoints.udl"
    myRS.Open "Select Description, CellName from Points " & _
              "Where State = 'CA' and PointType = 'school'", _
              myDB, adOpenDynamic, adLockOptimistic
    Debug.Print myRS.GetString(adClipString, -1, "|", vbCr)
    myRS.Close
    myDB.Close
End Sub
```

GetString 取得记录集中检索到的所有记录的所有字段并把它们放到一个大

的字符串中。我们指定了字段之间应当出现的分隔符，也指定了行(记录)之间的分隔符。本例用-1 作为检索记录个数的值，这就选择了检索所有记录。如果使用一个像 5 这样的数字，则 GetString 只返回最开头的五条记录。至于分隔符，我们用管道符(|)分隔每个字段，用回车符分隔每行。



AddNew 和 Update 方法

记录集的 AddNew 和 Update 方法用于建立新记录和更新字段值。下面是两个短的例子，随着本章的继续，我们将会看到更多对 AddNew 和 Update 的使用。

```
Sub TestRecordsetH()
    Dim myDB As ADODB.Connection
    Dim myRS As New Recordset
    Set myDB = New ADODB.Connection
    myDB.Open "file name=C:\MicroStation VBA\PlacePoints.udl"
    myRS.Open "PlaceNotes", myDB, adOpenDynamic, adLockOptimistic
    myRS.AddNew
    myRS("PlaceID") = 4
    myRS("NoteBy") = "JKW"
    myRS("NoteDate") = Now
    myRS("TheNote") = "New Note"
    myRS.Update
    myRS.Close
    myDB.Close
```

```
End Sub

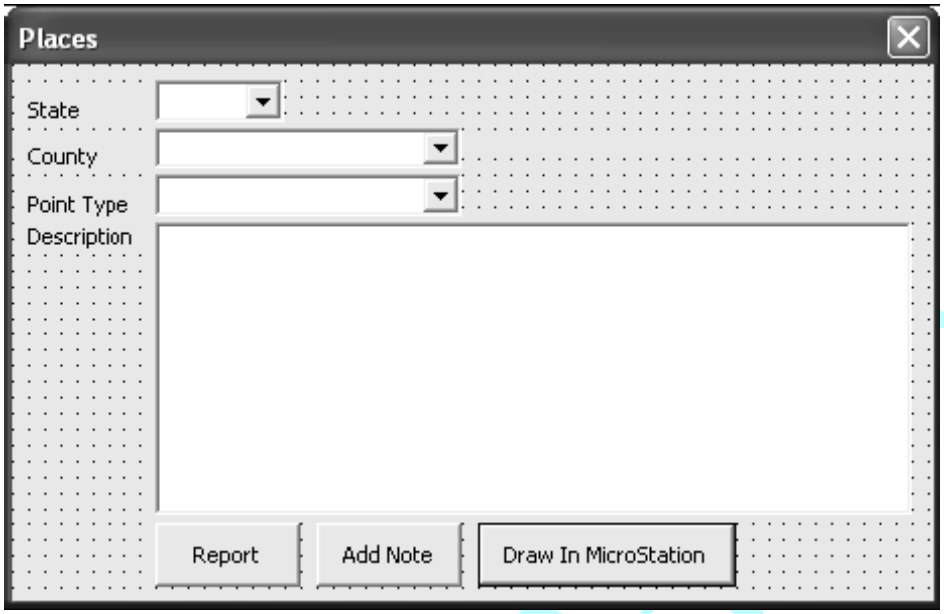

Sub TestRecordsetJ()
    Dim myDB As ADODB.Connection
    Dim myRS As New Recordset
    Set myDB = New ADODB.Connection
    myDB.Open "file name=C:\MicroStation VBA\PlacePoints.udl"
    myRS.Open "Select * from PlaceNotes Where PlaceID = 1", _
                    myDB, adOpenDynamic, adLockOptimistic
    While myRS.EOF = False
        myRS("PlaceID") = 14
        myRS.Update
        myRS.MoveNext
    Wend
    myRS.Close
    myDB.Close
End Sub
```

TestRecordsetH 在表中增加了一条新记录；TestRecordsetJ 查询数据库并改变 SQL 语句检索到的每条记录中的 PlaceID 值。在这两个例子中，都使用了 Update 方法为数据库提供字段值。

SQL 要素

既然已经讨论了用 Connection 对象连接数据库以及用 Recordset 对象连接数据库中的数据，那么让我们开始讨论 Connection 和 Recordset 对象可用的 SQL 语句。通过建立几个用户窗体来讨论 SQL 语句。

China BDN



建立的第一个窗体如上所示。窗体装载时需要查询数据库中所有不重复的州名值，这些值将添加到 **State(州)**复合框中。

Select 语句

Select 语句是我们将要使用的许多 SQL 语句中最基本的语句。它能让我们指定要检索的字段、字段所属的表、记录如何排序、字段如何分组等。

用如下语句取得 Points 表中不重复的州名：

```
Select Distinct State from Points
```

执行该语句后，记录集将被每个 State 字段值不重复的记录填充。在我们的例子中，还将每个 State 字段值放入 **cmbState** 复合框中。

```
Private Sub UserForm_Initialize()
    Dim myDB As New ADODB.Connection
    Dim myRS As New ADODB.Recordset
    myDB.Open "file name=C:\MicroStation VBA\PlacePoints.udl"
    myRS.Open "Select Distinct State from Points", myDB
    While myRS.EOF = False
        cmbState.AddItem myRS("State")
        myRS.MoveNext
    Wend
    myRS.Close
```

```
    myDB.Close
End Sub
```

窗体显示时，每个唯一的州名都被添加到了复合框中。本例中的数据取自一个数据库，该数据库从 USGS 站点 http://geonames.usgs.gov/stategaz/index.html上的信息建立。虽然美国 50 个州的数据都可用，但我们只用了两个。一个是加利福尼亚州(CA)，相当大的一个数据集；另一个是犹他州(UT)，非常小的一个数据集。所以在该例中只显示了两个州：CA和UT。

当用户从 **cmbState** 复合框选择一个州时，要将选中州的所有郡县(County)赋给 cmbCounty 复合框。但在向 **cmbCounty** 复合框添加内容前使用了 Clear 方法，对 cmbPointType 和 lstDescription 控件也执行了 Clear 方法。如果不清除复合框的话，每当选择一个州时郡县名将继续添加到 **cmbCounty** 中，这样 **cmbCounty** 复合框就不再只是显示选中州的郡县名了。

Where

“Where”语句提供查询准则，这些准则指定了要检索的记录。本例只想要 State 字段等于 **cmbState** 复合框中所选州的那些记录。由于 State 字段是一个字符串(文本)，所以在值的前后使用了单引号(‘)。

Order By

Order By 语句能让我们指定如何排序记录集中的记录。可以指定多个字段，“ASC”表示升序排列，“DESC”表示降序排列。

```
Private Sub cmbState_Click()
    Dim myDB As New ADODB.Connection
    Dim myRS As New ADODB.Recordset
    cmbCounty.Clear
    cmbPointType.Clear
    lstDescription.Clear
    myDB.Open "file name=C:\MicroStation VBA\PlacePoints.udl"
    myRS.Open "Select Distinct County from Points Where State = " & _
```

```
                """ & cmbState.Text & "' Order by County ASC", myDB
    While myRS.EOF = False
        cmbCounty.AddItem myRS("County")
        myRS.MoveNext
    Wend
    myRS.Close
    myDB.Close
End Sub
```

现在好了，当用户点击 State 后，**cmbCounty** 复合框中显示出数据库中对应的郡县名。当用户点击一个郡县时又将发生什么呢？让我们对 **cmbPointType** 复合框赋值，该框中仅显示那些选中州和选中郡县的记录的点类型。

```
Private Sub cmbCounty_Click()
    Dim myDB As New ADODB.Connection
    Dim myRS As New ADODB.Recordset
    cmbPointType.Clear
    lstDescription.Clear
    myDB.Open "file name=C:\MicroStation VBA\PlacePoints.udl"
    myRS.Open "Select Distinct PointType from Points Where State = " & _
                """ & cmbState.Text & "' and " & _
                "County = '" & cmbCounty.Text & "' " & _
                "Order by PointType ASC", myDB
    While myRS.EOF = False
        cmbPointType.AddItem myRS("PointType")
        myRS.MoveNext
    Wend
    myRS.Close
    myDB.Close
End Sub
```

当用户点击 PointType 复合框时，将能看到所有满足复合框中设置的选择准

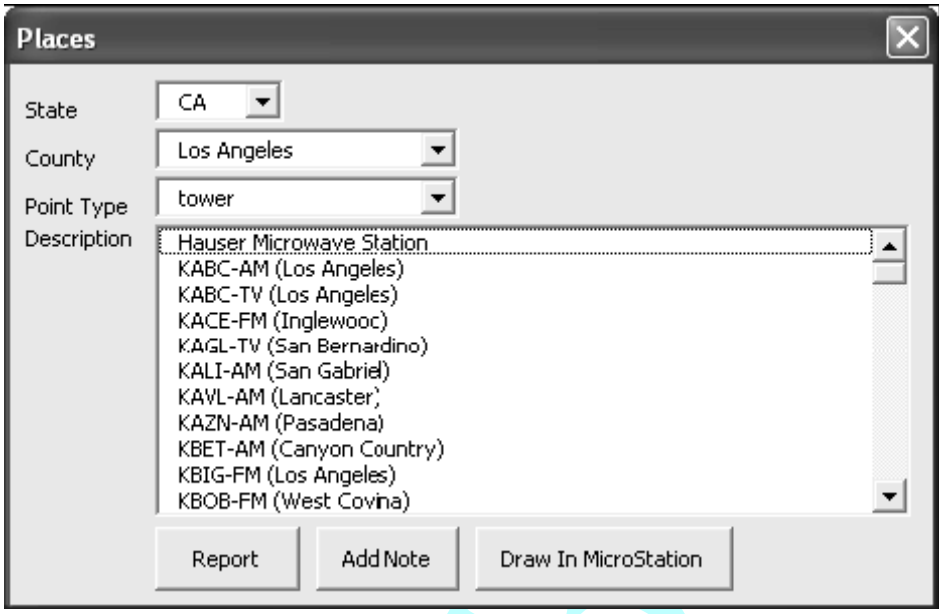
则的描述信息。我们在列表框的第二列放置了 UniqueID 字段(但隐藏了这个列，所以最终用户看不到)。

```
Private Sub cmbPointType_Click()
    Dim myDB As New ADODB.Connection
    Dim myRS As New ADODB.Recordset
    lstDescription.Clear
    myDB.Open "file name=C:\MicroStation VBA\PlacePoints.udl"
    myRS.Open "Select Description, UniqueID from Points Where State = " _
              & "'" & cmbState.Text & "' and " & _
              "County = '" & cmbCounty.Text & "' " & _
              "and PointType = '" & cmbPointType.Text & "' " & _
              "Order by Description ASC", myDB
    While myRS.EOF = False
        lstDescription.AddItem myRS("Description")
        lstDescription.List(lstDescription.ListCount - 1, 1) = _
                                        myRS("UniqueID")
        myRS.MoveNext
    Wend
    myRS.Close
    myDB.Close
End Sub
```

以上所有代码构成了让用户从数据库中选择地区的程序框架。

ChinaBDN

现在，窗体看起来像这样：

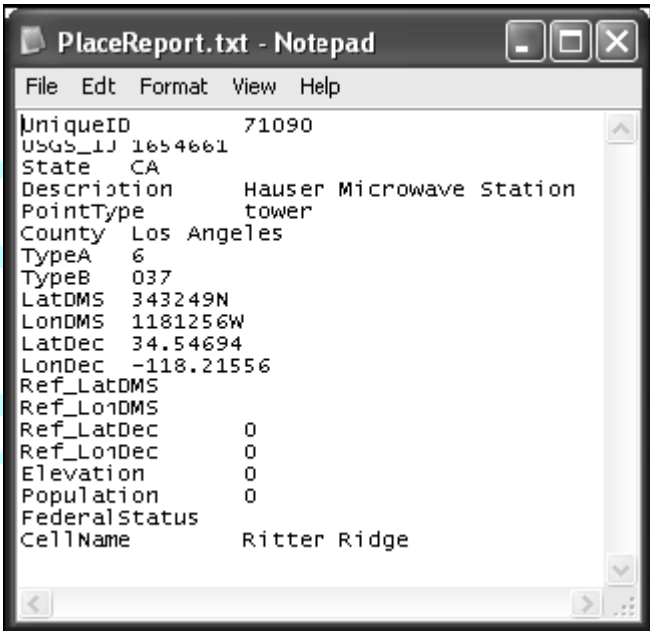


从窗体中选择州“CA”、郡“Los Angeles”以及“tower”点类型将显示所有满足该准则的点。

现在来看窗体上的按钮，从 Report(报表)按钮开始。当点击该按钮且选择了一个“地区”时，将生成一个从数据库提取的含所有选择地区字段值的 ASCII 文本文件。

先看结果文件，再看代码。

我们已经看过如何用字段名来取得特定字段的值，我们可以用这种方法来生成报表。然而，我们的目标是导出所有字段的名字及其值，用 Recordset 对象的 Fileds 集合能实现这一点。




```
Private Sub cmdReport_Click()
    If lstDescription.Text = "" Then Exit Sub
    Dim myDB As New ADODB.Connection
    Dim myRS As New ADODB.Recordset
    Dim myField As Field
```

```
    Dim FFile As Long
    myDB.Open "file name=C:\MicroStation VBA\PlacePoints.udl"
    myRS.Open "Select * from Points Where UniqueID = " & _
    lstDescription.List(lstDescription.ListIndex, 1), myDB
    If myRS.EOF = False Then
        FFile = FreeFile
        Open "C:\PlaceReport.txt" For Output As #FFile
        For Each myField In myRS.Fields
            Print #FFile, myField.Name & vbTab & myField.Value
        Next
        Close #FFile
    End If
    myRS.Close
    myDB.Close
End Sub
```

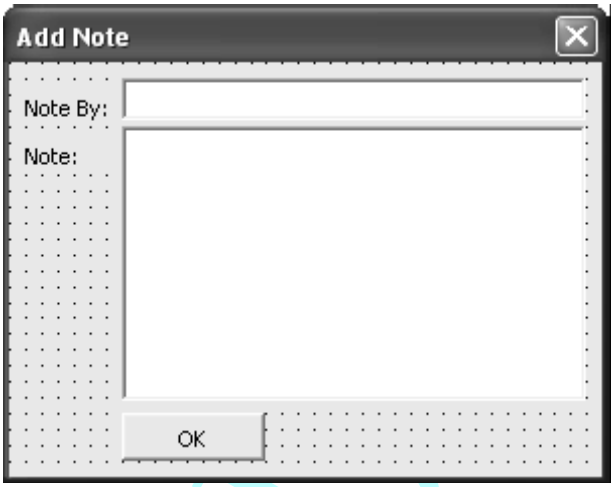
当把满足所选准则的每个“地区”的描述添加到列表框时，也在列表框的第二列(被隐藏)添加了 UniqueID 字段，我们的查询中使用了该值。如果找到了一条匹配的记录(应该总能找到一条匹配的记录)，就打开一个 ASCII 文本文件作为输出，把每个字段名和值写入文件中并关闭该文件。

还需要注意的是 Select 语句中的查询“Select *”。当使用星号字符时，表明要检索表中的所有字段。

Report 按钮好像工作得很好了，现在把注意力转到 Add Note(添加注释)按钮上。点击 Add Note 按钮后需要显示一个新的用户窗体以允许用户输入注释信息。Add Note 按钮通过设置 Add Note 窗体的 Tag 属性来告知 Add Note 窗体选择了那条记录，然后显示该窗体。

```
    Private Sub cmdAddNote_Click()
        If lstDescription.Text = "" Then Exit Sub
        frmAddNote.Tag = lstDescription.List(lstDescription.ListIndex, 1)
        frmAddNote.Show
    End Sub
```

这里是窗体 **frmAddNote**



当用户点击 OK 按钮时，执行下列代码：

```
Private Sub btnOK_Click()
    If frmAddNote.Tag = "" Then
        MsgBox "AddNote not executed correctly."
        Unload Me
    End If
    If txtNoteBy.Text = "" Then
        MsgBox "Please enter Note By."
        Exit Sub
    End If
    If txtNote.Text = "" Then
        MsgBox "Please enter Note."
        Exit Sub
    End If
    Dim myDB As New ADODB.Connection
    Dim myRS As New ADODB.Recordset
    Dim myField As Field
    Dim FFile As Long
    myDB.Open "file name=C:\MicroStation VBA\PlacePoints.udl"
    myRS.Open "Select * from PlaceNotes Where UniqueID = 0", _
                        myDB, adOpenDynamic, adLockOptimistic
    myRS.AddNew
```

```
    myRS("NoteBy") = txtNoteBy.Text
    myRS("TheNote") = txtNote.Text
    myRS("PlaceID") = frmAddNote.Tag
    myRS("NoteDate") = Now
    myRS.Update
    MsgBox "Note added."
    Unload Me
End Sub
```

有几种选择可向表中添加新记录。可以用 Connection.Execute 方法执行“Insert Into”SQL 语句。但在这个例子，我们打开 PlaceNotes 表并使用了 Recordset 对象的 AddNew 方法。这样做比使用“Insert Into”语句稍灵活一些。打开表时我们使用了一个 Select 语句并查找 UniqueID 为 0 的记录，为什么要这样做呢？如果想打开整个表，可以使用如下样子：

```
    myRS.Open "PlaceNotes", myDB, adOpenDynamic, adLockOptimistic
```

但这样做性能很差，因为我们打开了表中的每条记录。故意打开一个不含记录的记录集时，记录集被立即打开，因为它不需要检索任何数据。

最后一个要讨论的按钮是 Draw In MicroStation(在 MicroStation 中绘制)按钮。我们所使用的数据库具有经度和纬度值，这些值给出了数据库中“地区”的 Y 和 X 坐标。我们将使用这些值在选择的“地区”位置放置一个圆和文本元素。

首先看一下按钮背后的代码，然后看结果。

```
Private Sub btnDraw_Click()
    If lstDescription.ListCount = 0 Then Exit Sub
    Dim myDB As New ADODB.Connection
    Dim myRS As New ADODB.Recordset
    Dim myPoint As Point3d
    Dim myCircle As ArcElement
    Dim myText As TextElement
    Dim RotMatrix As Matrix3d
    Dim I As Long
    myDB.Open "file name=C:\MicroStation VBA\PlacePoints.udl"
```

```
    For I = 1 To lstDescription.ListCount
        If lstDescription.Selected(I - 1) = True Then
            myRS.Open "Select * from Points Where UniqueID = " & _
                    lstDescription.List(I - 1, 1), myDB
            If myRS.EOF = False Then
                myPoint.X = myRS("LonDec")
                myPoint.Y = myRS("LatDec")
                Set myCircle = CreateArcElement2(Nothing, myPoint, _
                                0.0025, 0.0025, RotMatrix, 0, 360)
                ActiveModelReference.AddElement myCircle
                Set myText = CreateTextElement1(Nothing, _
                        myRS("Description"), myPoint, RotMatrix)
                myText.TextStyle.Height = 0.01
                myText.TextStyle.Width = 0.01
                ActiveModelReference.AddElement myText
            End If
            myRS.Close
        End If
    Next I
    myDB.Close
End Sub
```

我们使用 LonDec 和 LatDec 字段作为每个圆的中心点和文本原点的 X 和 Y 分量。把 Description 字段值作为 MicroStation 中的文本元素。

在前一个按钮中，我们基于列表框的 ListIndex(列表索引)属性工作。由于这个按钮要在 MicroStation 中绘制，所以想让软件通过一个按钮点击放置多个点。这就是为什么要考察列表框中每项的 Selected 属性的原因。如果某项被选择，就用隐藏在列表框第 2 列的 UniqueID 来查询数据库。

让我们放慢脚步，我们正穿行于大量的代码之中。在如上所示的代码变得“晴朗”(或者还有点云)之后，我们还将增强程序的功能。把圆和文本放置到与点类

型同名的层上。

```
Private Sub btnDraw_Click()
    If lstDescription.ListCount = 0 Then Exit Sub
    Dim myDB As New ADODB.Connection
    Dim myRS As New ADODB.Recordset
    Dim myPoint As Point3d
    Dim myCircle As ArcElement
    Dim myText As TextElement
    Dim RotMatrix As Matrix3d
    Dim I As Long
    myDB.Open "file name=C:\MicroStation VBA\PlacePoints.udl"
    For I = 1 To lstDescription.ListCount
        If lstDescription.Selected(I - 1) = True Then
            myRS.Open "Select * from Points Where UniqueID = " & _
                    lstDescription.List(I - 1, 1), myDB
            If myRS.EOF = False Then
                myPoint.X = myRS("LonDec")
                myPoint.Y = myRS("LatDec")
                Set myCircle = CreateArcElement2(Nothing, myPoint, _
                        0.0025, 0.0025, RotMatrix, 0, 360)
                myCircle.Level = CheckLevel(cmbPointType.Text)
                ActiveModelReference.AddElement myCircle
                Set myText = CreateTextElement1(Nothing, _
                        myRS("Description"), myPoint, RotMatrix)
                myText.TextStyle.Height = 0.01
                myText.TextStyle.Width = 0.01
                myText.Level = CheckLevel(cmbPointType.Text)
                ActiveModelReference.AddElement myText
            End If
            myRS.Close
```

```
            End If
        Next I
        myDB.Close
End Sub
Function CheckLevel(LevelName As String) As Level
        On Error Resume Next
        Set CheckLevel = ActiveDesignFile.AddNewLevel(LevelName)
        If Err.Number <> 0 Then
            Set CheckLevel = ActiveDesignFile.Levels(LevelName)
        End If
End Function
```

我们只在 **btnDraw** 的单击事件中增加了两行代码并增加了函数 CheckLevel。现在，所有选择的地区都被添加到当前模型参考的指定层上，层名与选择的“点类型”相匹配。

正如我们的程序现在所表达的，我们具有了一些非常好的功能。我们能根据列表框中选择的项得到一个报表；能给列表框中选择的项添加注释；能在 MicroStation 的当前模型参考中绘制列表框中选择的项。

扩展 ActiveX 数据对象

正如其名字所隐含的，ActiveX 数据对象并不仅仅是数据库，它是数据的全部。数据以各种形式出现，数据库保存不断变化的数据，但数据库中的数据好像不会每 24 小时就发生完全不同的变化。

使 Internet 如此强大的一个因素是，它是那么的动态，它每天每秒都在变化。虽然 Internet 可以被看作一个庞大的数据库，但它代表的是整个数据而不是数据库。

当查看 UDL 文件的提供程序选项卡时，会看到一个“Microsoft OLE DB Simple Provider”引用。这个引用能做什么？我们能用它来访问 Internet 上公开的数据吗？是的，它能用于访问 Internet 上的某些数据。

让我们考察一下“Microsoft OLE DB Simple Provider”。这一提供程序用于建立内存中的数据库，还能用于操作 XML 文件。在前面一章已经处理过 XML

文件，现在让我们来看一看操作 XML 文件的另一种方法。

许多网站使用 RSS 技术。当一个按钮显示为“RSS”或“XML”时，该按钮链接到了站点的 RSS 传播(feed)。该传播通常包含链接到全文的标题。除了主新闻网络和其它高流量网站使用该技术外，较小的网站和公司内部网也能利用这项技术(如果现在还没有使用，在不久的将来也会使用)。

下一个例子将建立一个新的叫做 RSS.udl 的 UDL 文件。提供程序是“Microsoft OLE DB Simple Provider”。该 UDL 文件的数据源是“MSXML2.DSOControl.2.6”。我们将使用该 UDL 文件并在 Recordst.Open 方法中指定 RSS 传播的 URL。

RSS 文件遵从特定的文档结构。三个基本的层次是：

频道(Channel)

项(Item)

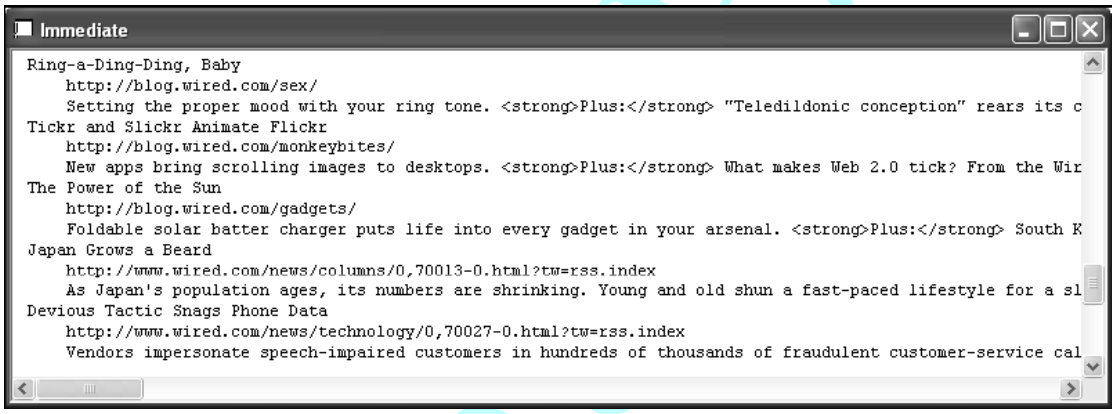
子项(Item Child)

我们将为频道和项使用记录集，并将每项的标题、链接和描述打印到立即窗口。这是我们的起点。

```
Sub ReadRSSA()
    Dim MyDB As New ADODB.Connection
    Dim MyRS As New ADODB.Recordset
    Dim ChannelRS As New ADODB.Recordset
    Dim ItemRS As New ADODB.Recordset
    MyDB.Open "File name=c:\MicroStation VBA\rss.udl"
    MyRS.Open "http://www.wired.com/rss/index.xml", MyDB
    While MyRS.EOF = False
        Set ChannelRS = MyRS("channel").Value
        While ChannelRS.EOF = False
            Set ItemRS = ChannelRS("item").Value
            While ItemRS.EOF = False
                Debug.Print ItemRS("title")
                Debug.Print vbTab & ItemRS("link")
                Debug.Print vbTab & ItemRS("description")
```

```
            ItemRS.MoveNext
        Wend
        ChannelRS.MoveNext
    Wend
    MyRS.MoveNext
Wend
End Sub
```

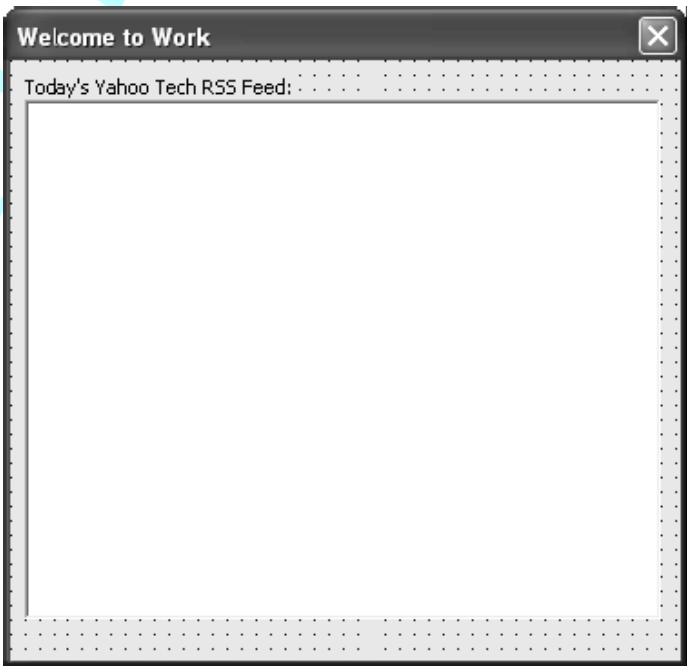
ReadRSSA 执行后，wired.com 网站的 RSS 显示在立即窗口中。由于 RSS 文件通常更新得很频繁，所以立即窗口中显示的结果会每天都不同，甚至会每小时都不同。

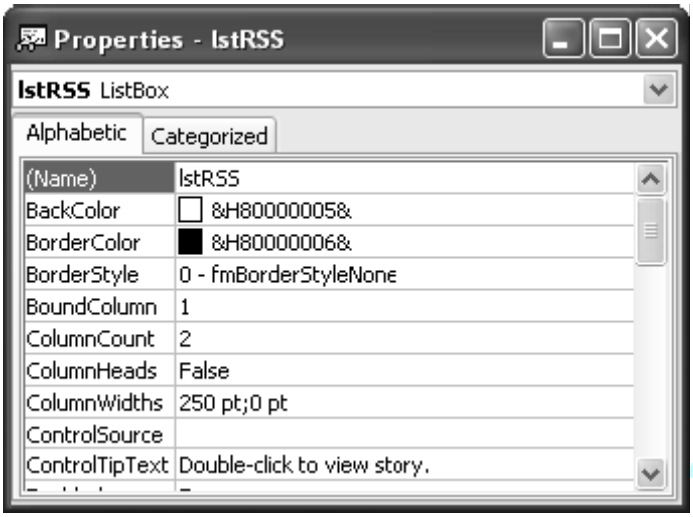


该项技术非常棒，检索信息只用了 22 行代码。既然我们知道了这项技术并且知道如何存取数据，那么在 VBA 工程中如何使用这项技术呢？

启动一个新的 VBA 工程，将其命名为 Chapter 35c.mvba 。添加引用“Microsoft ActiveX Data Objects 2.X Library”，然后插入一个用户窗体。

右图就是这个窗体。





窗体中唯一的控件是一个列表框，名字是 **lstRSS**。该列表框有两列，宽度分别为 250 点和 0 点。

第一列中将放置每个 RSS 传播项的标题，第二列(通过给定 0 点宽度将其隐藏)将放置与第一列的标题相关联的 URL。当用户双击列表中的一项时，打开一个新的网页浏览器并显示相应的内容。



双击选择项后，我们用 ShellExecute API 命令打开系统中缺省的浏览器并显示所选的内容。下面是代码：

```
Private Declare Function ShellExecute Lib "shell32.dll" _
                         Alias "ShellExecuteA" _
                         (ByVal hwnd As Long, _
                         ByVal lpOperation As String, _
                         ByVal lpFile As String, _
                         ByVal lpParameters As String, _
```

```
                                ByVal lpDirectory As String, _
                                ByVal nShowCmd As Long) As Long
Private Sub lstRSS_DblClick(ByVal Cancel As MSForms.ReturnBoolean)
    ShellExecute 0, "OPEN", lstRSS.List(lstRSS.ListIndex, 1), "", "", 0
End Sub
Private Sub UserForm_Initialize()
    Dim MyDB As New ADODB.Connection
    Dim MyRS As New ADODB.Recordset
    Dim ChannelRS As New ADODB.Recordset
    Dim ItemRS As New ADODB.Recordset
    MyDB.Open "File name=c:\MicroStation VBA\rss.udl"
    MyRS.Open "http://rss.news.yahoo.com/rss/tech", MyDB
    While MyRS.EOF = False
        Set ChannelRS = MyRS("channel").Value
        While ChannelRS.EOF = False
            Set ItemRS = ChannelRS("item").Value
            While ItemRS.EOF = False
                lstRSS.AddItem Replace(ItemRS("title"), vbLf, "")
                lstRSS.List(lstRSS.ListCount - 1, 1) = ItemRS("link")
                ItemRS.MoveNext
            Wend
            ChannelRS.MoveNext
        Wend
        MyRS.MoveNext
    Wend
End Sub
```

RSS 传播将会使用得越来越频繁。正如这段代码所做的，它将变得更重要和更有用。例如，一个公司可以在其内部网上建立自己的 RSS 传播系统来显示每个人的分配的任务。我今天做什么项目？打开我的 RSS 阅读器，它就会告诉

我。

分析数据库架构

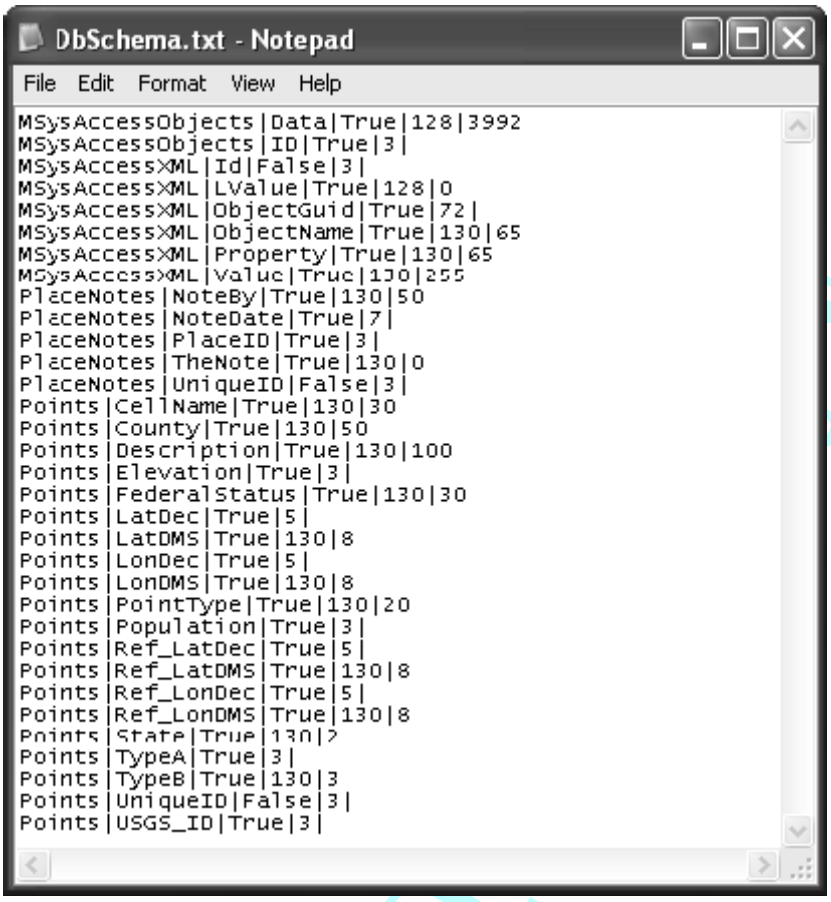
我们有时会面临这样一个任务：使用不能在本机应用程序中打开的数据。例如，我们有一个 Visual FoxPro 的.dbf 文件但却没有 Visual FoxPro 软件。同样的情况会发生在许多数据库上，如 Access、DB2 等。

ActiveX 数据对象为我们提供了查看所给数据库架构的能力。换句话说，架构就是数据库的结构。

```
Sub TestSchemaA()
    Dim myDB As New ADODB.Connection
    Dim myRS As New ADODB.Recordset
    myDB.Open "file name=C:\MicroStation VBA\PlacePoints.udl"
    Set myRS = myDB.OpenSchema(adSchemaColumns)
    While myRS.EOF = False
        Debug.Print myRS("TABLE_NAME") & "|" & _
                    myRS("COLUMN_NAME") & "|" & _
                    myRS("IS_NULLABLE") & "|" & _
                    myRS("DATA_TYPE") & "|" & _
                    myRS("CHARACTER_MAXIMUM_LENGTH")
        myRS.MoveNext
    Wend
End Sub
```

我们仍然使用了 Connection 对象和 Recordset 对象，仍然打开了数据库。但打开数据库后使用了 Connection 对象的“OpenSchema”方法。OpenSchema 方法带上 adSchemaColumns 参数后，返回的每条记录都有几个不同的字段。这些字段是 TABLE_NAME、COLUMN_NAME、IS_NULLABLE、DATA_TYPE 和 CHARACTER_MAXIMUM_LENGTH。通过这些字段能了解指定数据库(即在 UDL 文件中指定的)中所含每个字段和表的结构。

这每一个字段的值都打印到了立即窗口中。立即窗口不能保存无限多的行，所以有时把值写入一个文本文件会有更好的作用。



DATA_TYPE 字段返回一个对应于 ADO 常数的数字值。

- adArray = 8192 (一个标记值，总是与其它数据类型常数一起使用，表明是某种数据类型的数组)
- adBigInt = 20 (8 字节的有符号整数(DBTYPE_I8))
- adBinary = 128 (二进制值(DBTYPE_BYTES))
- adBoolean = 11 (布尔型值(BDTYPE_BOOL))
- adBSTR = 8 (以 null 结尾的字符串(Unicode)(DBTYPE_BSTR))
- adChapter = 136 (4 字节标题值，用于识别子行集中的行(DBTYPE_HCHAPTER))
- adChar = 129 (字符串值(DBTYPE_STR))
- adCurrency = 6 (货币值(DBTYPE_CY)。货币值是一个定点数，小数点右侧有四位数字。它是放大了 10000 倍的 8 字节有符号整数。)
- adDate = 7 (日期值(DBTYPE_DATE)。日期以双精度数存储，整数部分是自 1899 年 12 月 30 日至今的天数，小数部分是一天中的分数。)
- adDBDate = 133 (日期值(yyyymmdd)(DBTYPE_DBDATE))

- adDBTime = 134 (时间值(hhmmss)(DBTYPE_DBTIME))
- adDBTimeStamp = 135 (日期/时间戳(yyyymmddhhss 加上一个十亿分之一为单位的小数)(DBTYPE_DBTIMESTAMP))
- adDecimal = 14 (具有固定精度和范围的精确数值(DBTYPE_DECIAML))
- adDouble = 5 (双精度浮点值(DBTYPE_R8))
- adEmpty = 0 (没有值(DBTYPE_EMPTY))
- adError = 10 (32 位错误代码(DBTYPE_ERROR))
- adFileTime = 64 (自 1601 年 1 月 1 日至今的以千万分之一秒为间隔的 64 位值(DBTYPE_FILETIME))
- adGUID = 72 (全局唯一标识符(DBTYPE_GUID))
- adInteger = 3 (4 字节有符号整数(DBTYPE_I4))
- adLongVarBinary = 205 (长二进制值)
- adLongVarChar = 201 (长字符串值)
- adLongVarWChar = 203 (以 null 结尾的长 Unicode 字符串值)
- adNumeric = 131 (具有固定精度和范围的精确数值(DBTYPE_NUMERIC))
- adPropVariant = 138 (自动化 PROPVARIANT(DBTYPE_PROP_VARIANT))
- adSingle = 4 (单精度浮点值(DBTYPE_R4))
- adSmallInt = 2 (双字节有符号整数(DBTYPE_I2))
- adTinyInt = 16 (单字节有符号整数(DBTYPE_I1))
- adUnsignedBigInt = 21 (8 字节无符号整数(DBTYPE_UI8))
- adUnsignedInt = 19 (4 字节无符号整数(DBTYPE_UI4))
- adUnsignedSmallInt = 18 (双字节无符号整数(DBTYPE_UI2))
- adUnsignedTinyInt = 17 (单字节无符号整数(DBTYPE_UI1))
- adUserDefined = 132 (用户自定义变量(DBTYPE_UDT))
- adVarBinary = 204 (二进制值)
- adVarChar = 200 (字符串值)
- adVarNumeric = 139 (数字值)
- adVarWChar = 202 (以 null 结尾的 Unicode 字符串)
- adWChar = 130 (以 null 结尾的 Unicode 字符串(DBTYPE_WSTR))

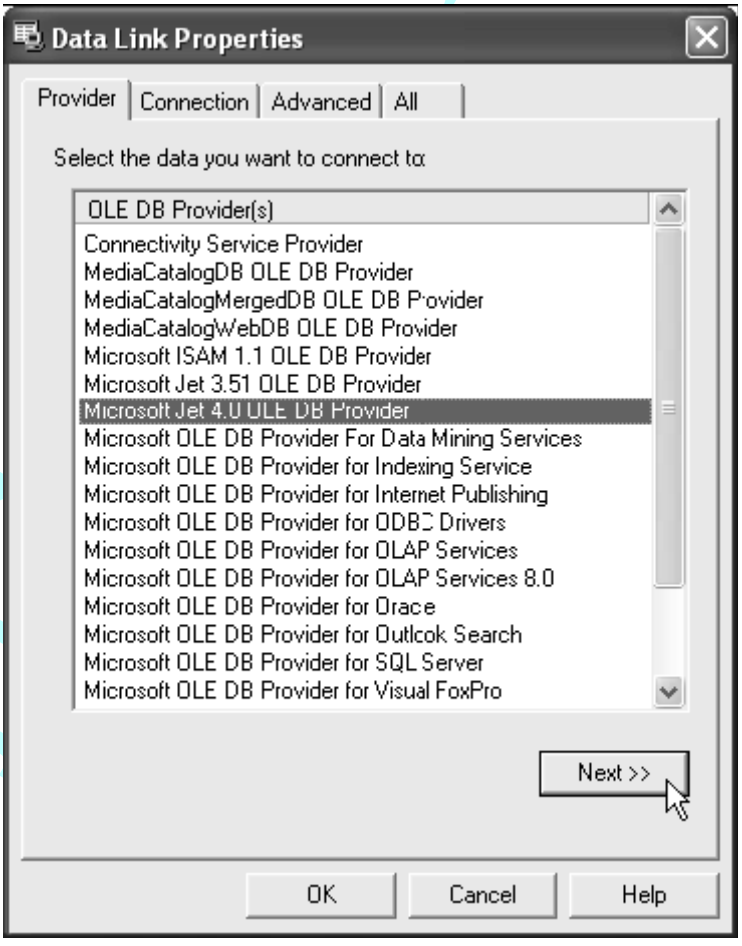
PlacePoints.mdb 数据库具有表“Points”和表“PlaceNotes”。在查看数据库架构时还看到了表“MSysAccessObjects”和“MSysAccessXML”。在数据库中建立系统表来实现诸如为索引字段建立索引之类的功能，这对于数据库来说并非罕见。大多数情况下，哪些表是“系统”表、哪些表是用户表都是很清楚的。

Excel 文件作为数据库

Excel 文件被分为行和列，对吗？对，但也很有理由用 ActiveX 数据对象来打开它们。让我们先指定一个要操作的 Excel 文件(.xls)，towerdat.xls 文件随 MicroStation 一起安装，在计算机中查找可知其安装在“Documents and Settings”下一个很深的路径中。将其复制并粘贴至一个较简单的路径下能简化使用。

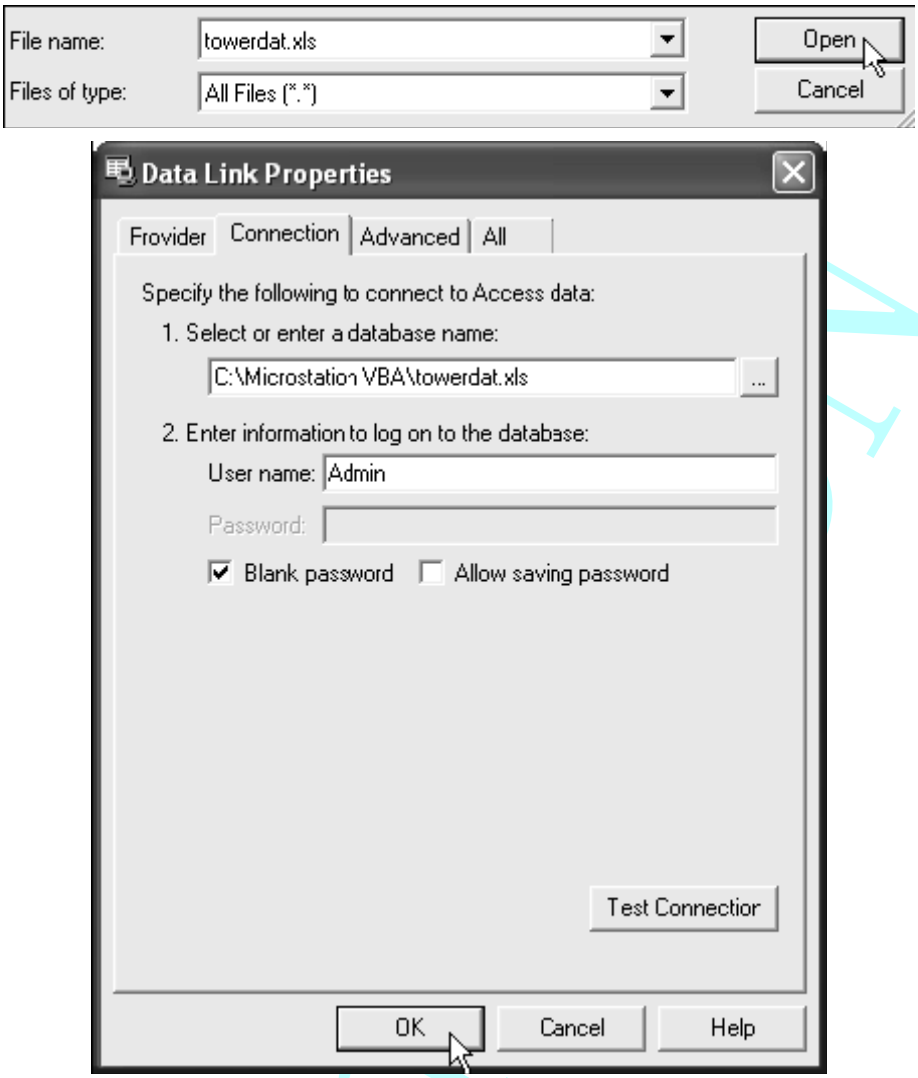
建立一个 UDL 文件，叫做 Excel.udl。

右图是 UDL 文件的设置。

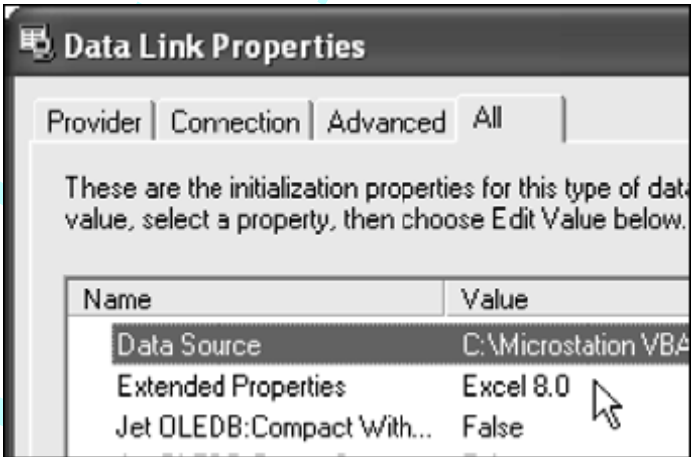


虽然 Jet 驱动程序主要用于连接 Microsoft Access 数据库，但也能用它连接 Excel。

在 Connection 选项卡中需要浏览到 Excel 文件。缺省情况下，Browse(浏览)按钮打开的对话框查找的是 Microsoft Access 数据库(.mdb)。我们可以在“Files of Type(文件类型)”复合框中选择“*.*”然后选择 Excel(.xls)文件。



在使用 Microsoft Access 文件时，至此就完成了对 UDL 文件的信息输入。但使用 Microsoft Excel 文件时，还需要作更多的改变。



在 All(所有)选项卡中需要给“Extended Properties(扩展属性)”赋值“Excel 8.0”。这样做了以后，Jet 驱动程序就知道它正在使用一个 Microsoft Excel 文件。如果忘记添加这个重要的“Extended Properties”，将会看到错误弹出，因为没

有这个值，UDL 文件将被作为 Microsoft Access 数据库对待。

在开始使用 Excel 文件前，我们想使其看起来尽可能像一个数据库。所以，删去文件顶部几行和第一列，只留下标题行。如下图所示：

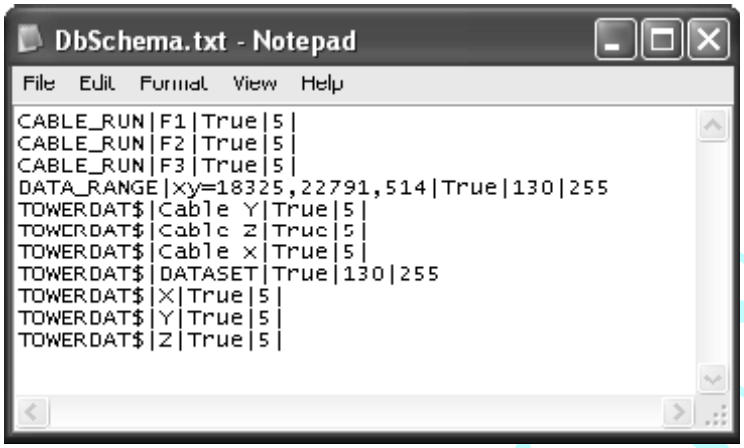
	A	B	C	D	E	F	G
1	X	Y	Z	DATASET	Cable x	Cable Y	Cable Z
2	18325	22791	514	xy=18325,22791,514	18290	22929	894
3	18414	22609	526	xy=18414,22609,526	18379	22747	906
4	18427	22444	465	xy=18427,22444,465	18392	22582	845

由于操作 Excel 是个新领域，那么如何能知道要操作哪些东西呢？让我们修改前一个“OpenSchema”过程来操作 Excel.udl 文件。

```
Sub TestSchemaC()
    Dim myDB As New ADODB.Connection
    Dim myRS As New ADODB.Recordset
    Dim FFile As Long
    myDB.Open "file name=C:\MicroStation VBA\Excel.udl"
    Set myRS = myDB.OpenSchema(adSchemaColumns)
    FFile = FreeFile
    Open "C:\DbSchema.txt" For Output As #FFile
    While myRS.EOF = False
        Print #FFile, myRS("TABLE_NAME") & "|" & _
                      myRS("COLUMN_NAME") & "|" & _
                      myRS("IS_NULLABLE") & "|" & _
                      myRS("DATA_TYPE") & "|" & _
                      myRS("CHARACTER_MAXIMUM_LENGTH")
        myRS.MoveNext
    Wend
    Close #FFile
End Sub
```

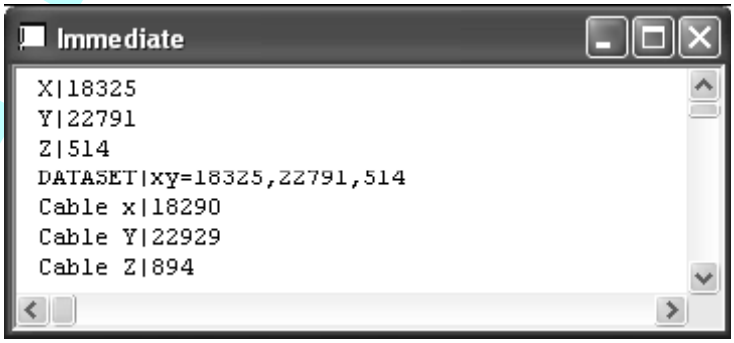
运行这个过程，一个包含 Excel 文件数据库架构的新文件被创建。让我们打开这个新文件看一看，这个文件的名字是 DbSchema.txt。

China BDI



从文件中能看到一个叫做“TOWERDAT$”的表，这很有趣。当我们查看 Excel 文件时能找到叫做“TOWERDAT”的工作表，看起来大有希望。让我们看一看能对“TOWERDAT$”做些什么。

```
Sub TestDBExcelA()
    Dim myDB As New ADODB.Connection
    Dim myRS As New ADODB.Recordset
    Dim myField As Field
    myDB.Open "file name=C:\MicroStation VBA\Excel.udl"
    myRS.Open "[TOWERDAT$]", myDB, _
    adOpenDynamic, adLockOptimistic
    While myRS.EOF = False
        For Each myField In myRS.Fields
            Debug.Print myField.Name & "|" & myField.Value
        Next
        Debug.Print vbCr
        myRS.MoveNext
    Wend
End Sub
```



现在取得了一些进展。我们能不打开 Microsoft Excel 而取得 Excel 文件中的数据。就此而言，计算机上甚至不需要安装 Excel。

让我们看一看还能做什么。

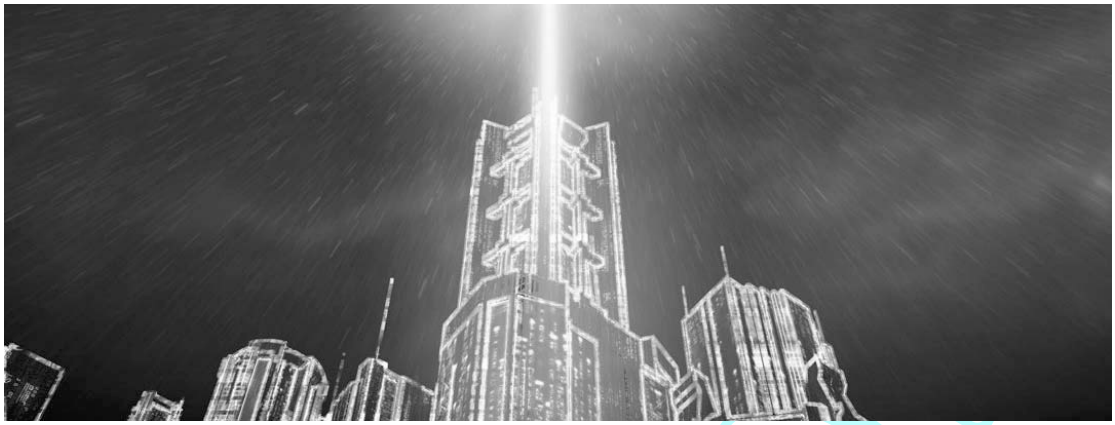
```
Sub TestDBExcelB()
    Dim myDB As New ADODB.Connection
    Dim myRS As New ADODB.Recordset
    Dim CenPt As Point3d
    Dim myCircle As ArcElement
    Dim RotMatrix As Matrix3d
    myDB.Open "file name=C:\MicroStation VBA\Excel.udl"
    myRS.Open "[TOWERDAT$]", myDB, _
                adOpenDynamic, adLockOptimistic
    While myRS.EOF = False
        CenPt.X = myRS("X")
        CenPt.Y = myRS("Y")
        CenPt.Z = myRS("Z")
        Set myCircle = CreateArcElement2(Nothing, _
                        CenPt, 4, 4, RotMatrix, 0, 360)
        ActiveModelReference.AddElement myCircle
        myRS.MoveNext
    Wend
End Sub
```

现在，我们正用 ActiveX 数据对象根据 Microsoft Excel 文件中的数据在 MicroStation 中绘制图形。

本章回顾

ActiveX 数据对象为我们提供了操作数据的工具。这些数据有时存储在数据库中，也可以“存储”在 Internet 的 RSS 文件中，甚至能存储在 Microsoft Excel 文件中。无论数据在什么地方，都能用 ActiveX 数据对象来检索数据。连接数据源的过程因使用 UDL 文件而大大简化。一旦连接成功，Connection 和 Recordset 对象就能用于检索、维护、编辑和添加数据。

China BDN



36. MicroStation 用 VBA 操作 Mathcad

每当一家公司开放其产品定制之时，消费者就赢得了胜利，Mathcad 就是这样的一种产品。Mathcad 工作表能用于计算，然后再通过 VBA 的使用把信息“传递”给 MicroStation。Mathcad 当然不是奔特力公司的产品，在本书中无论如何也不应当详细讨论 Mathcad。Microsoft Excel 以及在本书中讨论过的任何其它第三方产品也都是如此。我们已经说过，Mathcad 就像 Excel 一样能被定制，我们还能打开 Mathcad 与 MicroStation 之间的交流通道，从而生成一个集成的解决方案。

本章包括：

- Mathcad 的简要介绍
- 添加引用以及使用对象浏览器
- 与 Mathcad 通讯的基本宏
- 区域对象 — 所有计算的基础
- Mathcad 对象模型
- 从 Mathcad 中驱动 MicroStation 几何图形

Mathcad 的简要介绍

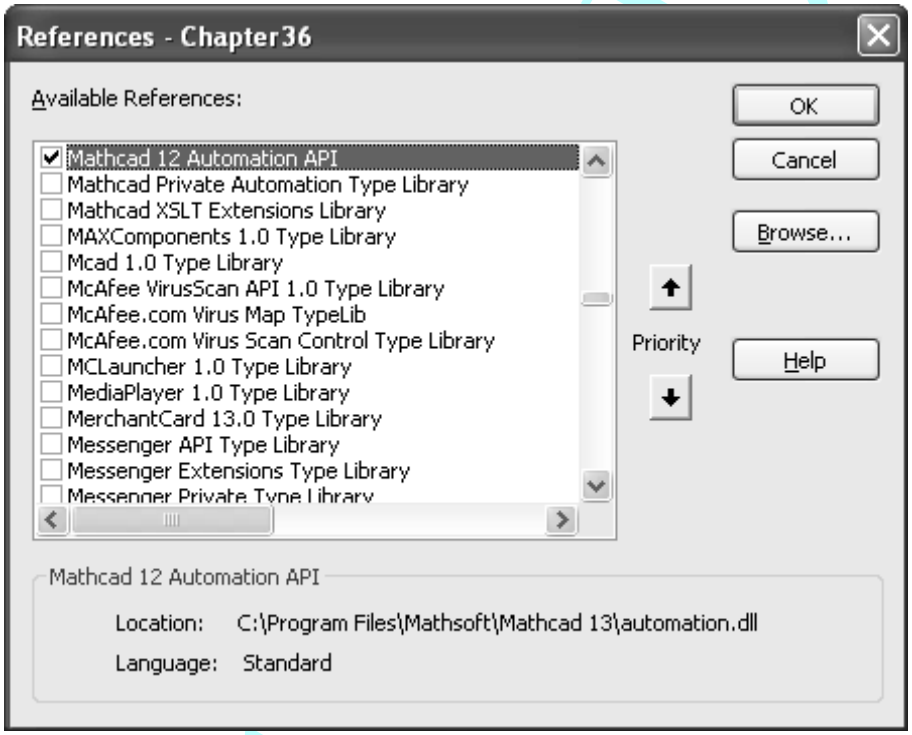
Mathcad 能让我们在画板类型的环境中执行(简单和复杂)计算功能。在这些计算中能使用变量，同时还使用了标准数学命名法以使 Mathcad 中的公式看起来就像参考资料中使用的那样。Mathcad 提供有数百个操作符和内置函数用来解决技术问题，Mathcad 既能执行数字计算也能进行符号式的公式推导。它会自动

跟踪和转换单位，能以标量、矢量和矩阵进行操作。Mathcad 不仅理解单位，而且还特别关注于从一种单位到另一种单位间的转换(英寸到米、加仑到升等)。如果需要用到一个 Mathcad 没有的单位(如：一掌)，还能将其添加进来。

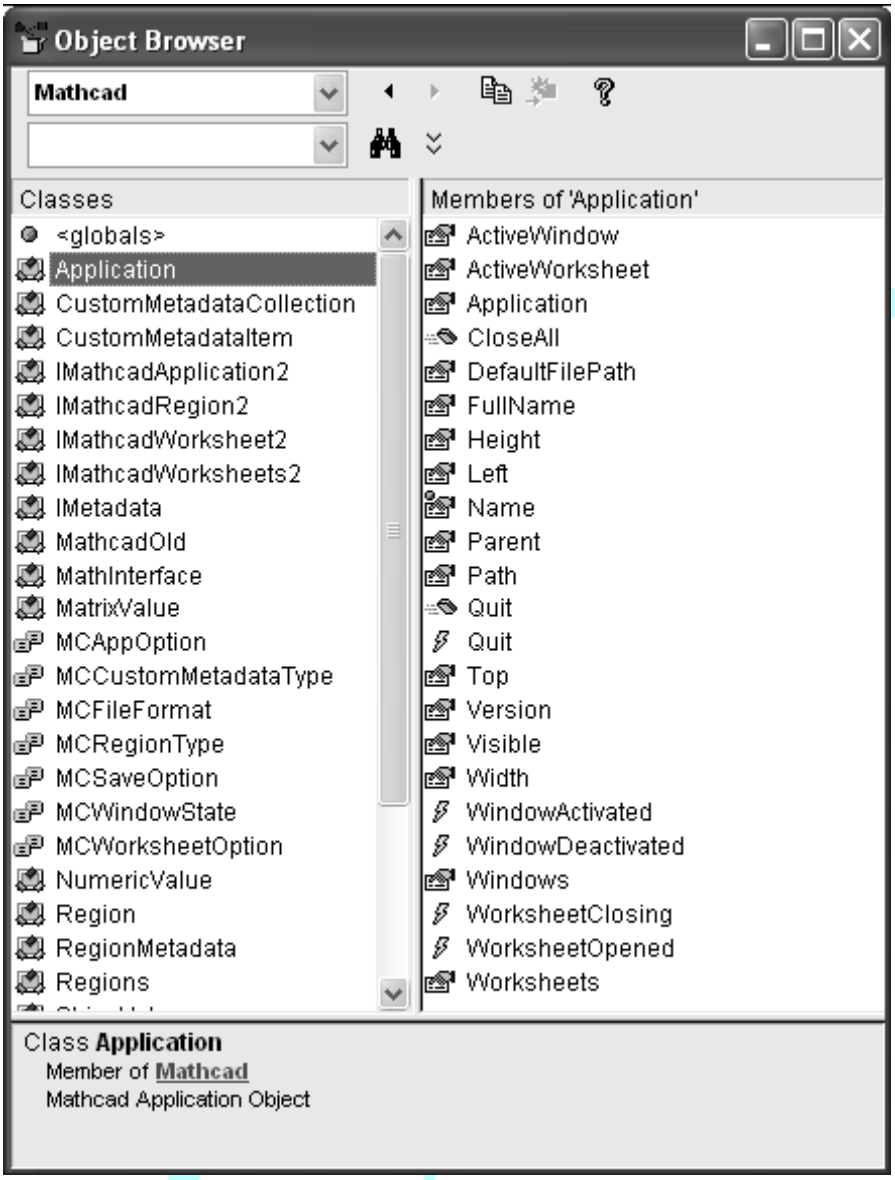
关于 Mathcad 的更多信息可以通过访问 www.mathcad.com 找到。

添加引用以及使用对象浏览器

在试图以任何方式与 Mathcad 进行通讯前，需要在 VBA 中添加一个引用。通过 VBA 菜单 **Tools > References(工具 > 引用)**可做到这一点。



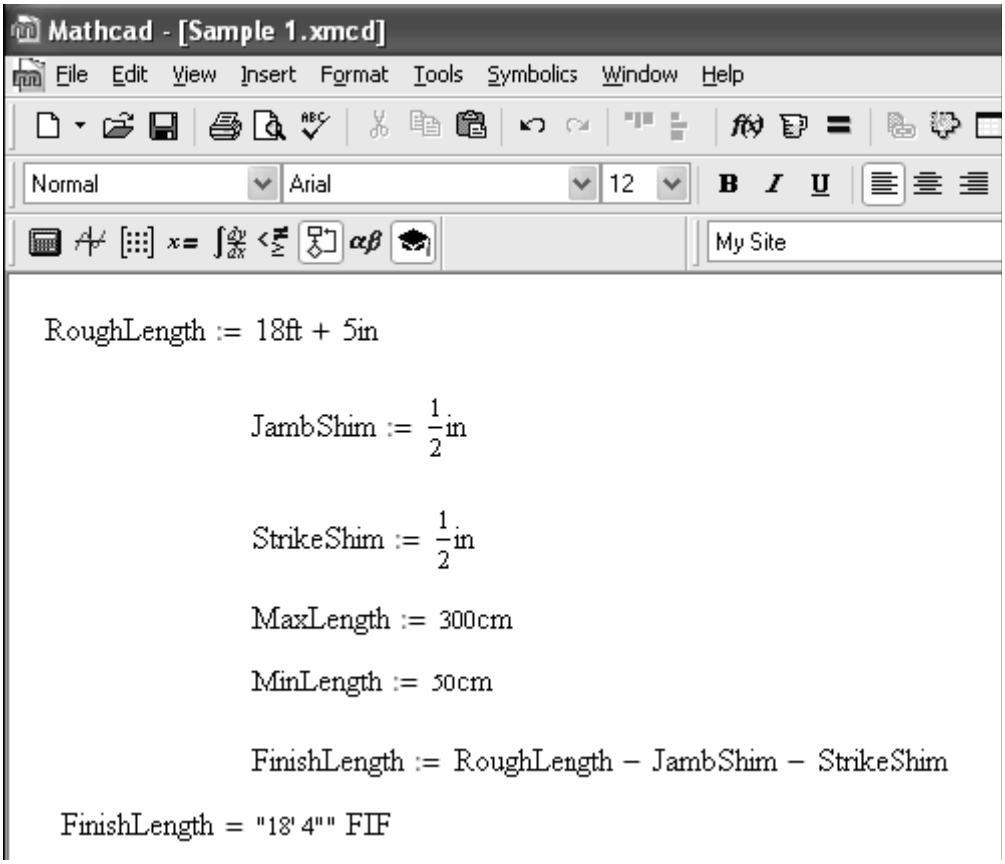
加入引用后可以用 VBA 对象浏览器来“浏览”Mathcad 的对象模型。



在对象浏览器顶部的复合框中选择“Mathcad”来过滤类后，可以看到 Mathcad 对象模型中的 Application 类，这是 Mathcad 中的顶级类。还能看到一个 ActiveWorksheet 对象。这两个对象足够我们开始使用 Mathcad 了。

我们开始在 VBA 中编写一个宏并将看到该宏所完成的功能。在此之前需要启动 Mathcad 并打开一个工作表。就使用随本书附赠的 CD 上的文件“Sample 1.xmcd”吧。

下图是 Mathcad 工作表的一部分。

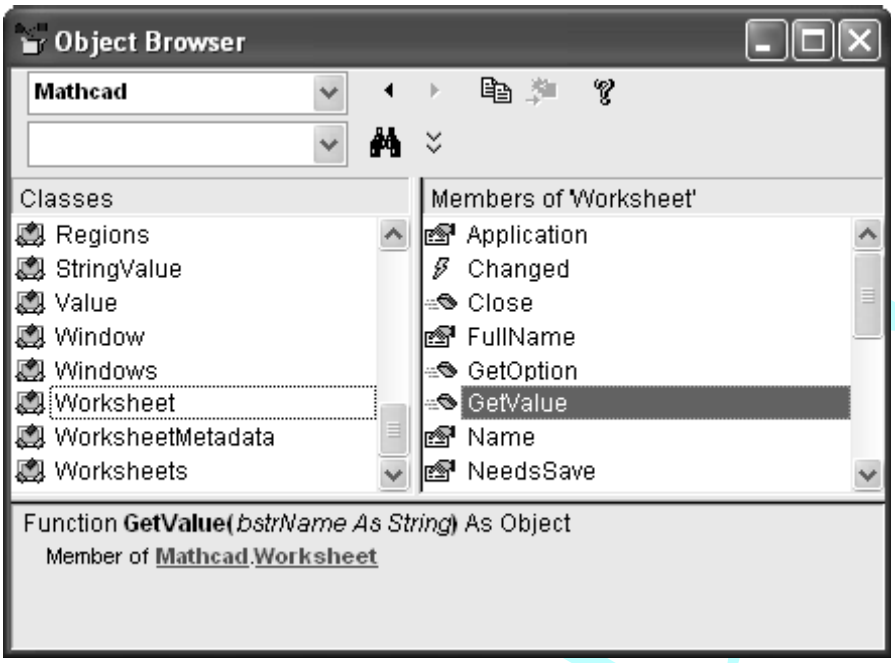


从上图能看到存在变量“RoughLength”、“JambShim”、“MaxLength”和“FinishLength”等。我们在 MicroStation VBA 环境中编写一个宏来取得 Mathcad 中变量 RoughLength 的值。

```
Sub TestMathcadA()
    Dim myMCA As Mathcad.Application
    Dim myMCW As Mathcad.Worksheet
    Set myMCA = GetObject(, "Mathcad.Application")
    Set myMCW = myMCA.ActiveWorksheet
End Sub
```

在对象浏览器中看到过 Application 和 ActiveWorksheet 对象。如果运行宏 TestMathcadA 的话，会发现代码执行后并没有取得 Mathcad 中变量 **RoughLength** 的值。

让我们再看一眼 VBA 中的对象浏览器。



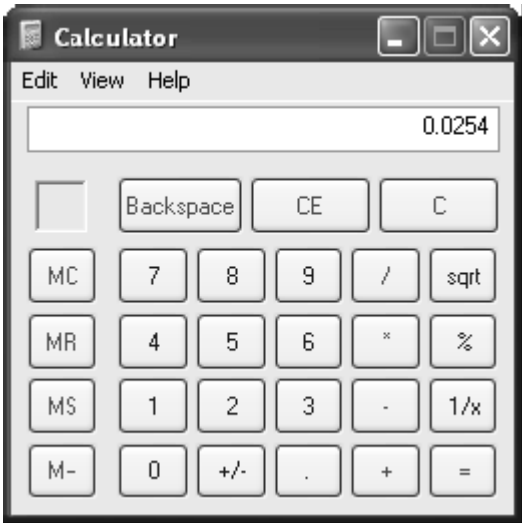
我们想取得工作表中变量的值，因此我们在类列表中选择“Worksheet”并开始查找能给出值的一个属性或方法。这样做的时候会看到一个叫做 GetValue 的方法，它使用一个参数“bstrName as String”。该方法返回一个对象，但是什么种类的对象呢？在对象浏览器中再次查看会找到一个叫做“Value”的对象(类类列表中)。让我们试一试 Value 对象。

```
Sub TestMathcadA()
    Dim myMCA As Mathcad.Application
    Dim myMCW As Mathcad.Worksheet
    Dim myMCV As Mathcad.Value
    Set myMCA = GetObject(, "Mathcad.Application")
    Set myMCW = myMCA.ActiveWorksheet
    Set myMCV = myMCW.GetValue("RoughLength")
    MsgBox myMCV.AsString & vbCr & myMCV.Type
End Sub
```



现在，除了连接到了 Mathcad 的 Application 和 ActiveWorksheet，还取得了参数 RoughLength 的值对象，并在消息框中显示其“AsString”和“Type”属性。在 Mathcad 中查看 RoughLength 属性会看到其值为 18’-5”。但运行完宏 TestMathcadA 后显示了一个值为 5.6134。为什么是这样呢？

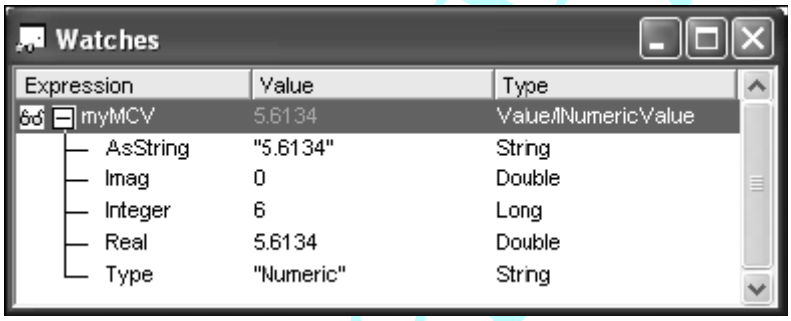
这肯定是一个单位问题。让我们用 5.6134 除以 221 英寸看得到的是什么。



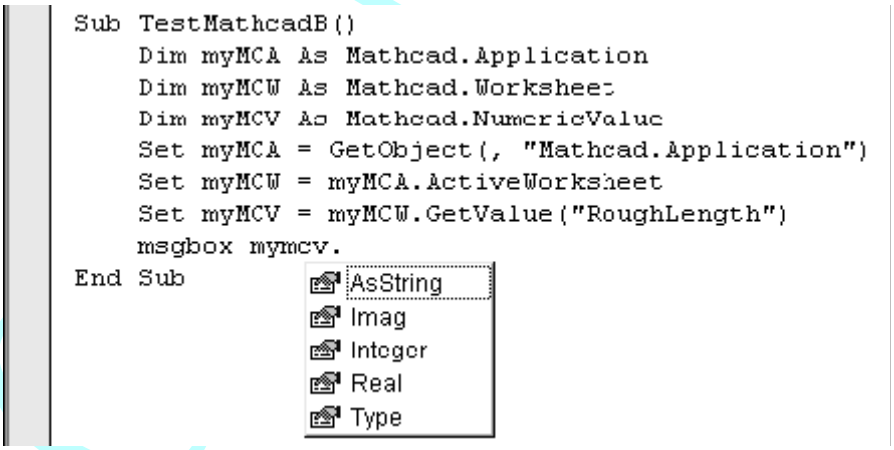
看到这个魔力数字 0.0254 后我们就知道正在处理英寸到米的转换问题。

在对象浏览器中查看函数 GetValue 能看到它返回了一个对象。当使用一个声明为“Value”对象的变量时，我们能取得“Value”对象的“AsString”和“Type”属性。还能做更多的事情吗？

对变量 **myMCV** 添加监控，看一看到属于 Value 对象的其它属性。



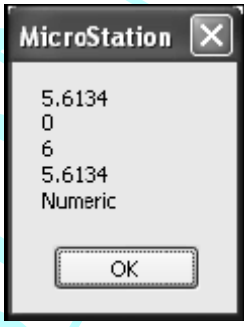
变量 **myMCV** 被声明为“Value”，但在这里能看出实际返回的是一个“NumericValue”(在类型列查看)对象。让我们对代码稍作修改，把变量 **myMCV** 声明为“NumericValue”对象看一看会发生什么。



在我们开始写代码在消息框中显示属性前能看到更多可用的属性，因为变量被声明为了“NumericValue”对象。

```
Sub TestMathcadB()
    Dim myMCA As Mathcad.Application
```

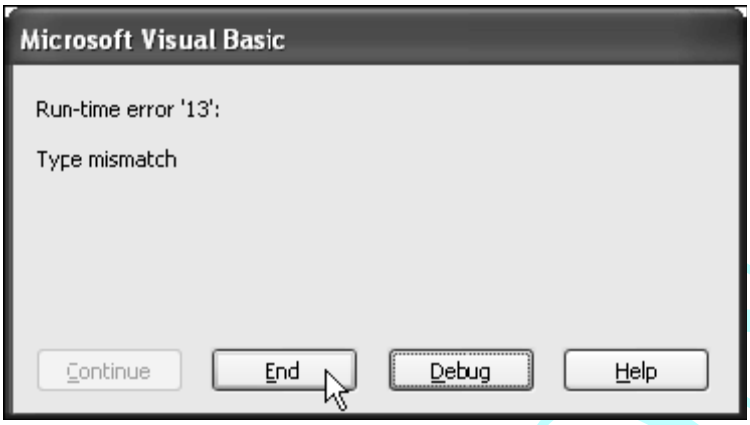
```
    Dim myMCW As Mathcad.Worksheet
    Dim myMCV As Mathcad.NumericValue
    Set myMCA = GetObject(, "Mathcad.Application")
    Set myMCW = myMCA.ActiveWorksheet
    Set myMCV = myMCW.GetValue("RoughLength")
    MsgBox myMCV.AsString & vbCr & _
           myMCV.Imag & vbCr & _
           myMCV.Integer & vbCr & _
           myMCV.Real & vbCr & _
           myMCV.Type
End Sub
```



我们现在正取得进展。在这个消息框中能看到“AsString”和“Real”属性值为 5.6134，“Integer”属性值为 6，“Type”属性值为“Numeric”。让我们进一步修改代码。

```
Sub TestMathcadC()
    Dim myMCA As Mathcad.Application
    Dim myMCW As Mathcad.Worksheet
    Dim myMCV As Mathcad.StringValue
    Set myMCA = GetObject(, "Mathcad.Application")
    Set myMCW = myMCA.ActiveWorksheet
    Set myMCV = myMCW.GetValue("RoughLength")
    MsgBox myMCV.AsString & vbCr & _
           myMCV.Imag & vbCr & _
           myMCV.Integer & vbCr & _
           myMCV.Real & vbCr & _
           myMCV.Type
End Sub
```

不是把 myMCV 声明为“NumericValue”而是声明为“StringValue”。会出现什么情况？



这次没有得到 Mathcad 变量“RoughLength”的值，而是看到了一个类型不匹配的错误对话框。为什么是这样？因为我们在处理一个数字值而不是一个字符串值。

从上面的练习中可以学到：当从 Mathcad 检索一个变量值时，变量值的类型存在某种不确定性。应当把我们要求的值的变量声明为“Value”对象，然后查看其“Type”属性并使用适当的值对象。下面是这样的一个例子：

```
Sub TestMathcadD()
    Dim myMCA As Mathcad.Application
    Dim myMCW As Mathcad.Worksheet
    Dim myMCV As Mathcad.Value
    Dim myMCNV As Mathcad.NumericValue
    Dim myMCSV As Mathcad.StringValue
    Set myMCA = GetObject(, "Mathcad.Application")
    Set myMCW = myMCA.ActiveWorksheet
    Set myMCV = myMCW.GetValue("RoughLength")
    Select Case myMCV.Type
    Case "Numeric"
        Set myMCNV = myMCV
        MsgBox myMCNV.AsString & vbCr & _
               myMCNV.Imag & vbCr & _
               myMCNV.Integer & vbCr & _
               myMCNV.Real & vbCr & _
               myMCNV.Type
```

```
    Case "String"
        Set myMCSV = myMCV
        MsgBox myMCSV.AsString & vbCr & _
               myMCSV.Type & vbCr & _
               myMCSV.Value
    End Select
End Sub
```

我们一开始使用通用的“Value”对象，然后查看 Value.Type 属性。如果它是“Numeric”，就使用“Mathcad.NumericValue”对象。如果它是“String”，就使用“Mathcad.StringValue”对象。

与 Mathcad 通讯的基本宏

对 Mathcad 越熟悉，就越能认识到其处理复杂计算的优势。虽然 Mathcad 处理的计算非常复杂，但与 Mathcad 的通讯一点也不复杂。

让我们看几个在不同方面与 Mathcad 通讯的宏。这些宏仍然利用“Sample 1”工作表。

```
Sub TestMathcadE()
    Dim myMCA As Mathcad.Application
    Dim myMCW As Mathcad.Worksheet
    Dim myMCV As Mathcad.Value
    Dim myMCNV As Mathcad.NumericValue
    Set myMCA = GetObject(, "Mathcad.Application")
    Set myMCW = myMCA.ActiveWorksheet
    Set myMCV = myMCW.GetValue("RoughLength")
    Set myMCNV = myMCV
    MsgBox "Rough Length: " & myMCNV.Real / 0.0254 & " Inches."
End Sub
```

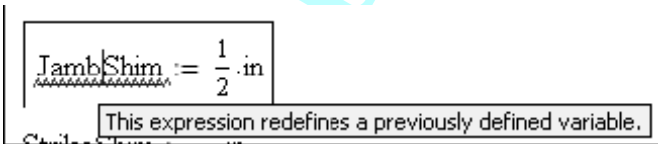


TestMathcadE 在将 RoughLength 变量转换为英寸后在消息框中显示其值。

前一个例子从 Pathcad 中取信息。下面的例子改变变量值，重新计算工作表，然后再取出值。

```
Sub TestMathcadF()
    Dim myMCA As Mathcad.Application
    Dim myMCW As Mathcad.Worksheet
    Dim myMCNV As Mathcad.NumericValue
    Set myMCA = GetObject(, "Mathcad.Application")
    Set myMCW = myMCA.ActiveWorksheet
    myMCW.SetValue "JambShim", 0.375 * 0.0254
    myMCW.Recalculate
    Set myMCNV = myMCW.GetValue("FinishLength")
    MsgBox "Finish Length: " & myMCNV.Real / 0.0254
End Sub
```

这个宏运行后在消息框中看到一个值，但好像没有进行计算。当查看 Mathcad 工作表时能发现，在我们试图给 **JambShim** 变量所赋的值上出了问题。



为什么会发生这种情况？Mathcad 先计算通过 ActiveX 自动化给变量所赋的值，然后才计算在 Mathcad 中定义的那些值。所以，当试图对 **JambShim** 设置一个值时，除了在工作表中所示的值以外，Mathcad 还想使用来自 SetValue 调用的值，这就引发了问题。

那么，知道了错误发生的原因，如何修正它呢？一种解决方法是删去赋给 **JambShim** 变量的值“1/2 英寸”，这样就不再出现重复赋值了。而且不给 **JambShim** 赋以值“0.375 * 0.0254”，而是赋给值“3/8”。下一步是改变 **FinishLength** 变量的公式，“转换”**JambShim** 的值为英尺和英寸。为了更好地演示这种变化，我们先看代码再看工作表。

```
Sub TestMathcadF2()
    Dim myMCA As Mathcad.Application
    Dim myMCW As Mathcad.IMathcadWorksheet2
    Dim myMCNV As Mathcad.NumericValue
    Set myMCA = GetObject(, "Mathcad.Application")
    Set myMCW = myMCA.ActiveWorksheet
```

```
    myMCW.SetValue "JambShim", "3/8"
    myMCW.Recalculate
    Set myMCNV = myMCW.GetValue("FinishLength")
    MsgBox "Finish Length: " & myMCNV.Real / 0.0254
End Sub
```

$$JambShim := \blacksquare$$

$$StrikeShim := \frac{1}{2}in$$

$$MaxLength := 300cm$$

$$MinLength := 50cm$$

$$FinishLength := RoughLength - FIF(JambShim) - StrikeShim$$

现在宏 TestMathcadF2 工作正常，因为 **JambShim** 的值在代码中设置，并且使用 Mathcad 函数 FIF 将提供的值(3/8)转换为英尺和英寸。

GetValue、SetValuve 和 Recalculate 构成了取得现有 Mathcad 工作表并对之进行修改和检索计算的基础。只要能确保 Mathcad 仅从一个来源给变量赋值且对所设置的值执行了正确的单位转换，那么 Mathcad 公式和我们的 VBA 程序将会每次都给出一致的正确结果。

每个值、变量、公式和计算都作为一个“区域(Region)”存储在 Mathcad 工作表中。

```
Sub TestMathcadG()
    Dim myMCA As Mathcad.Application
    Dim myMCW As Mathcad.Worksheet
    Dim myMCR As Mathcad.Region
    Set myMCA = GetObject(, "Mathcad.Application")
    Set myMCW = myMCA.ActiveWorksheet
    MsgBox myMCW.Regions.Count
    For Each myMCR In myMCW.Regions
        MsgBox myMCR.Type
    Next
End Sub
```

Region.Type 属性是一个叫做“MCRegionType”的枚举值。下面是其值：

mcBitmapRegion = 2

mcMathRegion = 1

mcMetafileRegion = 3

mcOLERegion = 4

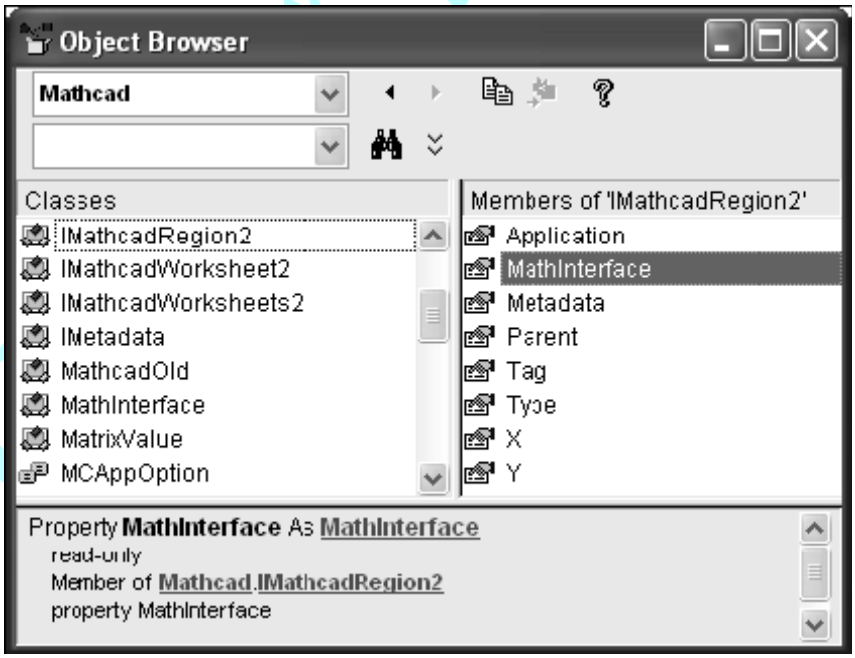
mcTextRegion = 0

在“Sample 1”工作表中看到了许多“mcMathRegion”区域。“mcMathRegion”是包含数字计算和变量赋值的区域。即使一个变量保存的是字符串值，它也属于一个 MathRegion(数学区域)。

区域对象 — 所有计算的基础

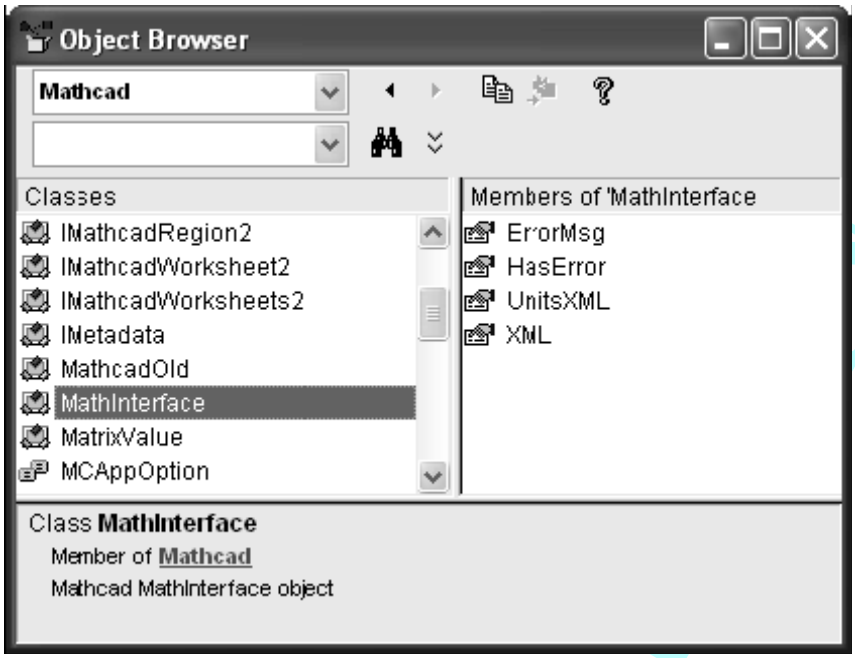
我们刚刚看过了工作表的区域对象。区域对象为我们提供了一些非常基本的信息，如类型和区域位置，区域在工作表中的位置是通过 X 和 Y 属性表达的。但计算是如何存储在 Mathcad 中的呢？考虑到 Mathcad 所做的一些复杂计算，不难想象要存储这些复杂的计算信息绝非一个简单属性所能完成。实际情况确实如此。

除了把变量声明为“Region”，还能把变量声明为“IMathcadRegion2”。这样做时，有一些附加的属性可为我们所用。



下面是 IMathcadRegion2 对象的属性，其中之一是“MathInterface(数学接

口)”属性。在类列表的“MathInterface”上点击能看到如下信息：



MathInterface 对象具有属性“XML”，这听起来很有趣。让我们试着运行如下宏：

```
Sub TestMathcadH()
    Dim myMCA As Mathcad.Application
    Dim myMCW As Mathcad.Worksheet
    Dim myMCR As Mathcad.IMathcadRegion2
    Dim myMCI As Mathcad.MathInterface
    Set myMCA = GetObject(, "Mathcad.Application")
    Set myMCW = myMCA.ActiveWorksheet
    For Each myMCR In myMCW.Regions
        Set myMCI = myMCR.MathInterface
        Debug.Print myMCI.XML
    Next
End Sub
```

运行这个宏以后，每个 IMathcadRegion2 对象的每个 MathInterface 的 XML 属性被写入到立即窗口中。下面是运行 TestMathcadE 后输出的一个例子：

```
<ml:define xmlns:ml="http://schemas.mathsoft.com/math20">
    <ml:id xml:space="preserve">RoughLength</ml:id>
```

```
    <ml:apply>
        <ml:plus/>
        <ml:apply>
            <ml:mult style="auto-select"/>
            <ml:real font="0">18</ml:real>
            <ml:id xml:space="preserve">ft</ml:id>
        </ml:apply>
        <ml:apply>
            <ml:mult style="auto-select"/>
            <ml:real font="0">5</ml:real>
            <ml:id xml:space="preserve">in</ml:id>
        </ml:apply>
    </ml:apply>
</ml:define>
```

这是工作表中与 **RoughLehgth** 变量相关的数据。让我们再看几个例子：

```
<ml:define xmlns:ml="http://schemas.mathsoft.com/math20">
    <ml:id xml:space="preserve">JambShim</ml:id>
    <ml:placeholder/>
</ml:define>
<ml:define xmlns:ml="http://schemas.mathsoft.com/math20">
    <ml:id xml:space="preserve">StrikeShim</ml:id>
    <ml:apply>
        <ml:mult style="auto-select"/>
        <ml:apply>
            <ml:div/>
            <ml:real>1</ml:real>
            <ml:real>2</ml:real>
        </ml:apply>
        <ml:id xml:space="preserve">in</ml:id>
    </ml:apply>
```


China BDN

```
</ml:define>
<ml:define xmlns:ml="http://schemas.mathsoft.com/math20">
    <ml:id xml:space="preserve">MaxLength</ml:id>
    <ml:apply>
        <ml:mult style="auto-select"/>
        <ml:real>300</ml:real>
        <ml:id xml:space="preserve">cm</ml:id>
    </ml:apply>
</ml:define>
<ml:define xmlns:ml="http://schemas.mathsoft.com/math20">
    <ml:id xml:space="preserve">FinishLength</ml:id>
    <ml:apply>
        <ml:minus/>
        <ml:apply>
            <ml:minus/>
            <ml:id xml:space="preserve">RoughLength</ml:id>
            <ml:apply>
                <ml:id xml:space="preserve">FIF</ml:id>
                <ml:id xml:space="preserve">JambShim</ml:id>
            </ml:apply>
        </ml:apply>
        <ml:id xml:space="preserve">StrikeShim</ml:id>
    </ml:apply>
</ml:define>
```

在这里能看到工作表中对变量 **FinishLength** 的计算是，**RoughLength** 减去 **JambShim** 的英尺和英寸值再减去 **StrikeShim**。

Mathcad 对象模型

让我们看几个在 Mathcad 中可用的 Mathcad 对象。正如大多数 ActiveX 自动化对象模型一样，Mathcad 的顶级对爹也是 Application 对象。

Application

ActiveWindow	**Height**	**Top**
ActiveWorksheet	**Left**	**Version**
Application	**Name**	**Visible**
CloseAll	**Parent**	**Width**
DefaultFilePath	**Path**	**Windows**
FullName	**Quit**	**Worksheets**

除了 Application 对象外，Mathcad 还有一个 IMathcadApplication2 对象。在引入新的属性和方法后，为了保持与旧的对象模型的兼容性，通常需要建立另一类应用类型对象。在这里要注意的是 GetOption 方法、HWND 属性(常用于 Windows API 编程)和 SetOption 方法。

IMathcadApplication2

Active	**Height**	**Top**
ActiveWindow	**HWND**	**Version**
ActiveWorksheet	**Left**	**Visible**
Application	**Name**	**Width**
CloseAll	**Parent**	**Windows**
DefaultFilePath	**Path**	**Worksheets**
FullName	**Quit**	
GetOption	**SetOption**	

关于 Mathcad 对象模型的更详细信息，VBA 中的对象浏览器能给出对象、属性、常数和枚举的名称及成员。

从 Mathcad 中驱动 MicroStation 几何图形

我们看了几个用 VBA 获取和设置 Mathcad 变量的例子。让我们看一个用 Mathcad 来驱动建立 MicroStation 几何图形的例子。

$Width := 1ft + 5in$ $HoleDia := 12mm$

$Height := 2ft + 2in$ $MaxSpacing := 50mm$

$OutsideBuffer := 50mm$

$$CalculateQuantity(Overall) := \frac{Overall - OutsideBuffer \cdot 2}{HoleDia + MaxSpacing}$$

$QtyWidth := trunc(CalculateQuantity(Width)) + 1$

$QtyHeight := trunc(CalculateQuantity(Height)) + 1$

$$ActualSpacing(Overall) := \frac{Overall - OutsideBuffer \cdot 2}{QtyWidth} - HoleDia$$

$SpacingXIn := ActualSpacing(Width)$

$SpacingYIn := ActualSpacing(Height)$

$HoleDiaIn := HoleDia$

$OutsideBufferIn := OutsideBuffer$

$WidthIn := Width$

$HeightIn := Height$

$WidthIn = 17\,in$

$HeightIn = 26\,in$

$QtyWidth = 6$

$QtyHeight = 10$

$SpacingXIn = 1.705\,in$

$SpacingYIn = 3.205\,in$

$HoleDiaIn = 0.472\,in$

$OutsideBufferIn = 1.969\,in$

我们的目标是在 MicroStation 中绘制一个带有孔图案的二维平板。X 方向的孔间距必须相等，Y 方向的孔间距也必须相等(但 X 间距可以不同于 Y 间距)。我们想在平板的外围留有一个“缓冲区”，想能指定板的宽度和高度、孔的直径、孔间最大距离、板外围“缓冲”区。

我们能查看完目标后决定在代码中执行所有的计算。但在这种情形下用 Mathcad 进行计算有几个好处。

1、使用 Mathcad 时能够以各种不同的单位来输入平板参数。例如，孔直径能以毫米输入，板宽和板高能以带小数的英尺输入；“缓冲区”能以码输入，最大间距能以腕尺(cubit)输入。输入设计参数所用的单位几乎没有任何影响，因为 Mathcad 为我们进行了单位转换。再者，当从工作表中检索任何变量的值时，可以指定任何的单位的返回值；

2、可以在 Mathcad 中改变计算而不用更改代码。这样就能在 Mathcad 中实现不同的设计规则，只要是“输出”同样的参数，代码就能继续顺利运行；

3、Mathcad 能用标准的数学符号执行计算，这对 VBA 来说却是一个极大的挑战。

从 Mathcad 工作表中能看到用户输入的两类语句：一类使用了“X=5”符号，另一类使用了“X:=5”符号(注意冒号)。当在 Mathcad 中看到“:=”时，我们知道输入的是一个带有或用于计算的公式；当看到“=”(没有冒号)时，我们知道正在显示变量的值或公式的结果。随后将看到区分这两“类”语句的重要性。

在这种情况下，把“逻辑”与结果信息的“演示”分开是明智的。现在所需要的只是用代码从 Mathcad 中检索结果并在 MicroStation 中绘制带有孔图案的平板。

在本章的前一个例子中，我们使用 Mathcad 工作表对象的 GetValue 方法取得 Mathcad 变量的值。我们在 VBA 中执行了一个从米到英寸的转换。由于 Mathcad 已经能为我们进行单位转换了，所以用 Mathcad 转换值代替代码中执行的单位转换是有意义的。

缺省情况下，Mathcad 以米来存储所有长度类型的值，GetValue 也以这种缺省的测量单位来检索值。除了用 GetValue 在 Mathcad 中检索值以外，还能通过查看每个区域的 IMathcadRegion2 对象以在 Mathcad 工作表中所示的单位来检索值。

查看 Mathcad 区域对象会发现，有好几种类型的区域。我们在此最感兴趣的两个是“define”和“eval”类型。关于这些区域的信息是用 XML 格式存储的。下面是我们所感兴趣的这两类区域的一些例子：

```
<ml:define xmlns:ml="http://schemas.mathsoft.com/math20">
    <ml:id xml:space="preserve">HeightIn</ml:id>
    <ml:id xml:space="preserve">Height</ml:id>
</ml:define>
<ml:eval xmlns:ml="http://schemas.mathsoft.com/math20">
    <ml:id xml:space="preserve">WidthIn</ml:id>
    <ml:unitOverride><ml:id xml:space="preserve">in</ml:id>
    </ml:unitOverride>
    <result xmlns="http://schemas.mathsoft.com/math20">
        <unitedValue>
            <ml:real>17</ml:real>
            <unitMonomial xmlns="http://schemas.mathsoft.com/units10">
                <unitReference unit="inch"></unitReference>
            </unitMonomial>
        </unitedValue>
    </result>
</ml:eval>
```

这里能看到一个“define”区域和一个“eval”区域。注意“eval”区域具有一个值为“WidthIn”的“id”和一个值为“17”的“real”。这些是我们感兴趣的区域值。

要想从 Mathcad 中检索一个区域，最好是在一个单独的函数中一次性将所有区域都检索出来，然后分析它们以找到要查找的值。我们以函数 GetAllRegions 开始。

```
Function GetAllRegions() As String
    On Error GoTo errhnd
    Dim myMCA As Mathcad.IMathcadApplication2
    Dim myMCW As Mathcad.IMathcadWorksheet2
    Dim myMCR As Mathcad.IMathcadRegion2
    Dim myMCI As Mathcad.MathInterface
    Dim DomStrings() As String
```

```
    ReDim DomStrings(0)
    Set myMCA = GetObject(, "Mathcad.Application")
    Set myMCW = myMCA.ActiveWorksheet
    Dim myRegs As Mathcad.Regions
    Set myRegs = myMCW.Regions
    For Each myMCR In myRegs
        Set myMCI = myMCR.MathInterface
        'Debug.Print myMCI.XML
        'Debug.Print myMCI.UnitsXML
        DomStrings(UBound(DomStrings)) = myMCI.XML
        ReDim Preserve DomStrings(UBound(DomStrings) + 1)
        DomStrings(UBound(DomStrings)) = myMCI.UnitsXML
        ReDim Preserve DomStrings(UBound(DomStrings) + 1)
    Next
    GetAllRegions = Join(DomStrings, vbCr & vbCr & vbCr)
    Exit Function
errhnd:
    MsgBox Err.Number & vbCr & Err.Description
End Function
```

GetAllRegions 返回一个由区域 XML 数据组成的字符串，这些数据以三个回车符做为分隔。我们在过程 GetEvals 中使用这些字符将返回值分割回区域。

GetEvals 从 GetAllRegions 中取得返回字符串，提取 Eval 区域，仅返回以等号(=)分隔的每个 Eval 区域的名字和值。

```
Function GetEvals() As Variant
    Dim myDOM As New DOMDocument60
    Dim myDOM2 As New DOMDocument60
    Dim myNode As IXMLDOMNode
    Dim myNodeList As IXMLDOMNodeList
    Dim EvalName As String
    Dim EvalValue As Double
```

```
    Dim AllRegions As String
    Dim xSplit() As String
    Dim EvalArray() As String
    ReDim EvalArray(0) As String
    AllRegions = GetAllRegions
    xSplit = Split(AllRegions, vbCr & vbCr & vbCr)
    For I = LBound(xSplit) To UBound(xSplit)
        If xSplit(I) <> "" Then
            myDOM.loadXML xSplit(I)
            If myDOM.firstChild.baseName = "eval" Then
                EvalName = myDOM.getElementsByTagName("ml:id")(0).Text
                EvalValue = myDOM.getElementsByTagName("ml:real")(0).Text
                Debug.Print EvalName
                Debug.Print EvalValue
                EvalArray(UBound(EvalArray)) = EvalName & "=" & EvalValue
                ReDim Preserve EvalArray(UBound(EvalArray) + 1)
            End If
        End If
    Next I
    If UBound(EvalArray) > 0 Then
        ReDim Preserve EvalArray(UBound(EvalArray) - 1)
    End If
    GetEvals = EvalArray
End Function
```

既然有了区域值，我们就可以开始利用这些值在 MicroStation 中绘制平板及其孔图案了。

```
Sub DrawFromMathcad()
    Dim AllEvals() As String
    Dim PartWidth As Double
    Dim PartHeight As Double
```


China BDN

```
    Dim OutsideBuffer As Double
    Dim HoleDia As Double
    Dim SpacingX As Double
    Dim SpacingY As Double
    Dim QtyX As Double
    Dim QtyY As Double
    Dim FilterReturn() As String
    AllEvals = GetEvals
    FilterReturn = Filter(AllEvals, "WidthIn=")
    PartWidth = CDbl(Replace(FilterReturn(0), "WidthIn=", ""))
    FilterReturn = Filter(AllEvals, "HeightIn=")
    PartHeight = CDbl(Replace(FilterReturn(0), "HeightIn=", ""))
    FilterReturn = Filter(AllEvals, "OutsideBufferIn=")
    OutsideBuffer = CDbl(Replace(FilterReturn(0), "OutsideBufferIn=", ""))
    FilterReturn = Filter(AllEvals, "HoleDiaIn=")
    HoleDia = CDbl(Replace(FilterReturn(0), "HoleDiaIn=", ""))
    FilterReturn = Filter(AllEvals, "SpacingXIn=")
    SpacingX = CDbl(Replace(FilterReturn(0), "SpacingXIn=", ""))
    FilterReturn = Filter(AllEvals, "SpacingYIn=")
    SpacingY = CDbl(Replace(FilterReturn(0), "SpacingYIn=", ""))
    FilterReturn = Filter(AllEvals, "QtyWidth=")
    QtyX = CDbl(Replace(FilterReturn(0), "QtyWidth=", ""))
    FilterReturn = Filter(AllEvals, "QtyHeight=")
    QtyY = CDbl(Replace(FilterReturn(0), "QtyHeight=", ""))
    DrawPart PartWidth, PartHeight, OutsideBuffer, _
             HoleDia, SpacingX, SpacingY, QtyX, QtyY
End Sub
```

“过滤”数组 **AllEvals**，仅取得想要的参数。找到想要的参数后，用空串替换参数名和等号然后用标准 VBA 函数 CDbl 将余下的文本(数字值)转换为一个

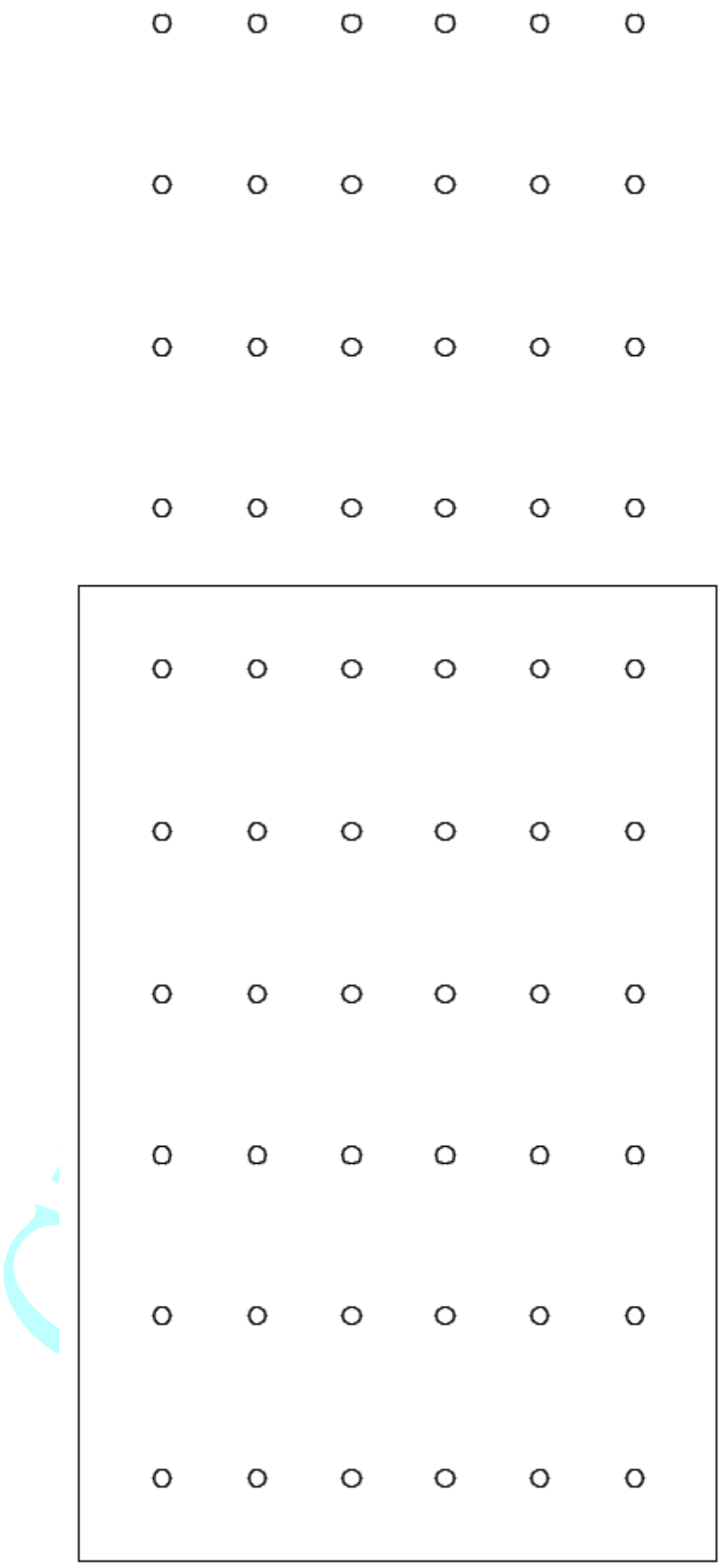
双精度数，这样就得到了参数所关联的值。每个参数值都放入其自有的变量中，然后调用过程 DrawPart 时使用到了这些变量。

```
Sub DrawPart(Width As Double, Height As Double, _
                OutBuffer As Double, HoleDia As Double, _
                SpacingX As Double, SpacingY As Double, _
                QtyWidth As Double, QtyHeight As Double)
    Dim myLine As LineElement
    Dim myCircle As EllipseElement
    Dim XPos As Double
    Dim YPos As Double
    Dim RotMatrix As Matrix3d
    Set myLine = CreateLineElement2(Nothing, _
                Point3dFromXY(0, 0), Point3dFromXY(Width, 0))
    ActiveModelReference.AddElement myLine
    Set myLine = CreateLineElement2(Nothing, _
                Point3dFromXY(Width, 0), Point3dFromXY(Width, Height))
    ActiveModelReference.AddElement myLine
    Set myLine = CreateLineElement2(Nothing, _
                Point3dFromXY(Width, Height), Point3dFromXY(0, Height))
    ActiveModelReference.AddElement myLine
    Set myLine = CreateLineElement2(Nothing, _
                Point3dFromXY(0, Height), Point3dFromXY(0, 0))
    ActiveModelReference.AddElement myLine
    XPos = OutBuffer + HoleDia / 2
    For X = 1 To QtyWidth
        YPos = OutBuffer + HoleDia / 2
            For Y = 1 To QtyHeight
                '*** Draw the Circle ***
                Set myCircle = CreateEllipseElement2(Nothing, _
                                Point3dFromXY(XPos, YPos), HoleDia / 2, _
```

```
                                HoleDia / 2, RotMatrix)
                        ActiveModelReference.AddElement myCircle
                        YPos = YPos + SpacingY
                    Next Y
                XPos = XPos + SpacingX
            Next X
End Sub
```

代码就绪了，Mathcad 工作表打开了，设计准则也输入了。现在唯一要做的事就是运行过程 DrawFromMathcad 并查看在 Mathcad 中输入的计算生成了什么样的结果。

China BDN



运行完 DrawFromMathcad 并在 MicroStation 中查看结果能发现有些不对的地方。水平间距好像正确但垂直间距不正确。

仔细查看 Mathcad 中的公式会发现，ActualSpacing 函数使用了我们定义的 Mathcad 变量 **QtyWidth**。由于 **QtyWidth** 和 **QtyHeight** 变量可能会有不同的值，所以，需要给 ActualSpacing 函数多增加一个参数并在计算 SpacingXIn 和 SpacingYIn 值时提供值。

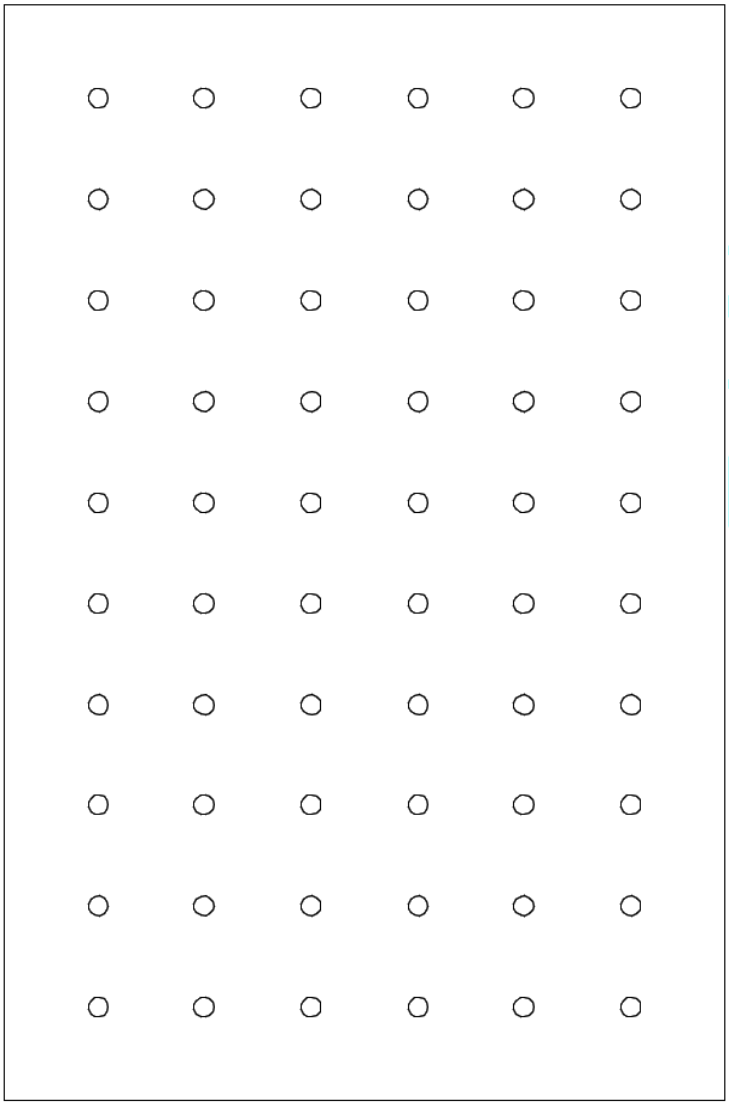
$$\text{ActualSpacing}(\text{Overall},\text{Qty}) := \frac{\text{Overall} - \text{OutsideBuffer}\cdot 2 - \text{HoleDia}}{\text{Qty} - 1}$$

$$\text{SpacingXIn} := \text{ActualSpacing}(\text{Width},\text{QtyWidth})$$

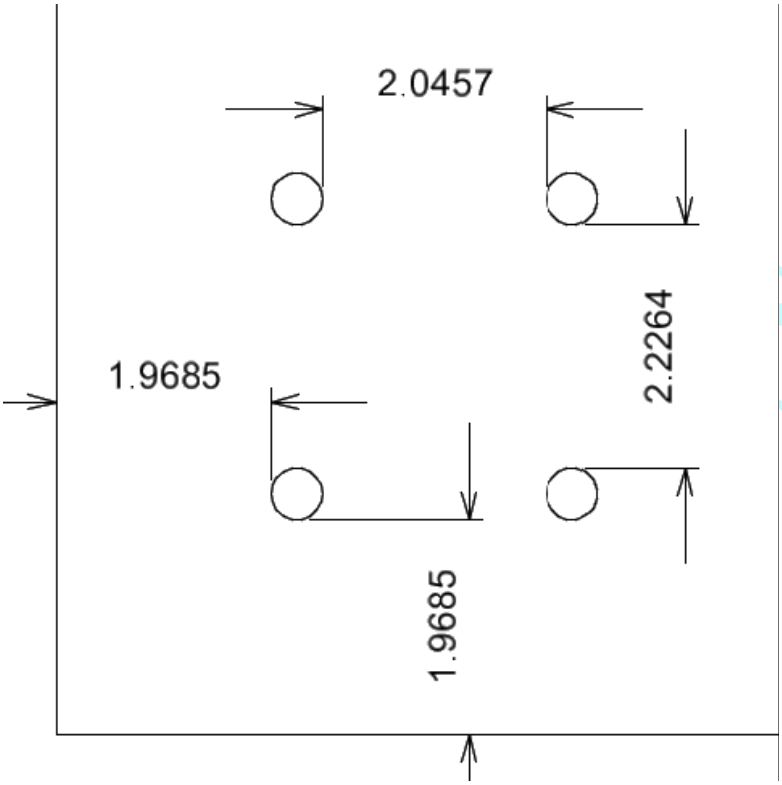
$$\text{SpacingYIn} := \text{ActualSpacing}(\text{Height},\text{QtyHeight})$$

既然在 ActualSpacing 函数中增加了一个“Qty”参数，就能在 SpacingXIn 和 SpacingYIn 公式中提供该参数，Mathcad 工作表中的值自动更新。我们可以在 MicroStation 中删除已绘制的元素并重新运行 DrawFromMathcad 过程。

China



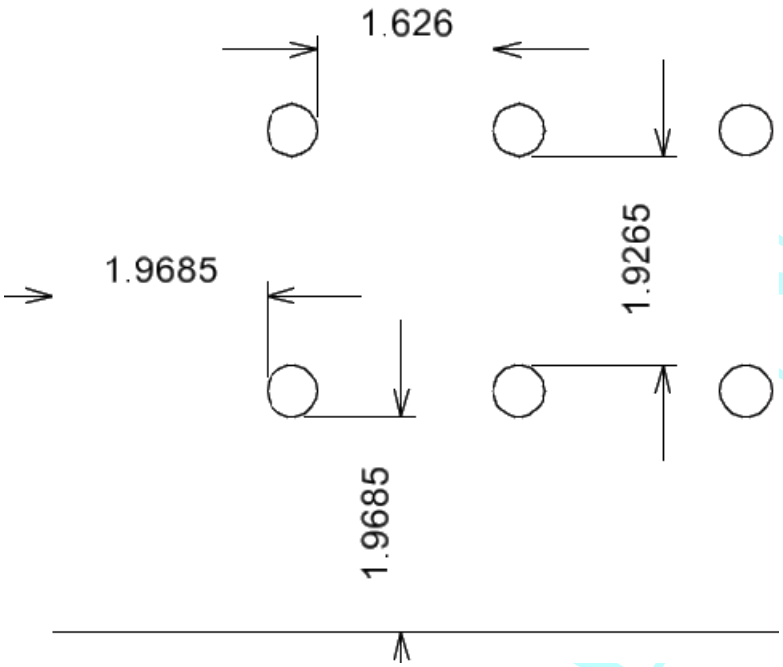
现在，在 MicroStation 中看起来孔的间距是正确的。让我们增加几个标注以验证绘制的平板满足在 Mathcad 中建立的设计准则。



缓冲区看起来是对的，但孔间距没有满足准则。查看 Mathcad 中的公式和函数表明，CalcualteQuantity 函数存在一个错误，它需要计算孔的数量而不是间距的数量。所以，要给公式加 1 以得到孔的个数而不是间距的个数。

$$\text{CalculateQuantity}(\text{Overall}) := \frac{\text{Overall} - \text{OutsideBuffer}\cdot 2 - \text{HoleDia}}{\text{HoleDia} + \text{MaxSpacing}} + 1$$

修复完 Mathcad 中的公式后，从 MicroStation 的 VBA 环境中运行宏，平板被重新绘制。给平板添加几个标注来显示其结果。

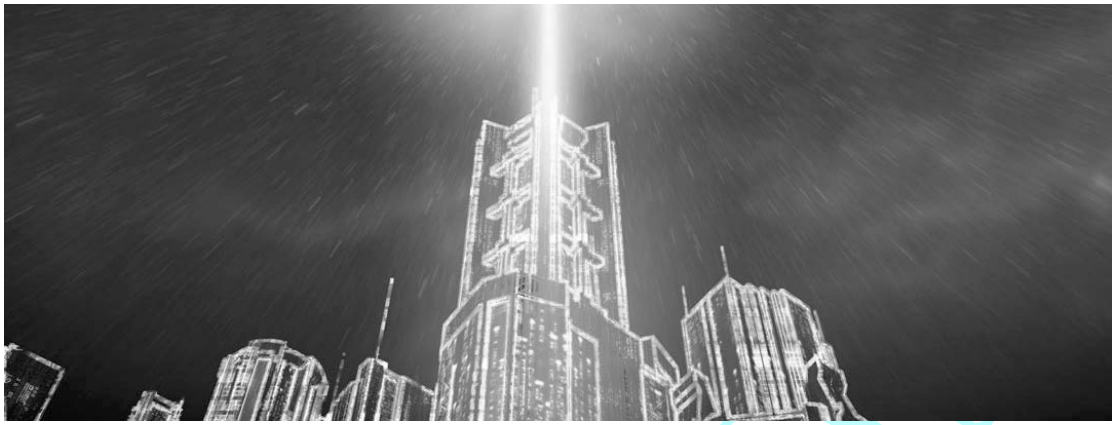


现在，孔间距满足了我们的准则“MaxSpacing=50mm”。在 Mathcad 中改变宽度、高度、孔径和外围缓冲并重新运行宏应该生成类似的结果 — 在 MicroStation 中绘制出了一个孔距均匀(分别在 X 和 Y 方向上)且间距在 MaxSpacing 范围内的平板。

本章回顾

有人曾经说过：“给我一根杠杆，我将撬动整个世界。”研究不同类型杠杆的作用后知道，用一根杠杆可以移动那些不用杠杆就无法移动的物体。Mathcad 和其它应用程序能让我们通过 ActiveX 自动化与其通讯，ActiveX 自动化在这里就像杠杆一样。ActiveX 能让我们只需花一点努力或者根本不需要费力就能完成以前易错且耗时的任务。

本章决不是关于应用 Mathcad 或其 API 的完整参考资料，这个主题需要一整本书来讨论。此处的目的是介绍和演示从 Mathcad 中取得工程数据和计算是多么的容易，并在我们的设计工作中在 MicroStation 内使用它们。



37. 从外部程序存取 MicroStation 数据

我们已经能在 MicroStation 的 VBA 环境中控制 MicroStation，也能在 MicroStation 的 VBA 环境中控制 Microsoft Excel。显而易见，从 Excel 的 VBA 环境中也能控制 MicroStation。是的，我们能。

本章包括：

- ActiveX/COM 基础
- 引用、前期绑定和后期绑定
- GetObject、SetObject 和“New”
- “WithEvents”能干什么？
- 什么时候从 Excel 中运行宏，什么时候从 MicroStation 中运行宏
- 从 Excel 中控制 MicroStation

ActiveX/COM 基础

“ActiveX 自动化”和“COM”有何共同点？从一个“公用”的观点来看，它们在各个方面都有共同点，因为它们经常指的是同样的东西。当我们谈论 ActiveX 自动化和/或 COM 时，其实我们是在谈论一种机制，这种机制能让一个应用程序与另一个应用程序“对话”。我们将讨论一个应用“驱动”另一个应用或者从另一个应用“派生”信息。

多年以来，应用生产厂家一直努力为其客户提出定制其软件的方法，人们发明了大量方法来实现定制。动态数据交换(DDE)是一种让一个应用向其它应用发送消息的方法；另一些软件开发者提出了用软件模拟用户交互的聪明方法，包括

模拟鼠标移动和点击应用程序按钮，也包括模拟在其它应用中的键盘输入；还有人发明了脚本机制，该机制会读取一个文件并执行文件中的指令。

ActiveX 自动化为我们提供了一种手段，通过使用 Application 对象模型以编程方式来控制应用。我们自顶向下查看对象模型直到某个对象为止，该对象公开了我们想要的属性、方法和事件。

Application 对象具有对象名和类 ID，哪一种最容易使用呢？让我们看一下 MicroStationDGN 的应用对象，我们愿意使用“MicroStationDGN.Application”还是“{121c4649-9bb6-11d4-81ce-0050049e89cb}”呢？很明显，我们愿意用对象名来引用类而不是用它的 ID(标识符)。

“MicroStationDGN.Application”、“Excel.Application”和“Word.Application”是一些我们可能知道的类名的例子。这些顶级对象作为一个起始点，能让我们向下挖掘出文档、工作表、段落等类似的对象。

有些应用虽然没有实现或获许使用 Microsoft VBA 环境，但它们公开了 COM 接口。所以，一个应用没有嵌入 VBA 并不意味着我们无法与其通讯，该应用能通过 COM(ActiveX 自动化)被完全打开来定制。这样的一个例子就是 Internet Explorer(互联网浏览器 IE)。

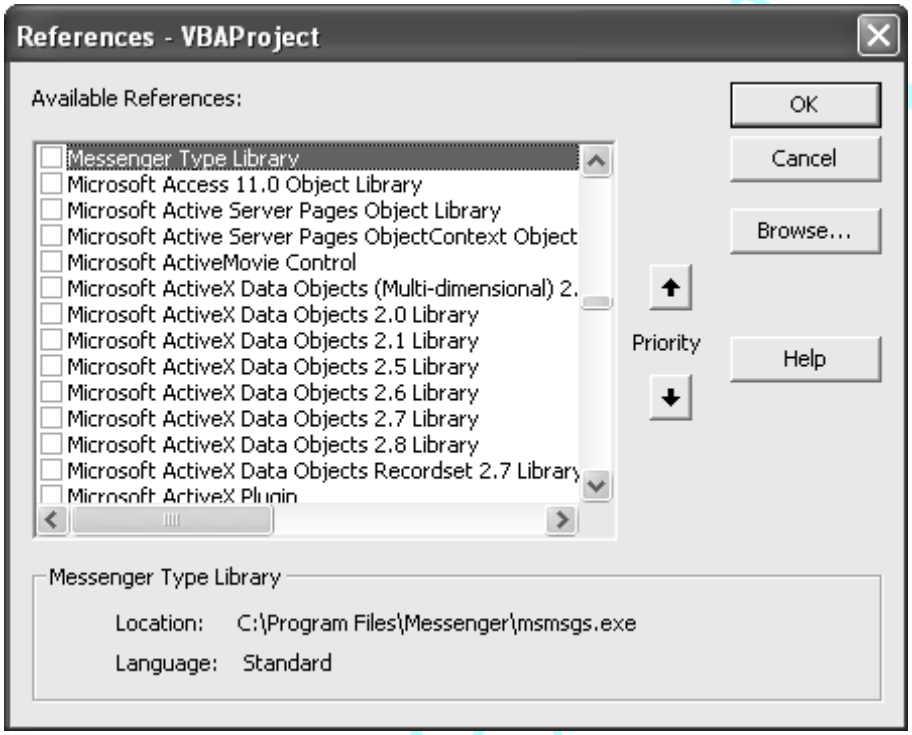
```
Sub RunInternetExplorer()
    Dim myInet As Object
    Set myInet = CreateObject("InternetExplorer.Application")
    myInet.Visible = True
    myInet.Navigate2 "http://www.bentley.com"
    MsgBox "At Bentley's Website."
    myInet.Navigate2 "http://www.microsoft.com"
    MsgBox "At Microsoft's Website."
End Sub
```

上例通过建立新的 Internet Explorer 实例来“连接”它，并使其可见、在其中显示两个网站。Internet Explorer 具有一个 COM 接口，所以可在 MicroStation 的 VBA 环境中控制它。

引用、前期绑定和后期绑定

本书至今已经添加过许多引用。为什么要这样做？可以将引用添加给什么？

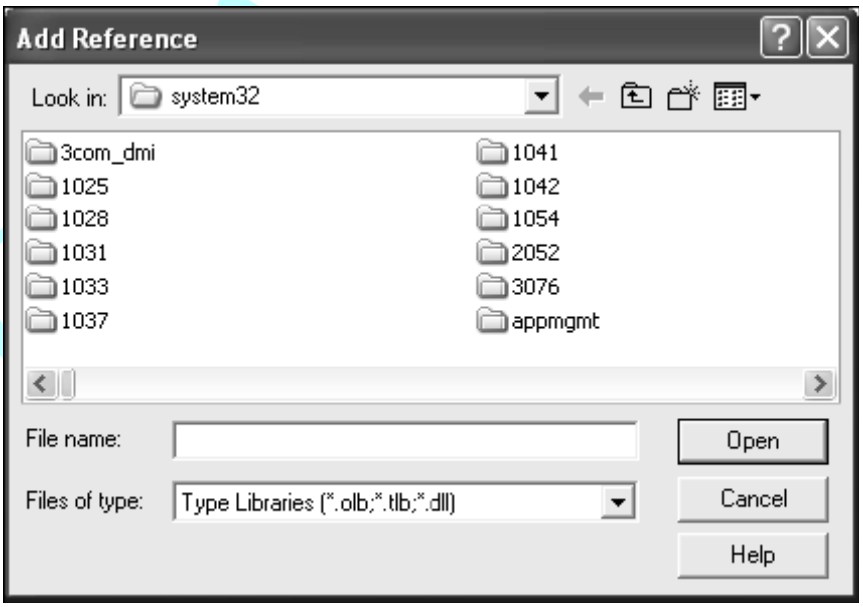
在 VBA 中，选菜单 **Tools > References(工具 > 引用)**。



以前见过这个对话框，让我们多谈论点实际看到的是什么。

当查看引用对话框时，我们实际上是在查看能添加到当前工程中的引用。这些引用存储在文件中，通常具有三种不同的文件扩展名之一：.olb、.tlb 和.dll。

如果点击 Browse(浏览)按钮，能看到 Add Reference(添加引用)对话框。



假如需要添加的引用没有出现在引用对话框中，就能在添加引用对话框中浏览到它并添加之。

类型库能为我们做两件事情：其一是提供对象模型 — 带有属性、方法和事件的对象结构。借此引入了智能感应特性 — 编写代码时出现的列表和提示文本框。另一件事是提供对象的简单存取并能执行前期绑定。

前期绑定？后期绑定？绑定就是取得变量并设置其为对象的处理过程。当谈及“前期绑定”时，意思是声明一个变量为特定类型的对象。绑定为前期的变量性能优良，在开发代码时能实现变量的智能感应以及类型检查。

让我们回顾一小段前面编写的代码：

```
Sub RunInternetExplorer()
    Dim myInet As Object
    Set myInet = CreateObject("InternetExplorer.Application")
    myInet.Visible = True
    myInet.Navigate2 "http://www.bentley.com"
    MsgBox "At Bentley's Website."
    myInet.Navigate2 http://www.microsoft.com
    MsgBox "At Microsoft's Website."
End Sub
```

这里使用了前期绑定还是后期绑定？后期绑定。为什么？因为我们声明变量 **myInet** 为一个对象。这是一种通用对象，它可以是 Internet Explorer 应用、一个线元素或一个数据库连接。直到如下代码设置之前都不知道该变量是什么类型的对象。

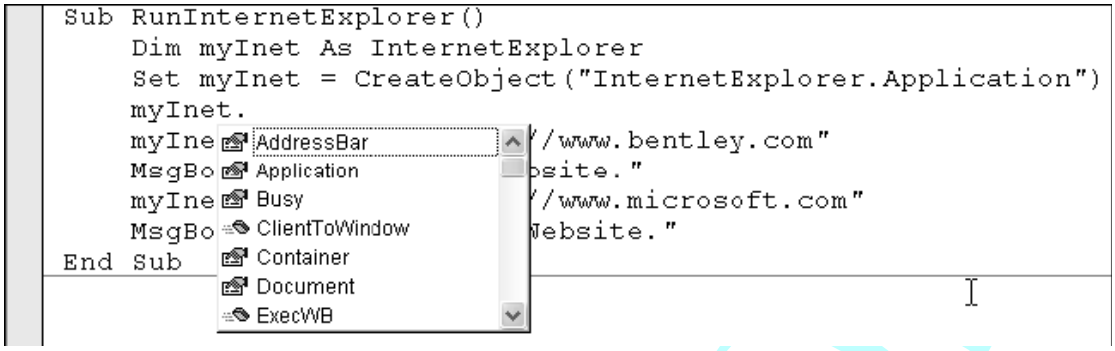
```
Set myInet = CreateObject("InternetExplorer.Application")
```

这行代码运行后，变量 **myInet** 才知道其是何种类型的对象。在使用后期绑定时智能感知未被激活，因为变量不知道其是何种类型的对象。

```
Sub RunInternetExplorer()
    Dim myInet As Object
    Set myInet = CreateObject("InternetExplorer.Application")
    myInet.
```

当在上图中键入“myInet.”(在变量名后有一个点号)时，并没有显示出智能感知。然而，如果把 **myInet** 声明为“InternetExplorer”，并且也添加了正确的

引用“Microsoft Internet Controls”，智能感知就会起作用。



现在，键入“myInet.”能得到有助于我们的智能感知，因为已经把变量 **myInet** 声明为了一种特定类型的对象并该对象被正确引用。

用后期绑定编写的代码能够运行，但用前期绑定编写的代码能工作得更好，开发速度更快，调试更容易。

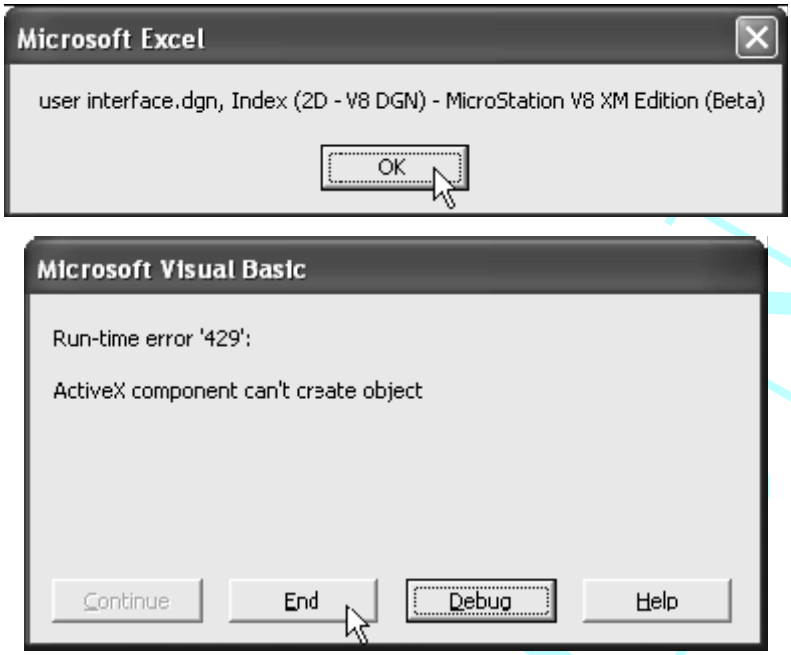
GetObject、SetObject 和 New

如果要用 ActiveX 自动化来操作一个应用程序，必须在做任何事之前“连接”该应用。有三种连接应用的方式。

下面的几个例子将在 Microsoft Excel 的 VBA 环境中开发。我们还将添加引用“Bentley MicroStation DGN #.# Object Library”。

```
Sub ConnectToMicroStationA()
    Dim myMSAppCon As MicroStationDGN.ApplicationObjectConnector
    Dim myMSApp As MicroStationDGN.Application
    Set myMSAppCon = GetObject(, _
                        "MicroStationDGN.ApplicationObjectConnector")
    Set myMSApp = myMSAppCon.Application
    MsgBox myMSApp.Caption
    Set myMSApp = Nothing
    Set myMSAppCon = Nothing
End Sub
```

这段代码运行后可能会看到如下两种情况：

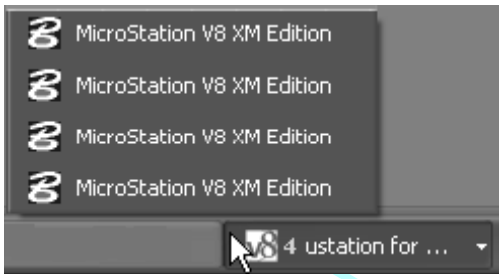


在第一个消息框中看到了 MicroStation 的标题，第二个消息框告诉我们“ActiveX 组件不能创建对象”。如果看到的是第一个消息框，就知道代码正常工作了；如果看到的是第二个消息框，就知道代码未正常工作。为什么不能正常工作呢？因为 GetObject 用于取得一个现有的对象实例。在这个例子中，MicroStation 必须在使用 GetObject 前运行。

下一个例子使用了 CreateObject。

```
Sub ConnectToMicroStationB()
    Dim myMSAppCon As MicroStationDGN.ApplicationObjectConnector
    Dim myMSApp As MicroStationDGN.Application
    Set myMSAppCon = _
            CreateObject("MicroStationDGN.ApplicationObjectConnector")
    Set myMSApp = myMSAppCon.Application
    myMSApp.Visible = True
    MsgBox myMSApp.Caption
    Set myMSApp = Nothing
    Set myMSAppCon = Nothing
End Sub
```

现在，不是取得一个已经存在的 MicroStation 实例，而是用 CreateObject 创建一个新实例。如果多次运行这个宏，就得到多个运行中的 MicroStation 实例。



查看任务栏能看到，ConnectToMicroStationB 已经被运行了四次。

GetObject 取得应用的一个现有实例，CreateObject 创建应用的一个新实例。New 关键字也能用于建立应用的新实例，有两种使用方式。

```
Sub ConnectToMicroStationC()
    Dim myMSAppCon As New _
                        MicroStationDGN.ApplicationObjectConnector
    Dim myMSApp As MicroStationDGN.Application
    Set myMSApp = myMSAppCon.Application
    myMSApp.Visible = True
    MsgBox myMSApp.Caption
    Set myMSApp = Nothing
    Set myMSAppCon = Nothing
End Sub
```

在声明一个对象类型的变量时可以使用 New 关键字。ConnectToMicroStationC 启动一个新的 MicroStation 应用。关键字“New”也能如下所示来使用：

```
Sub ConnectToMicroStationD()
    Dim myMSAppCon As MicroStationDGN.ApplicationObjectConnector
    Dim myMSApp As MicroStationDGN.Application
    Set myMSAppCon = New MicroStationDGN.ApplicationObjectConnector
    Set myMSApp = myMSAppCon.Application
    myMSApp.Visible = True
    MsgBox myMSApp.Caption
    Set myMSApp = Nothing
    Set myMSAppCon = Nothing
End Sub
```

Qinjia BDN

这个例子先声明变量 **myMSAppCon**，然后用代码建立了一个新的 MicroStation 实例。

```
Set myMSAppCon = New MicroStationDGN.ApplicationObjectConnector
```

我们已经使用过取得现有 MicroStation 实例的代码，也使用过创建新的 MicroStation 实例的代码，这些代码都能正常运行。现在让我们讨论何时使用这些机制。

什么时候使用 GetObject、CreateObject 和 New

先来讨论 CreateObject 和 New。有三个理由想要建立新的 MicroStation 实例：

1、需要操作 MicroStation，如果存在一个正在运行的 MicroStation 最好，但当前没有。在这种情况下，我们可能使用 GetObject 但得到一个错误，所以使用 CreateObject 或 New(如 ConnectToMicroStationD 中那样使用)来建立新实例从而才能操作 MicroStation；

2、MicroStation 的一个实例正在运行，但我们不想用代码打断其会话过程，因此要建立一个新的 MicroStation 实例；

3、有多个 MicroStation 实例正在运行，如果用 GetObject 就不能确定“连接”的是哪个 MicroStation 实例，所以使用 CreateObject 或 New 建立新实例并“连接”之。

第二和第三种情况已经解释清楚了。第一种情况应当用一些代码来演示：

```
Sub ConnectToMicroStationE()
    Dim myMSAppCon As MicroStationDGN.ApplicationObjectConnector
    Dim myMSApp As MicroStationDGN.Application
    'Attach to existing or create new if no existing MicroStation
    On Error Resume Next
    Set myMSAppCon = GetObject(, _
                        "MicroStationDGN.ApplicationObjectConnector")
    If Err.Number = 429 Then
        Set myMSAppCon = _
            CreateObject("MicroStationDGN.ApplicationObjectConnector")
```

```
        Err.Clear
    End If
    On Error GoTo 0
    Set myMSApp = myMSAppCon.Application
    myMSApp.Visible = True
    MsgBox myMSApp.Caption
    Set myMSApp = Nothing
    Set myMSAppCon = Nothing
End Sub
```

在这个例子中，我们很清楚 MicroStation 有可能没有运行。使用语句“On Error Resume Next”后，假如确实遇到错误代码也将继续执行。在尝试了 GetObject 之后立即查看 Err.Number，看一看是否遇到了错误号 429(这是使用 GetObject 但所取对象不存在时的错误号)。上例中如果看到错误 429 就使用 CreateObject 并清除错误。然后使用“On Error GoTo 0”恢复 VBA 中的错误处理程序。

“WithEvents”能干什么？

当要与其它应用程序通讯并在发生某些事件时能得到提示，就要使用关键字 WithEvents。

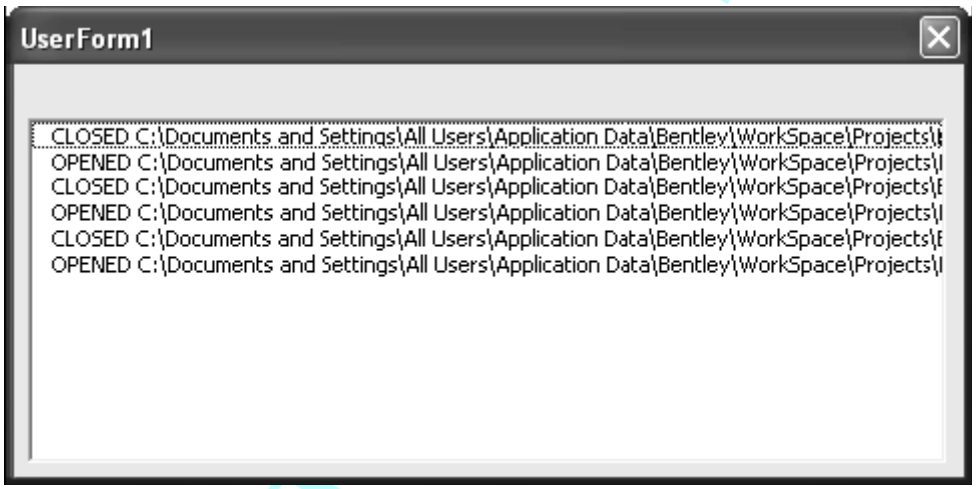
让我们在 Microsoft Excel 的 VBA 环境中建立一个新的用户窗体，还要在该窗体中添加一个列表框。这些做完后输入如下代码：

```
Dim myMSAppCon As MicroStationDGN.ApplicationObjectConnector
Dim WithEvents myMSApp As MicroStationDGN.Application
Private Sub UserForm_Initialize()
    Set myMSAppCon = GetObject(, _
                    "MicroStationDGN.ApplicationObjectConnector")
    Set myMSApp = myMSAppCon.Application
End Sub
Private Sub myMSApp_OnDesignFileClosed(ByVal _
                                DesignFileName As String)
```

```
    ListBox1.AddItem "CLOSED " & DesignFileName
End Sub
Private Sub myMSApp_OnDesignFileOpened(ByVal _
                                    DesignFileName As String)
    ListBox1.AddItem "OPENED " & DesignFileName
End Sub
```

在声明变量 **myMSApp** 时使用了关键字 WithEvents，这样做就把变量 **myMSApp** 添加到了窗体代码区的对象复合框中。当在窗体代码区的对象复合框中选择 myMSApp 时，与该变量所代表的对象相关联的事件显示在过程复合框中。

该窗体运行后，它连接到了一个现有的 MicroStation 实例中，每当关闭或打开一个文件时，相应事件中的代码被执行。



WithEvents 能让我们捕获并响应外部应用的事件。在这个例子中，我们将 MicroStation 中打开和关闭的文件名添加到了在 Excel 的 VBA 环境中建立的列表框中。在得到 MicroStation 中发生事件的通报后可以做各种事情。

这里应当说明的是，同样的 WithEvents 关键字也能用于 Microsoft Excel.Application 对象的引用。我们能在 MicroStation 的 VBA 环境中捕获发生在 Excel 中的事件。

从 Excel 还是从 MicroStation 中运行宏？

何时从 Excel 中运行宏？何时从 MicroStation 中运行宏？

如果我们能从 Excel 中控制 MicroStation，也能从 MicroStation 中控制

Excel，什么时候在 MicroStation 中编写代码，什么时候在 Excel 中编写代码呢？这是一个好问题，有几个因素应当加以考虑。这些因素未按重要性排序，因为某项因素的重要性会根据情况的不同而不同。为了方便讨论，我们使用 MicroStation 和 Microsoft Excel 作为两个要开发的应用。

- 这台计算机安装有 MicroStation 吗？如果在 MicroStation 的 VBA 环境中开发，代码就要在安装有 MicroStation 的计算机上运行。如果在 Excel 中开发，代码是保存在 Excel 中，有可能本地不需要安装 MicroStation；
- 通过 COM 管道(pipeline)通讯比较费时，虽然不是很多时间，但的确需要花费一些时间。如果在 Excel 中有大量数据且需要对 Excel 中的数据做大量工作，那么应用程序在 Excel 中运行将花费较少的时间。然而，如果绝大多数处理都是在 MicroStation 中完成，那么代码就应该在 MicroStation 的 VBA 环境中开发；
- 当前使用的是 GetObject 还是 CreateObject？Microsoft Excel 被许多开发者使用，有时 Excel 的实例在后台(虽然应用不可见)运行。我们并不知道 Excel 是否在后台运行，除非在 Windows 的任务管理器的进程选项卡中查找“EXECEL.EXE”。数据经常是存储在 Excel 中而应用是从 MicroStation 中运行。我们的想法是，用户在 Excel 中打开他/她想要的文件，程序在 MicroStation VBA 环境中使用 GetObject 来读取数据。然而，假如有一个 Excel 实例在后台运行，它(正在后台运行的实例)可能就成为 GetObject 返回的 Excel.Application 对象的实例，我们的程序要么崩溃要么使用了并非用户所期望的数据。另外，在使用 CreateObject 时应当总是设置 Visible 属性为 True，这样就不会遇到在后台运行的应用了。

在开发一个应用程序前，为开发工作选择最佳工具应该是要考虑的第一要素。我们已经列出了几个值得考虑的因素，对于特定的项目或任务来说还会有其它要考虑的因素。

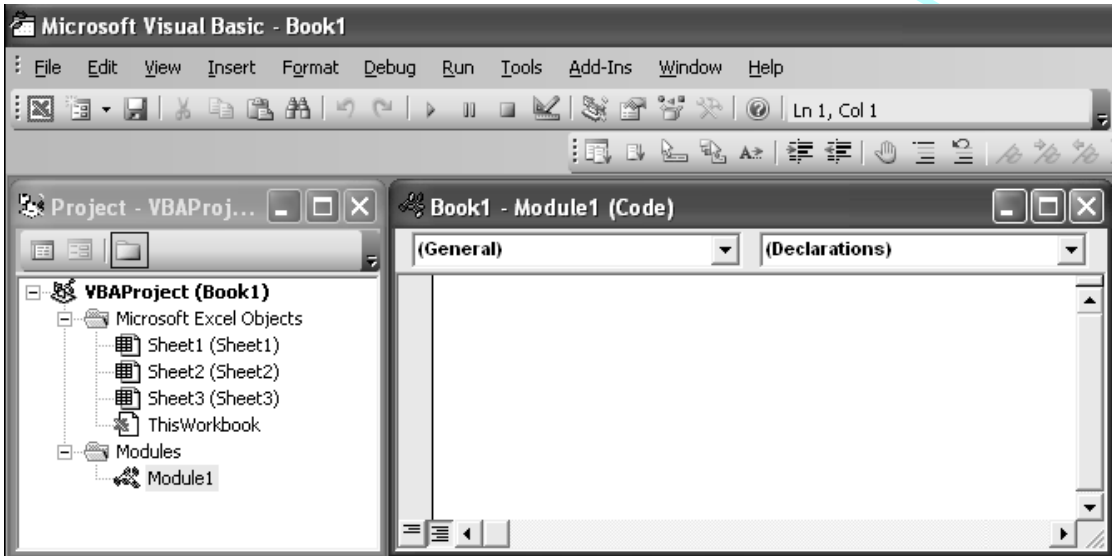
从 Excel 中控制 MicroStation

MicroStation VBA 工程保存在.mvba 文件中，而 Excel 略有不同。在 Excel 中编写 VBA 代码时，代码存储在 Microsoft Excel 的工作簿(.xls 文件)中。Excel 的 VBA 程序也能存储在 Excel 的附加程序(.xla 文件)中，本章稍后部分将讨论附

加程序。

让我们建立一个新的 Excel 工作簿。然后通过点击 **Tools > Macro > Visual Basic Editor(工具 > 宏 > Visual Basic 编辑器)**来进入 Excel 的 VBA 环境。

接下来是插入一个新模块。



Excel 中的 VBA 环境看起来与 MicroStation 中的 VBA 环境非常类似。然而，工程窗口有一处不同，“Microsoft Excel 对象”可为我们所用。它们是什么？这些 Excel 对象能让我们直接在属于特定工作表或工作簿的事件中编写代码。

这样，我们就有了一个插入好的代码模块。我们要编写与 MicroStation 通讯的代码。下一步是什么？应当添加引用“Bentley MicroStation DGN #.# Object Library”。

引用就位后，就该编写与 MicroStation 通讯的一些简单代码了。

```
Sub XLMSA()
    Dim myMSAppCon As MicroStationDGN.ApplicationObjectConnector
    Dim myMSApp As MicroStationDGN.Application
    Dim myLevel As Level
    Dim CurRow As Long
    Set myMSAppCon = GetObject(, _
                    "MicroStationDGN.ApplicationObjectConnector")
    Set myMSApp = myMSAppCon.Application
    CurRow = 2
```

```
    For Each myLevel In myMSApp.ActiveDesignFile.Levels
        Sheet1.Cells(CurRow, 1) = myLevel.Number
        Sheet1.Cells(CurRow, 2) = myLevel.Name
        CurRow = CurRow + 1
    Next
End Sub
```

在 Excel 中进行开发时，有些对象拿来就能使用，因为我们就在 Excel 中。不需要声明它们或实例化它们，直接使用就行了。在上例中，“Sheet1”就是这些对象变量中的一个。我们声明了层变量和当前行变量，但不需要声明“Sheet1”变量。

右图是运行“XLMSA”后的结果：

	A	B	C
1			
2	2	ExistingTerrain	
3	6	Building SiteTerrain	
4	7	ExistingTerrainMesh	
5	1	Frame	
6	8	BuildingSiteMesh	
7	4	Links	
8	0	Default	

层号和层名被放入 Excel 的“Sheet1”中。

现在，不是从 MicroStation 的 ActiveDesignFile 中读取层，而是根据 Excel 中的内容建立层，先要在 MicroStation 中建立一个新的设计文件。

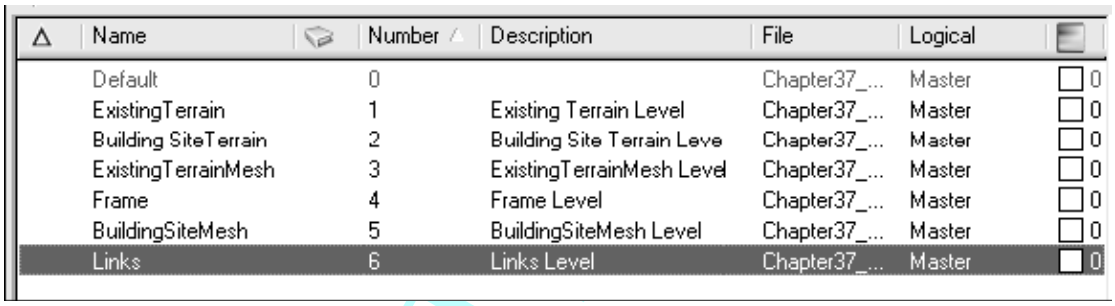
	A	B	C
1	#	Name	Description
2	1	ExistingTerrain	Existing Terrain Level
3	2	Building SiteTerrain	Building Site Terrain Level
4	3	ExistingTerrainMesh	ExistingTerrainMesh Level
5	4	Frame	Frame Level
6	5	BuildingSiteMesh	BuildingSiteMesh Level
7	6	Links	Links Level

用以上 Excel 中的数据运行 XLMSB，在新的.dgn 文件中就能得到建立的层。下面是代码：

```
Sub XLMSB()
    Dim myMSAppCon As MicroStationDGN.ApplicationObjectConnector
    Dim myMSApp As MicroStationDGN.Application
    Dim myLevel As Level
    Dim CurRow As Long
```

```
    Set myMSAppCon = GetObject(, _
                        "MicroStationDGN.ApplicationObjectConnector")
    Set myMSApp = myMSAppCon.Application
    CurRow = 2
    While Sheet1.Cells(CurRow, 1) <> ""
        Set myLevel = myMSApp.ActiveDesignFile.AddNewLevel _
                        (Sheet1.Cells(CurRow, 2))
        myLevel.Number = Sheet1.Cells(CurRow, 1)
        myLevel.Description = Sheet1.Cells(CurRow, 3)
        CurRow = CurRow + 1
    Wend
End Sub
```

以上代码运行后，在 MicroStation 的层管理器中能看到如下结果：

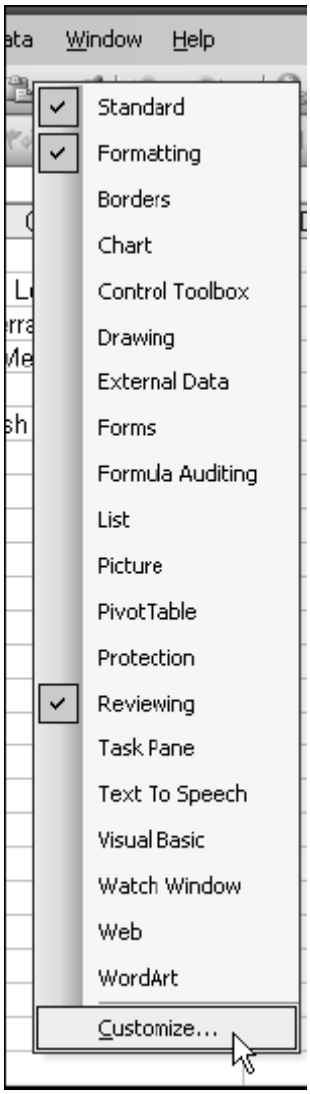


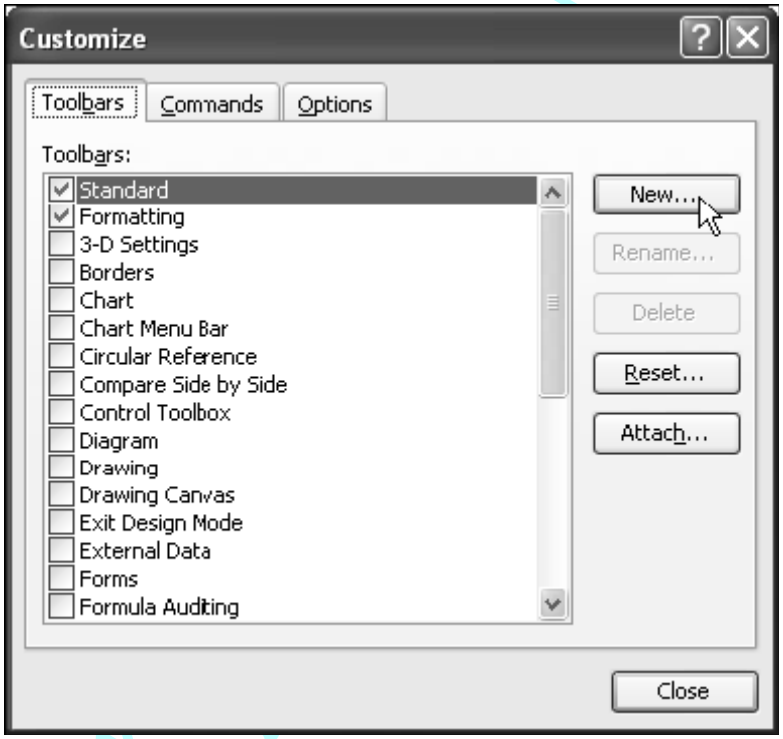
Name	Number	Description	File	Logical	
Default	0		Chapter37_...	Master	0
ExistingTerrain	1	Existing Terrain Level	Chapter37_...	Master	0
Building SiteTerrain	2	Building Site Terrain Leve	Chapter37_...	Master	0
ExistingTerrainMesh	3	ExistingTerrainMesh Level	Chapter37_...	Master	0
Frame	4	Frame Level	Chapter37_...	Master	0
BuildingSiteMesh	5	BuildingSiteMesh Level	Chapter37_...	Master	0
Links	6	Links Level	Chapter37_...	Master	0

运行 Excel 宏

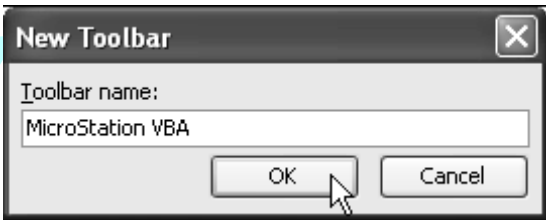
在 Excel 的 VBA 环境中开发完代码后，我们想让用户使用宏但不能修改代码。Excel 的 VBA 环境(就想 MicroStation 的 VBA 环境一样)能让我们“锁定”工程，做法是在工程属性的 Protection(保护)选项卡中选择“Lock project for viewing(查看时锁定工程)”。但当工程被锁定或者不想让用户进入 VBA 环境来执行宏时又该怎么办呢？让我们在 Excel 中建立一个自定义的按钮。

1、在 Excel 中的任意工具栏按钮上右击，然后点击 Customize(自定义)。自定义对话框被打开；





2、点击 New(新建)按钮启动在 Microsoft Excel 中建立新工具栏的步骤；



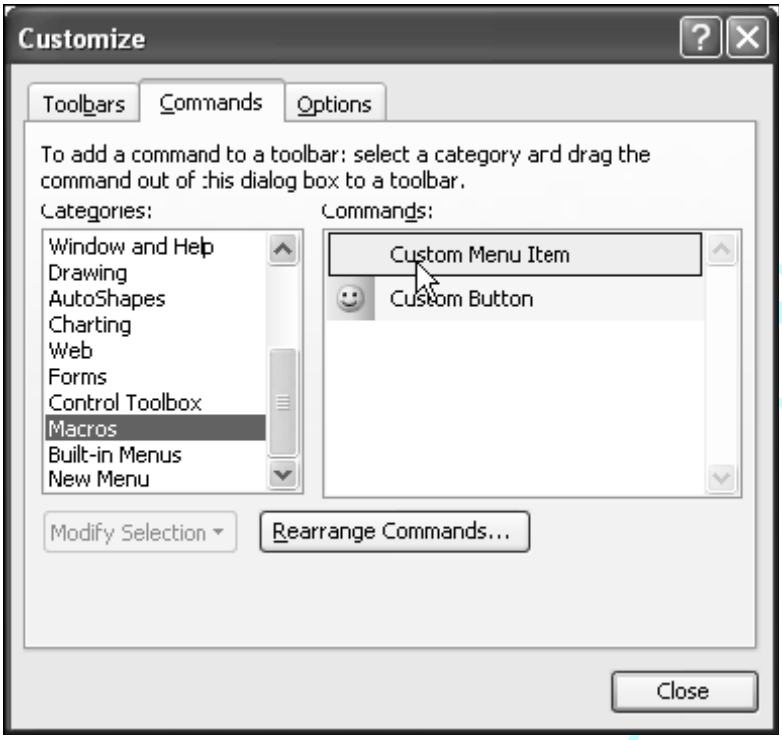
3、把新工具栏命名为“MicroStation VBA”并点击 OK(确定)；



需要提醒的是：新工具栏很小，因为其中还没有任何按钮或菜单项。

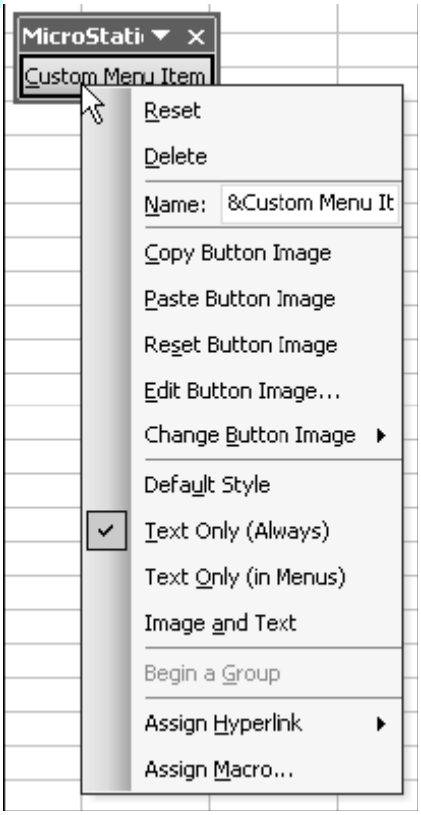
4、下一步是切换到 Command(命令)选项卡并在 Categories(类别)列表框中选择“Macros(宏)”。此时出现了两个命令 — “Custom Menu Item(自定义菜单项)”和“Custom Button(自定义按钮)”；

5、把“Custom Menu Item(自定义菜单项)”拖拽到刚建的新工具栏中；





现在很想点击 Close(关闭)按钮关闭自定义对话框，因为我们已经成功地将一个新菜单项拖拽到了新工具栏上。但能继续进行的关键是要保持自定义对话框处于打开状态；

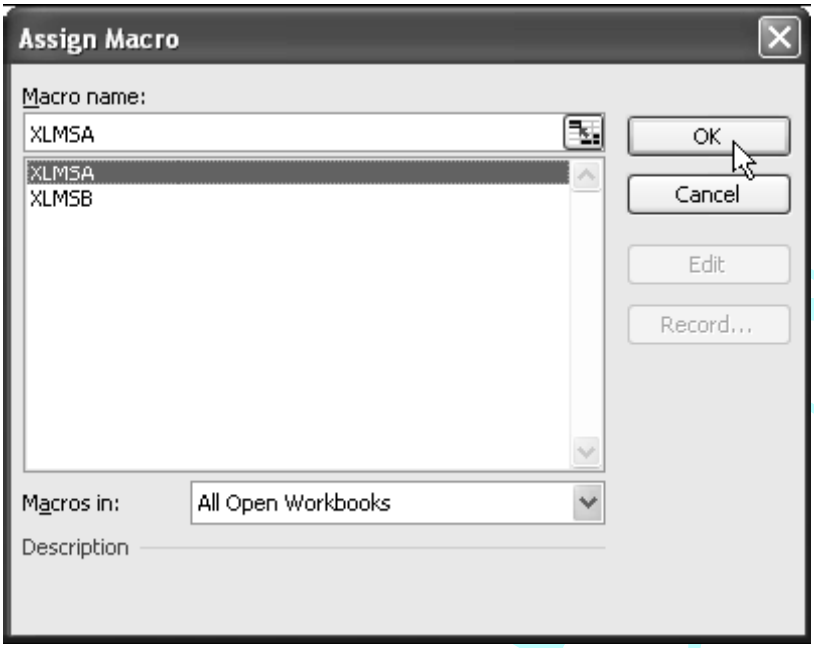


6、右击新菜单项在弹出的上下文菜单中查看；

7、Name(命名)属性指定菜单项的显示名。它目前设置为“Custom Menu Item(自定义菜单项)”，让我们修改为“&Get DesignFile Levels(取得设计文件的层)”；



8、再次在菜单项上右击来“Assign Macro(指定宏)”；



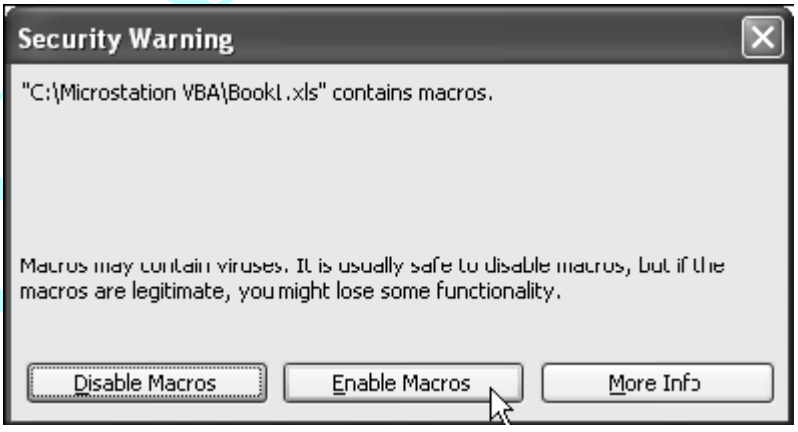
9、我们能选择一个宏并点击 OK 按钮，这就将选择的宏赋给了按钮。从现在开始，任何时候只要点击 Excel 中的这个按钮，就会运行宏 XMMSA。

既然已经重命名并指定了菜单项，关闭自定义对话框后就能运行它了；

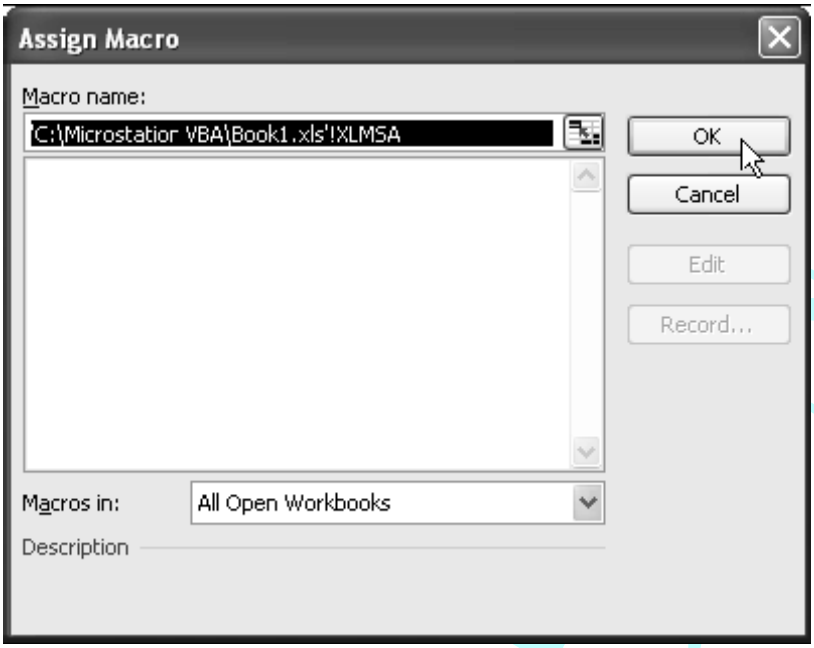
10、我们将要运行的宏在 Sheet1 的单元中赋值。在处理之前删除 Sheet1 中的数据，然后点击 Get DesignFile Levels 按钮来看一看是否一切工作正常；

11、现在我们保存这个 Excel 文件并关闭它。关闭该工作簿后再开始一个新的工作簿，然后再次点击 Get DesignFile Levels 按钮。根据您 Excel 中安全性设置的不同会发生不同的情况。

可能会看到像这样的提示：



当把一个宏指定给新的菜单项时，它是赋给了特定文件中的一个宏。即使含有宏的工作簿被关闭，菜单项仍然链接到那个文件。针对目前的情况，假如 Book1.xls 被关闭，当我们回到 Assign Macro(指定宏)的对话框时将看到如下内容：



因此，假如关闭了编写宏的那个 Excel 文件，当点击链接到那个宏的按钮时，Excel 会打开那个工作簿使宏能够运行。如果我们“Enable Macros(启用宏)”(假设提示我们这样做的话)，则宏运行后得到了层。但层在什么地方呢？即使点击按钮时我们打开了另一个工作簿，它们仍被输入到了 Book1.xls 中。让我们再看一眼代码：

```
Sub XLMSA()
    Dim myMSAppCon As MicroStationDGN.ApplicationObjectConnector
    Dim myMSApp As MicroStationDGN.Application
    Dim myLevel As Level
    Dim CurRow As Long
    Set myMSAppCon = GetObject(, _
                    "MicroStationDGN.ApplicationObjectConnector")
    Set myMSApp = myMSAppCon.Application
    CurRow = 2
    For Each myLevel In myMSApp.ActiveDesignFile.Levels
        Sheet1.Cells(CurRow, 1) = myLevel.Number
        Sheet1.Cells(CurRow, 2) = myLevel.Name
        CurRow = CurRow + 1
    Next
```

```
End Sub
```

为什么层被写入到了 Book1.xls 中而不是新的工作簿中？因为我们直接写入了“Sheet1”，而这个对象是在编写代码的那个工作簿中预定义的。

让我们复制、粘贴宏 XLMSA 并将其改名为 XLMSC。我们将做出几处改变，让层在选择菜单项后写入打开的工作簿中：

```
Sub XLMSC()
    Dim myMSAppCon As MicroStationDGN.ApplicationObjectConnector
    Dim myMSApp As MicroStationDGN.Application
    Dim myLevel As Level
    Dim CurRow As Long
    Set myMSAppCon = GetObject(, _
                        "MicroStationDGN.ApplicationObjectConnector")
    Set myMSApp = myMSAppCon.Application
    CurRow = 2
    For Each myLevel In myMSApp.ActiveDesignFile.Levels
        ActiveSheet.Cells(CurRow, 1) = myLevel.Number
        ActiveSheet.Cells(CurRow, 2) = myLevel.Name
        CurRow = CurRow + 1
    Next
End Sub
```

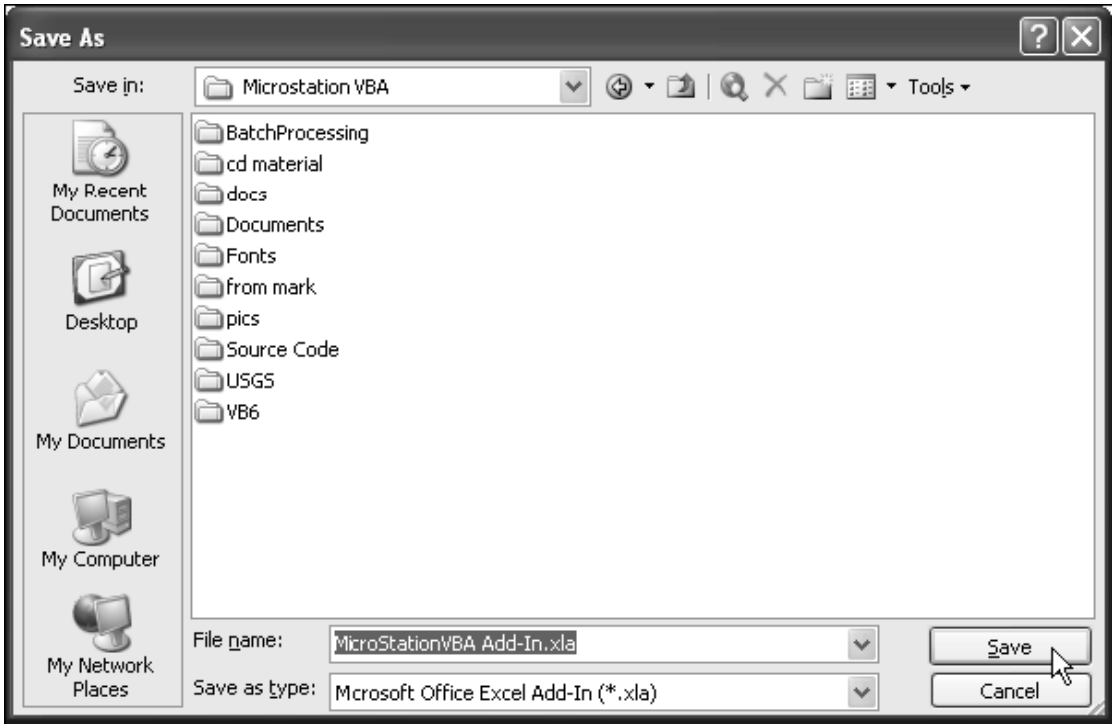
变化虽小但变化的意义却不小。现在让我们将 XLMSC 赋给选择菜单项后要运行的宏。

这样做了以后，选择菜单时工作簿 Book1.xls 仍处于打开状态，但层被放入了当前的工作表中。代码运行得比较好了，但每当有人在 Excel 中点击该按钮时就要打开宏所在的工作簿，这一点还能加以改进。

我们知道了在运行宏之前，宏所在的文件应当被打开。但当点击菜单项时我们期望代码能运行而不打开工作簿，也不出现提示启用宏的对话框。怎样才能做到这一点呢？

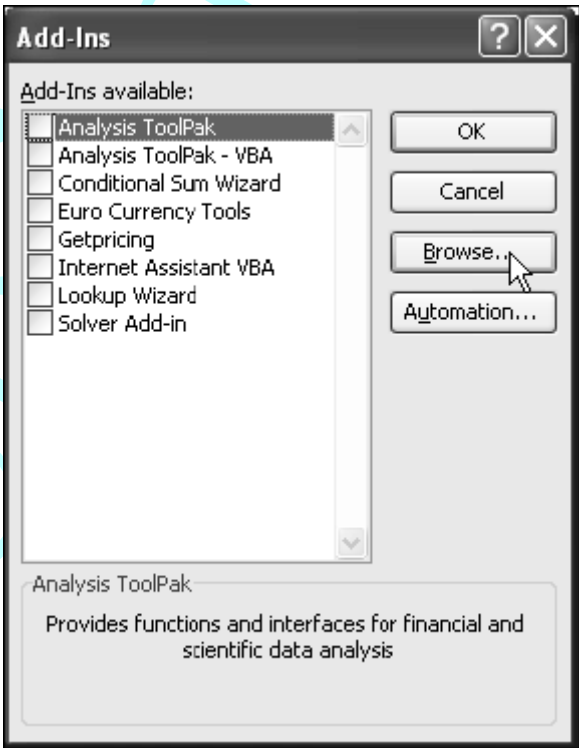
1、在 Excel 中打开 Book1.xls，然后将文件 Book1.xls 另存为 Excel 加载宏。

让我们使用文件名 MicroStationVBA Add-in.xla；

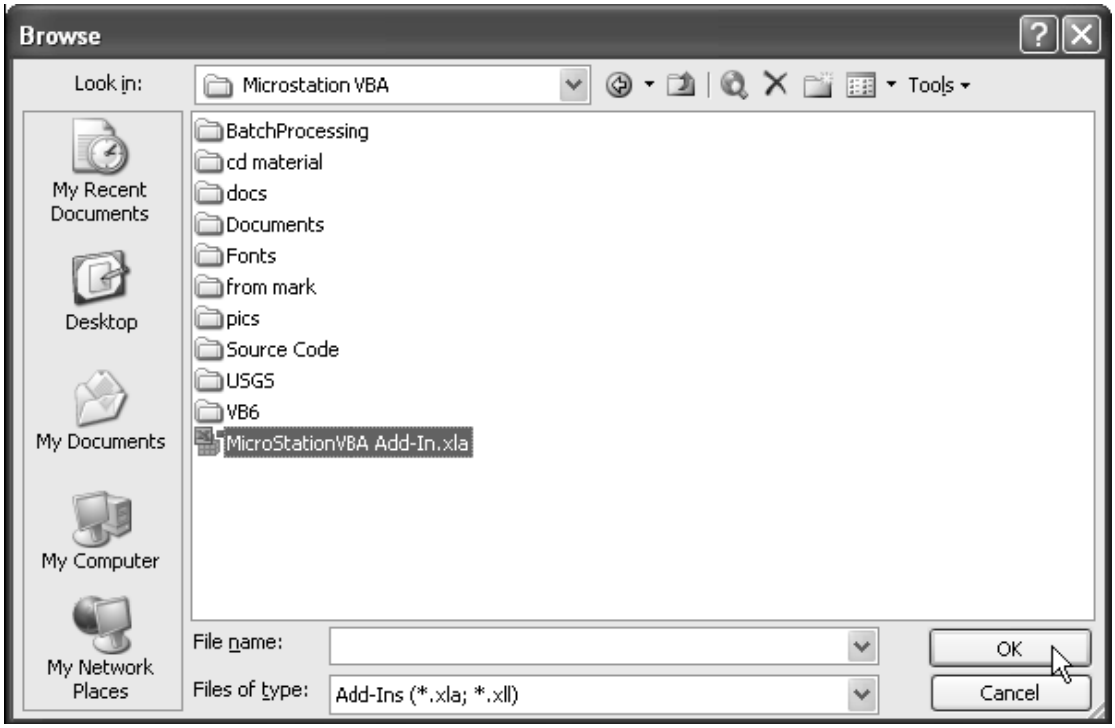


2、把一个工作簿(.xls 文件)保存为加载宏(.xla 文件)执行了一种“另存为的复制”，此时原来的.xls 文件仍处于打开状态。所以在保存完加载宏文件后可以关闭原来的 Book1.xls 文件并打开一个新工作簿；

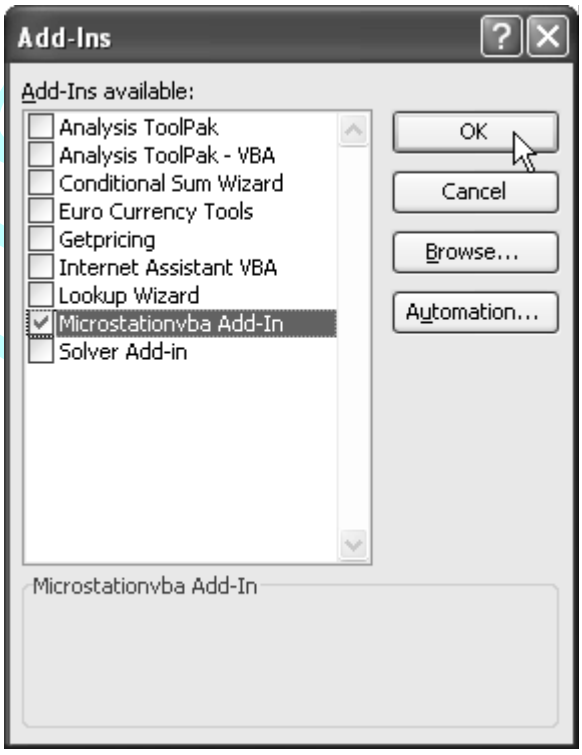
3、接下来在 Excel 的菜单中选择 **Tools > Add-Ins(工具 > 加载宏)**；



4、浏览到刚建立的新.xla 文件；



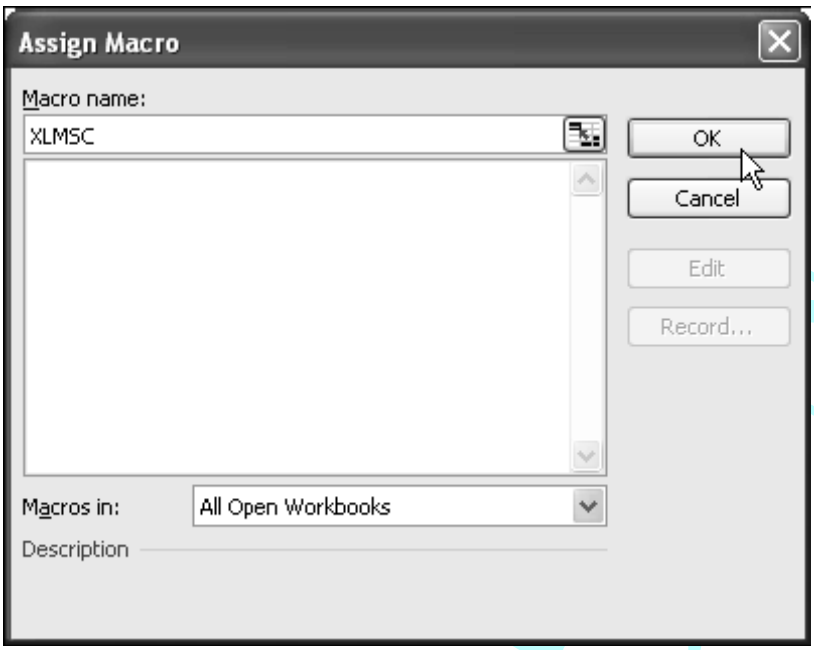
5、选择完文件后点击 OK，该文件显示在加载宏对话框中并被选中。这样就将该文件装载到内存中(我们能在 VBA 中使用文件但不是在 Excel 的工作表界面)，当点击菜单项时代码已经被装载并准备运行；



6、点击加载宏对话框中的 OK 按钮以完成加载宏的处理过程；

7、现在，添加到自定义工具栏中的菜单项仍然指向 Book1.xls，我们想让它运行刚建立的加载宏文件中的宏。右击菜单项选择自定义，再次右击菜单项选择指定宏，这样就回到指定宏对话对话框；

8、我们不想运行 Book1.xls 中的宏，而想运行加载宏中的宏。但加载宏中的宏未在可用宏列表中显示，此时在指定宏对话框中直接键入宏名并点击 OK。



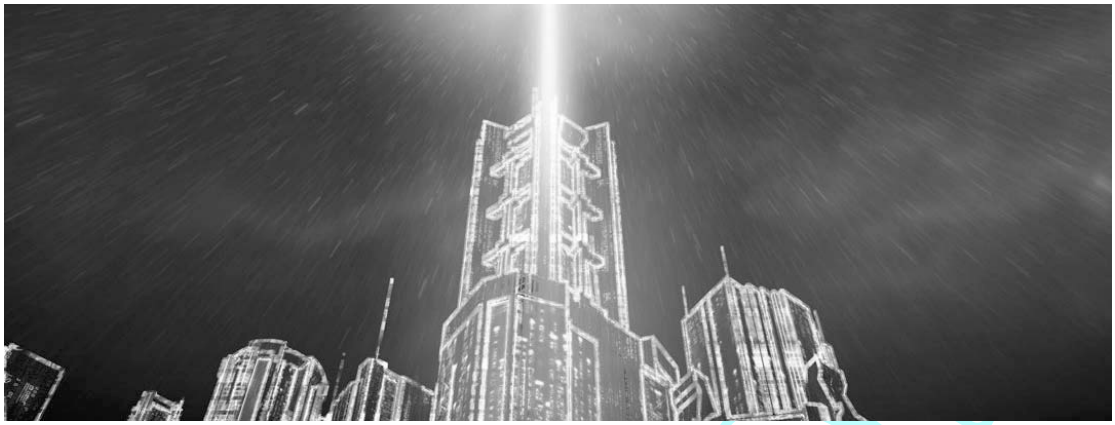
9、在指定宏对话框中点击 OK 后，在自定义对话框中点击 Close(关闭)按钮，我们可以测试菜单项了。

现在点击 Get DesignFile Levels 按钮时宏被立即执行，因为加载宏已经装载。如果从“Sheet1”变到“Sheet2”并再次点击菜单项，层就显示在“Sheet2”中。请记住，在我们的代码中将层写入到了“ActiveSheet”，所以 Excel 的当前工作表将接受层信息。

在建立菜单项时，将其命名为“&Get DesignFile Levels”。与字符(&)表明菜单项中的哪个字符具有下划线。下划线字符的意思是，我们可以在键盘上按住**<Alt>**键的同时按**<G>**键(下划线字符)，该菜单项就被“点击”。这就意味着除了在 Excel 中点击菜单项外，还能执行键盘上的**<Alt+G>**来运行宏。

本章回顾

本书至今为止所编写的几乎所有代码都是在 Excel 或 MicroStation 的 VBA 环境中运行的。当在 Excel 中开发时，我们添加了引用“Bentley MicroStation DGN #.# Object Library”；当在 MicroStation 中开发且想使用 Excel 时，我们添加了引用“Microsoft Excel #.# Object Library”。用 GetObject 取得要连接的应用的现有实例，用 CreateObject 或 New 来建立要操作的应用的新实例。



38. 编写 VB6 应用程序

我们已经在 VBA 中编写了大量代码，现在到了在 Visual Basic 6 中编写代码的时候了。

本章包括：

- VBA 与 VB6 之间的不同
- VB6 工程基础
- 用 VB6 控制 MicroStation
- 编译和分发应用程序

VBA 与 VB6 之间的不同

VBA 和 VB6 有许多共同点。它们当然都用“Visual Basic”语言，工程都划分为代码模块、用户窗体和类模块，每个环境都允许向工程中添加引用。但也有一些不同点。

- 在 VBA 中，代码模块、类和用户窗体都包含在单个的.mvba 文件里。在 VB6 中，代码模块、类和用户窗体各自有其自己的文件(.bas、.cls 和.frm 文件)，而且还有一个工程文件(.vbp)将它们组织在一起；
- MicroStation VBA 工程必须在 MicroStation 的 VBA 环境中运行，Excel VBA 工程必须在 Excel 的 VBA 环境中运行。VB6 工程被编译成可执行程序(.exe)，可不依赖于任何应用程序来运行。VB6 工程还能编译成 DLL 文件和 ActiveX 控件(.ocx 文件)；
- 由于 VBA 工程是自包含的，所以对 VBA 工程中代码模块、类和用户窗体

的改变仍保留在工程中。多个 VB6 工程能利用同一个代码模块、类模块或用户窗体文件，所以对 VB6 资源的改变有可能影响多个 VB6 工程。这是一个强大的特性，但要注意在一个工程中对资源文件(.bas、.cls、.frm)的改变不要对其它的工程产生负面影响；

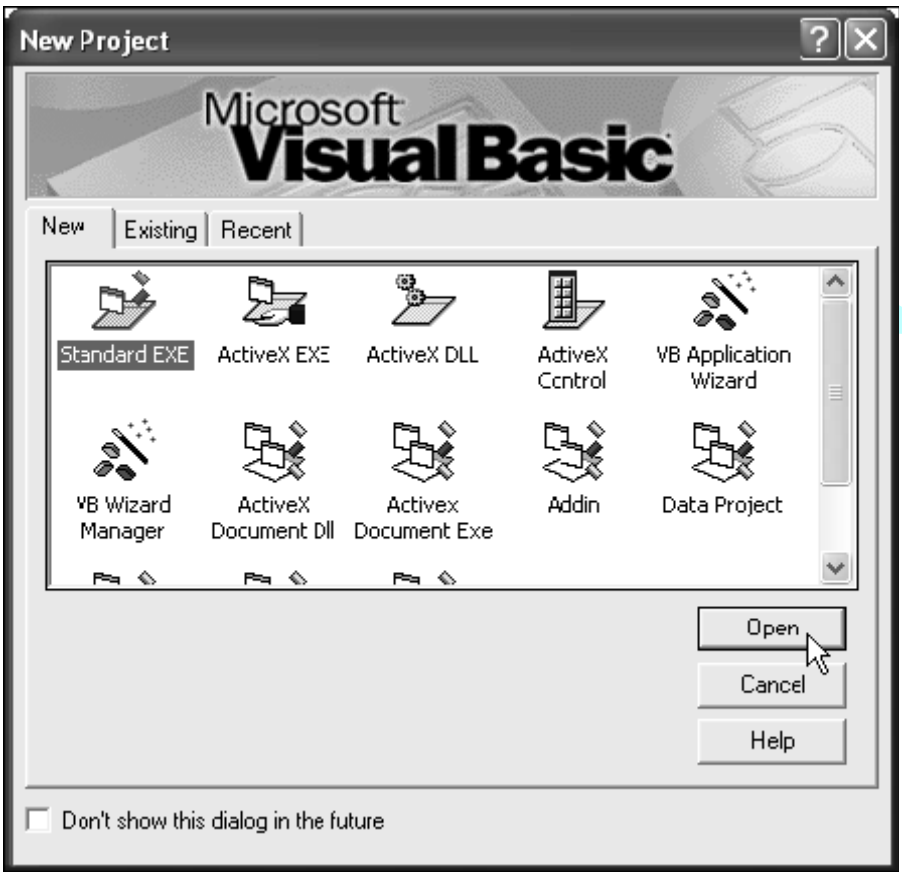
- 一个 VBA 工程只需提供单个.mvba 文件就能分发。VB6 工程虽然被编译成单个文件(.exe、.dll、.ocx)，但最好是用一个 Setup 程序来分发，因为要使程序正常运行可能还需要安装 DLL 和其它资源(如 Visual Basic 运行时库)；
- 由于 VBA 程序必须从宿主(host)应用程序中运行，所以我们知道在用户的计算机上有宿主应用程序。而由于被编译的 VB6 程序能独立于宿主应用程序运行，这就有可能造成在计算机上安装了为 MicroStation 设计的程序但计算机上没有安装 MicroStation；
- VBA 程序被分发后也易于调试，因为在用户的机器上就能启动调试且能单步进入源代码(在代码不被“锁定”的情况下)。VB6 程序被编译后，在用户的机器上不能单步调试或查看代码；
- VBA 环境随应用程序(如 MicroStation 和 Microsoft Excel)一起安装。VB6 则必须购买后安装才能使用；

VB6 工程的结构

我们已经指出 VB6 工程与 VBA 工程类似，它们都使用了代码模块、类模块和用户窗体。

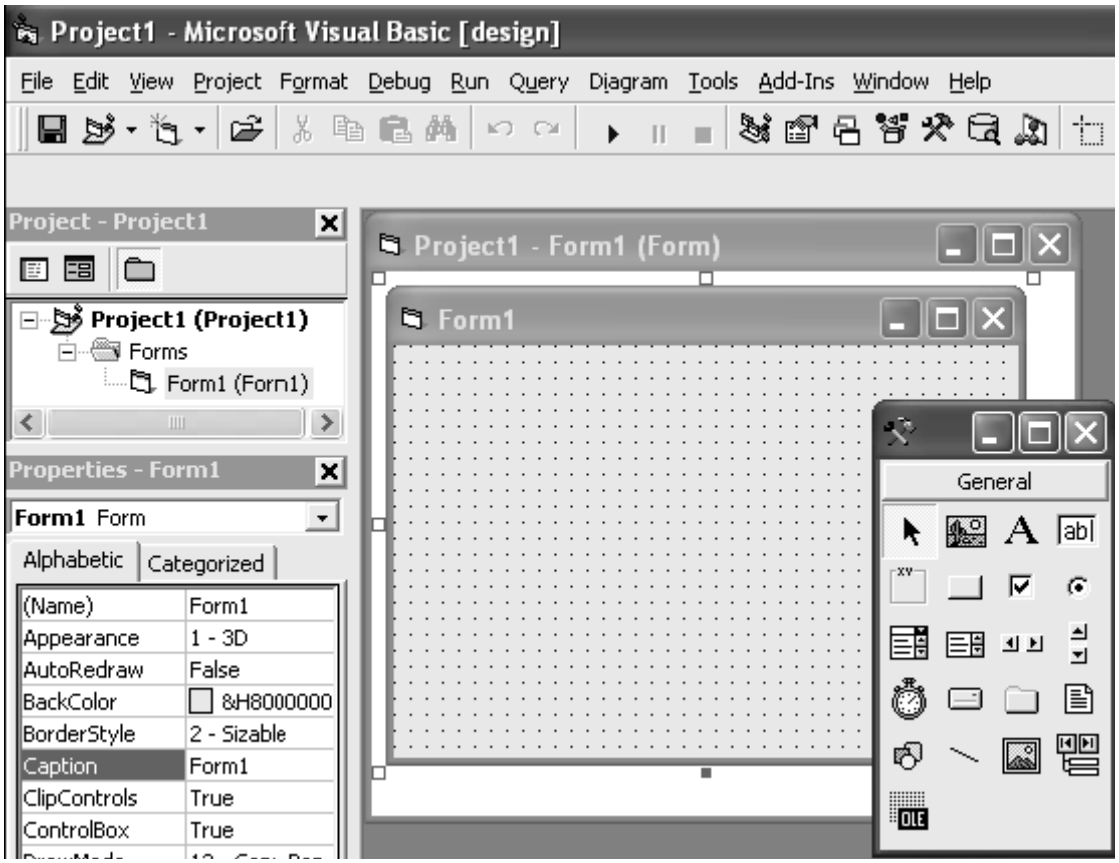
当开始一个新的 VB6 工程时，可以建立几种不同类型的 VB6 工程。我们先从建立一个“标准的 EXE”开始。

China BDN

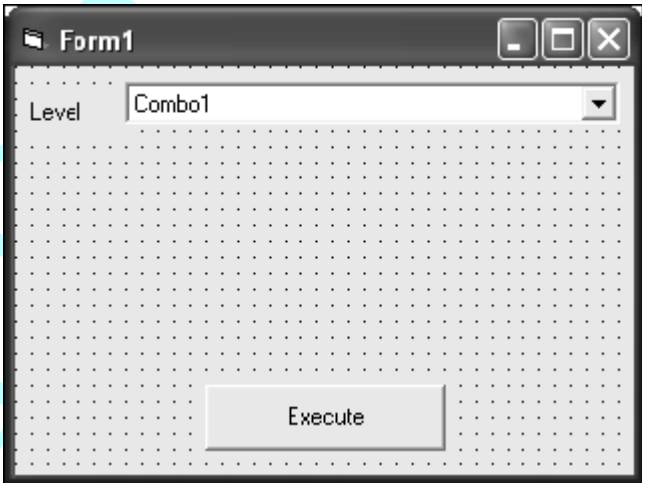


开始一个新的 VBA 工程时，缺省情况下建立了一个代码模块。而在 VB6 中开始一个新的“标准 EXE”工程时，一个叫做“Form1”的用户窗体被缺省建立。

China



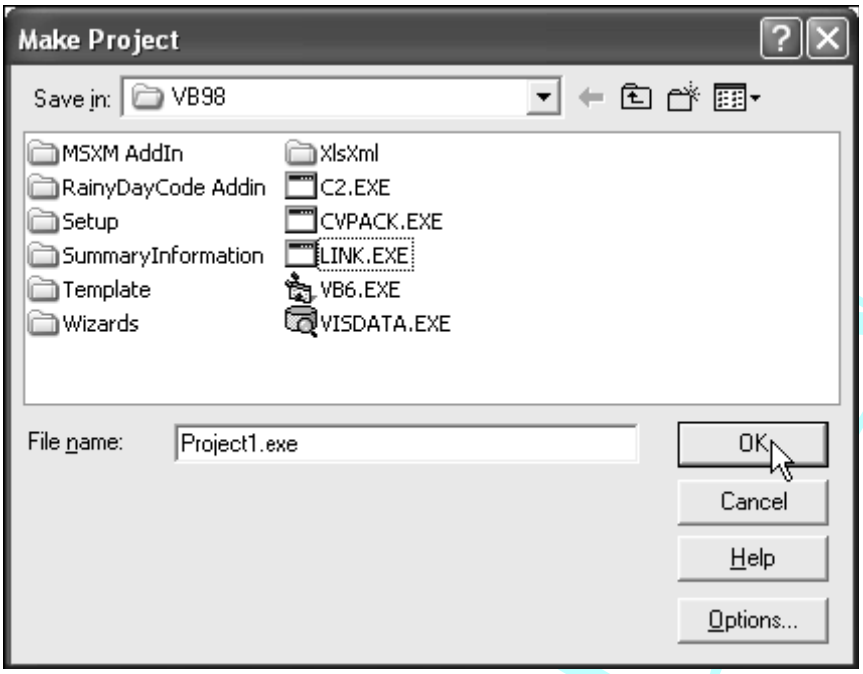
VB6 的 IDE 几乎与 VBA 的一样。能看到工程、属性、工具箱和用户窗体。



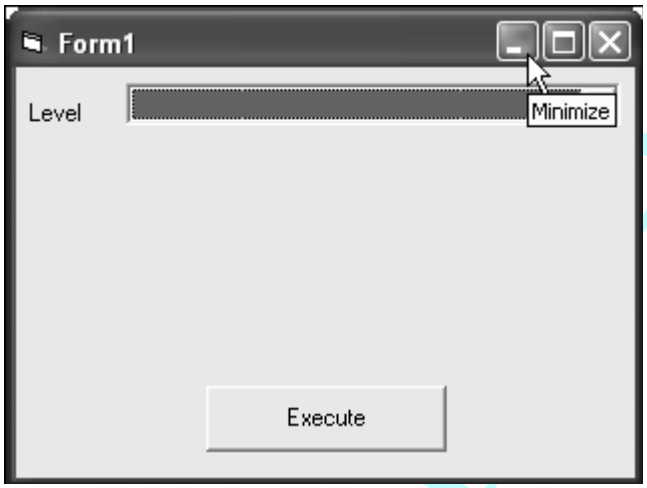
让我们从工具箱中取几个控件放入窗体中。我们将在窗体上放置一个复合框、一个标签和一个命令按钮。

现在，我们不需要对工程做任何工作，只是在 VB6 的菜单中选 **File > Make Project1.exe** 来编译程序。

Make Project 对话框显示出来，我们指定一个文件位置和文件名。



文件 Project1.exe 被建立，无论 MicroStation 运行与否，也无论 MicroStation 安装与否，该文件都能运行。



运行新程序能看到建立的窗体。它能被最大化也能被最小化，还能被缩放到任意大小。存在有控制这些窗体大小的窗体属性，但缺省情况下，窗体插入到工程中时具有最小化、最大化和缩放能力。

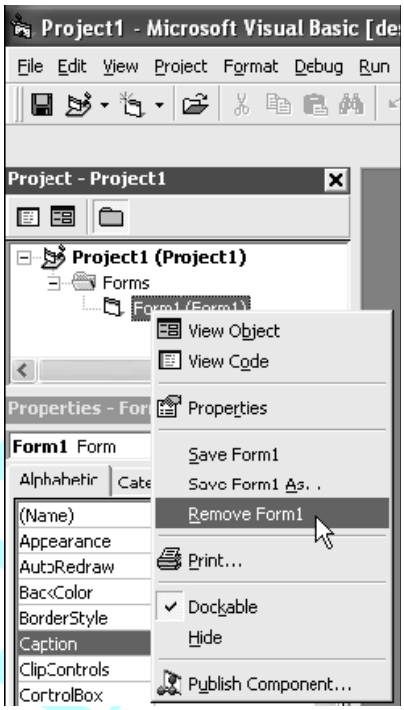
保存工程时会提问两个文件的名字和位置，这两个文件是 Form1.frm 和 Project1.vbp。这一点不同于 VBA，在 VBA 中仅提问单个.mvba 文件的名字和位置。

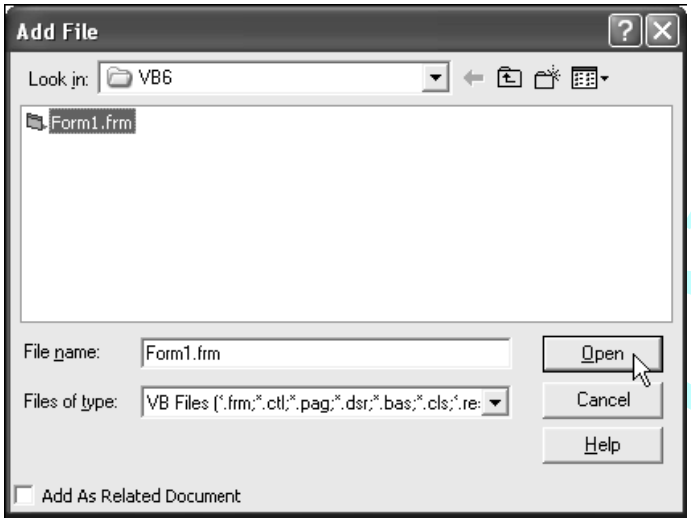
简单回顾一下，我们此时已经建立了三个不同的文件：一个.frm 文件(用户窗体)、一个.vbp 文件(VB6 工程文件)、一个.exe 文件(编译的可执行文件)。当准备向其他人分发应用时，在这三个文件中，只有.exe 文件是需要分发的。

让我们在 VB6 中“移除”正在使用的当前工程并且建立一个新工程。可以通过完全关闭 VB6 或在 VB6 文件菜单中选 **File > Remove Project** 来移除工程。

建立一个新工程时，一个新的用户窗体再次被建立。我们不要使用这个新的窗体，所以在工程窗口中的窗体上右击，然后选择 **Remove Form1** 来移除它。

移除完新的缺省窗体后，我们能看到现在只有一个空的“Project1”工程。让我们从前一个工程中添加窗体，这需要在菜单中选择 **Project > Add File**。





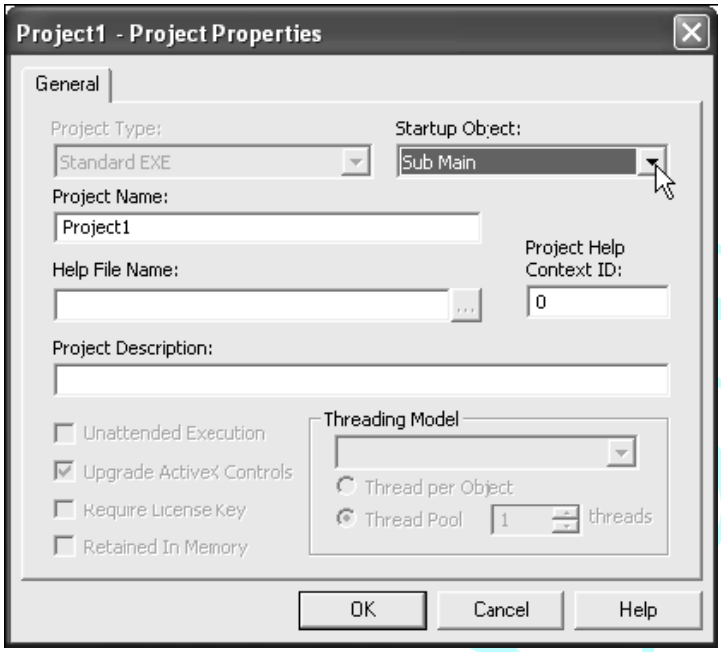
选择以前保存的窗体并点击 Open，它就被添加到工程中。在“Forms”文件夹中该窗体上双击能打开它，还能编辑它。

记住，我们没有添加前一个工程，只是它的窗体(前一工程的一部分)。

让我们给窗体添加另一个复合框和标签，并改变窗体的大小。



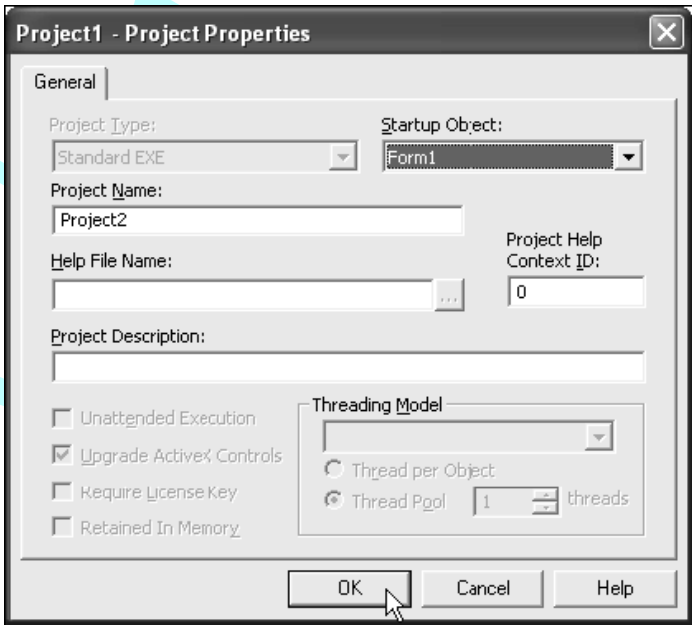
建立的第一个工程叫做“Project1”，这是 VB6 给的名字。我们想给这个工程取名为“Project2”，在 VB6 菜单中选择 **Project > Project1 Properties**。



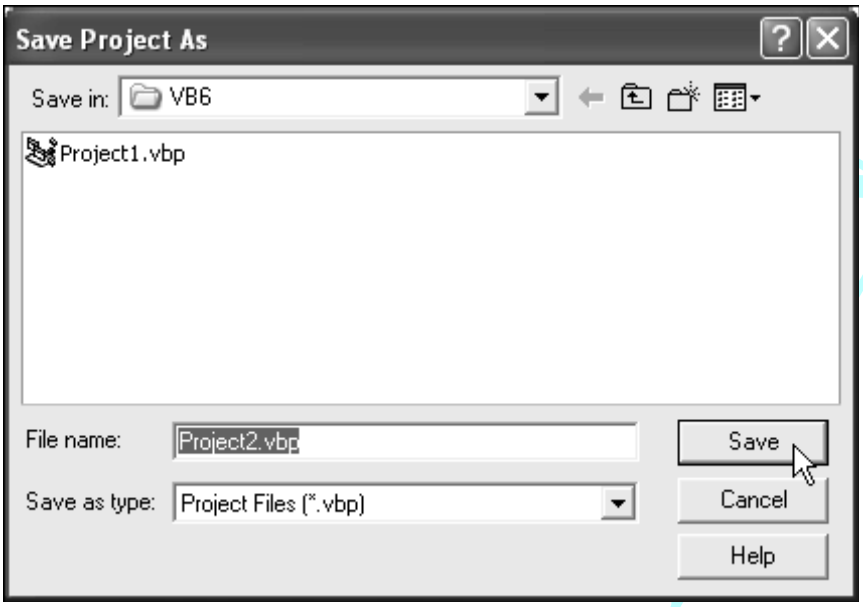
让我们讨论以下这个对话框。我们能看到 Project Name(工程名)处显示的是“Project1”，把它改为“Project2”。

Startup Object(启动对象)的选择很重要。现在，其设置是“Sub Main”，意思是当运行该程序时，不管是在设计时还是运行时，都将在代码模块中查找一个叫做“Main”的过程。对于我们的例子，我们不想启动“Sub Main”，因为我们没有“Sub Main”。我们想从列表中选择“Form1”。Startup Object 复合框包含有所有可用作启动对象的窗体名，如果该工程中有“Form2”和“Form3”的窗体，则它们也将在该列表中显示。

这些改变完成后，工程属性对话框看起来应该是这个样子：



点击 OK 后就能保存工程了。点击 Save(保存)按钮时提问工程文件名和保存的位置。



.vbp 文件名缺省为工程属性对话框中工程名的设置。在当前情况下，我们接受 Project2.vbp 缺省名。没有提问窗体的位置或名字，因为它已经保存在前一个工程中。.frm 文件以新的控件和新的大小属性保存。

在 VB 菜单中选择 **File > Remove Project2** 移除 Project2。现在再次打开 Project1。

当再次打开“Project1”并查看“Form1”时，能看出该窗体反映了在“Project2”中所做的改变。这一将程序设计元素与工程“链接”的功能很强大，但在使用这种强大功能之时，应当小心不要有意或无意地滥用这种功能。例如，如果忘记了 Form1 被两个工程所用，就可能会在“Project2”中作出修改从而使其在“Project1”中不能正常工作。

现在，我们在开始 Project2 前编译 Project1，在 Project2 中对 Form1 作出了改变。当执行 Project1.exe 时会发生什么？会得到新窗体还是得到编译 Project1 时的窗体？答案是：编译一个程序时，所有在编译时的东西都被编译到了可执行文件中。可执行程序不会改变来反映对其设计元素的修改。然而，如果打开 Project1 并重新编译，则新的可执行文件(.exe)将能反映重编译该工程时所做的修改。

这一练习教会了我们几点非常重要的原则。

- 多个 VB 工程(.vbp 文件)能利用相同的资源文件(.frm、.bas、.cls 等)；
- 编译的.exe 文件是静态的。对编译成.exe 文件的设计元素的修改不能反映到.exe 文件中，除非重新编译它；
- 缺省情况下，编译一个.exe 文件并不引起对源文件.frm、.bas 或.cls 的保存。也有可能用 VB6 建立了整个程序并编译它而没有保存任何源文件到磁盘。虽然这样存在潜在的危险，但在如下情况下却很有用：我们想做出相当大的改变“仅仅是为了看能发生什么”并且不想失去“当前工作中”的代码。我们可以做出改变，在设计模式运行它或编译它然后“看发生了什么”。只有在知道代码正常工作后才保存工程。另一方面，如果在每次编译工程时想让 VB6 自动保存工程，可以改变 **Tools > Options > Environment 选项卡 > When a program starts** 设置。

迄今为止，我们已经建立了两个“标准 EXE”工程。还有两类值得注意的工程，它们是：

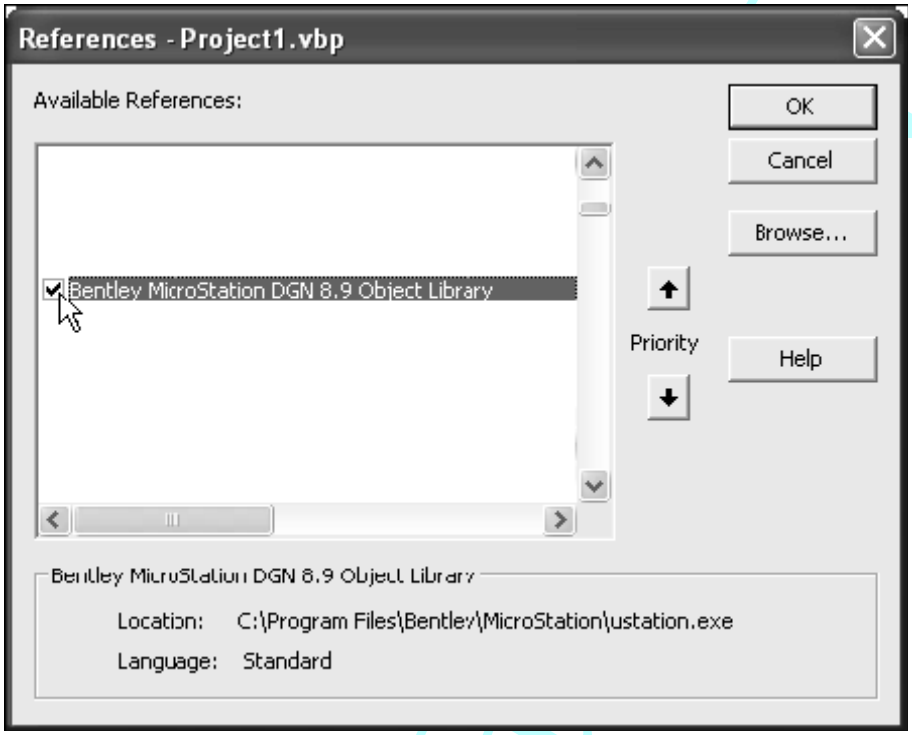
- ActiveX DLL — 能让我们建立一个不能单独执行但能被其它程序“连接”的“程序”。一个 ActiveX DLL 的例子就是“Microsoft ActiveX Data Objects Library”。我们不能把它作为独立的程序来“执行”，但能“连接”到它以进行便捷的数据库存取；
- ActiveX 控件 — 能让我们建立自己的显示在 VBA 和 VB6 工具箱中的“控件”。ActiveX 控件的例子是一个能自动显示.dgn 文件中层列表的复合框。我们从基本的复合框开始，编写代码对其赋值。一旦完成并编译(成一个.ocx 文件)，就能将其放入 MicroStation VBA 环境中的窗体里，它将自动用层来赋值自己。

用 VB6 控制 MicroStation

我们已经建立了几个 VB6 工程，也已经在窗体中放置了一些控件并编译了其中的一个工程。但即使应用“Project1”被编译了，它也什么事情都没做。我们可以点击复合框或按钮，但什么也没有发生。可以最小化、最大化和缩放窗体，但这其实也没做什么事情。让我们看一看从应用“Project1”中控制 MicroStation 所需的步骤。

与 MicroStation 通讯的第一步是添加引用。在 VBA 中是用菜单项 **Tools > References** 实现的，在 VB6 中要用菜单项 **Project > References** 完成。

显示的列表会随计算机的不同而不同，因为每台计算机上都安装有不同的软件。列表按字符顺序排列，我们要浏览到“Bentley MicroStation DGN #.# Object Library”。



一旦选择这个引用后，就能很容易地操作 MicroStation 了。

现在，我们要在窗体显示前对复合框赋值。可以在“Form Load(窗体装载)”事件中编写一些代码(VBA 中使用的是窗体初始化事件)。

窗体装载事件代码如下：

```
Private Sub Form_Load()
    Dim myMStationC As MicroStationDGN.ApplicationObjectConnector
    Dim myMStation As MicroStationDGN.Application
    Dim myLevel As MicroStationDGN.Level
    Dim myCell As MicroStationDGN.CellInformation
    Dim myCellInfoEnum As MicroStationDGN.CellInformationEnumerator
    Set myMStationC = GetObject(, _
                        "MicroStationDGN.ApplicationObjectConnector")
    Set myMStation = myMStationC.Application
    'Populate Levels
    For Each myLevel In myMStation.ActiveDesignFile.Levels
```

```
        Combo1.AddItem myLevel.Name
    Next
    'Populate Cells
    If myMStation.IsCellLibraryAttached = True Then
        Set myCellInfoEnum = _
                    myMStation.GetCellInformationEnumerator(True, True)
        While myCellInfoEnum.MoveNext
            Set myCell = myCellInfoEnum.Current
            Combo2.AddItem myCell.Name
            myCellInfoEnum.MoveNext
        Wend
    End If
End Sub
```



让我们看一下代码。我们想连接到 MicroStation，因而使用了 GetObject 并把 **MicroStationDGN.ApplicationObjectConnector** 作为“取得”的类来连接 MicroStation 的。然后从 **ApplicationObjectConnector** 得到 **Application 对象**。一旦有了 ApplicationObjectConnector 的 Application，我们就能像在 VBA 中一样来操作 MicroStation 了。

我们把当前设计文件中的每个层名放入“Combo1”，如果连接有单元库，就把库中每个单元名添加到“Combo2”。

用户点击按钮后发生什么呢？

```
Private Sub Command1_Click()
    Dim myMStationC As MicroStationDGN.ApplicationObjectConnector
    Dim myMStation As MicroStationDGN.Application
    Dim myCell As CellElement
    Dim CellInsPt As Point3d
    Set myMStationC = GetObject(, _
                    "MicroStationDGN.ApplicationObjectConnector")
    Set myMStation = myMStationC.Application
```

```
    Set myCell = myMStation.CreateCellElement3(Combo2.Text, _
                                                CellInsPt, True)
    myCell.Level = myMStation.ActiveDesignFile.Levels(Combo1.Text)
    myMStation.ActiveModelReference.AddElement myCell
End Sub
```

点击命令按钮后，在选择的层上(0,0,0)点处插入选择的单元。

每次使用 GetObject 时，需要花事件来连接到 MicroStation。让我们稍微修改一下代码，不需要每次点击按钮时都连接 MicroStationDGN.ApplicationObjectConnector。

```
‘General Declarations Area of Form
Dim myMStationC As MicroStationDGN.ApplicationObjectConnector
Dim myMStation As MicroStationDGN.Application
Private Sub Command1_Click()
    Dim myCell As CellElement
    Dim CellInsPt As Point3d
    Set myCell = myMStation.CreateCellElement3(Combo2.Text, _
                                                CellInsPt, True)
    myCell.Level = myMStation.ActiveDesignFile.Levels(Combo1.Text)
    myMStation.ActiveModelReference.AddElement myCell
End Sub
Private Sub Form_Load()
    Dim myLevel As MicroStationDGN.Level
    Dim myCell As MicroStationDGN.CellInformation
    Dim myCellInfoEnum As MicroStationDGN.CellInformationEnumerator
    Set myMStationC = GetObject(, _
                        "MicroStationDGN.ApplicationObjectConnector")
    Set myMStation = myMStationC.Application
    'Populate Levels
    For Each myLevel In myMStation.ActiveDesignFile.Levels
        Combo1.AddItem myLevel.Name
```

```
    Next
    'Populate Cells
    If myMStation.IsCellLibraryAttached = True Then
        Set myCellInfoEnum = _
                myMStation.GetCellInformationEnumerator(True, True)
        While myCellInfoEnum.MoveNext
            Set myCell = myCellInfoEnum.Current
            Combo2.AddItem myCell.Name
            myCellInfoEnum.MoveNext
        Wend
    End If
End Sub
```

在窗体的通用声明区中声明变量 **myMStationC** 和 **myMStation** 后，只要这些变量在窗体装载事件中被初始化设置，它们就能在窗体的其它地方使用，如命令按钮的单击事件中。

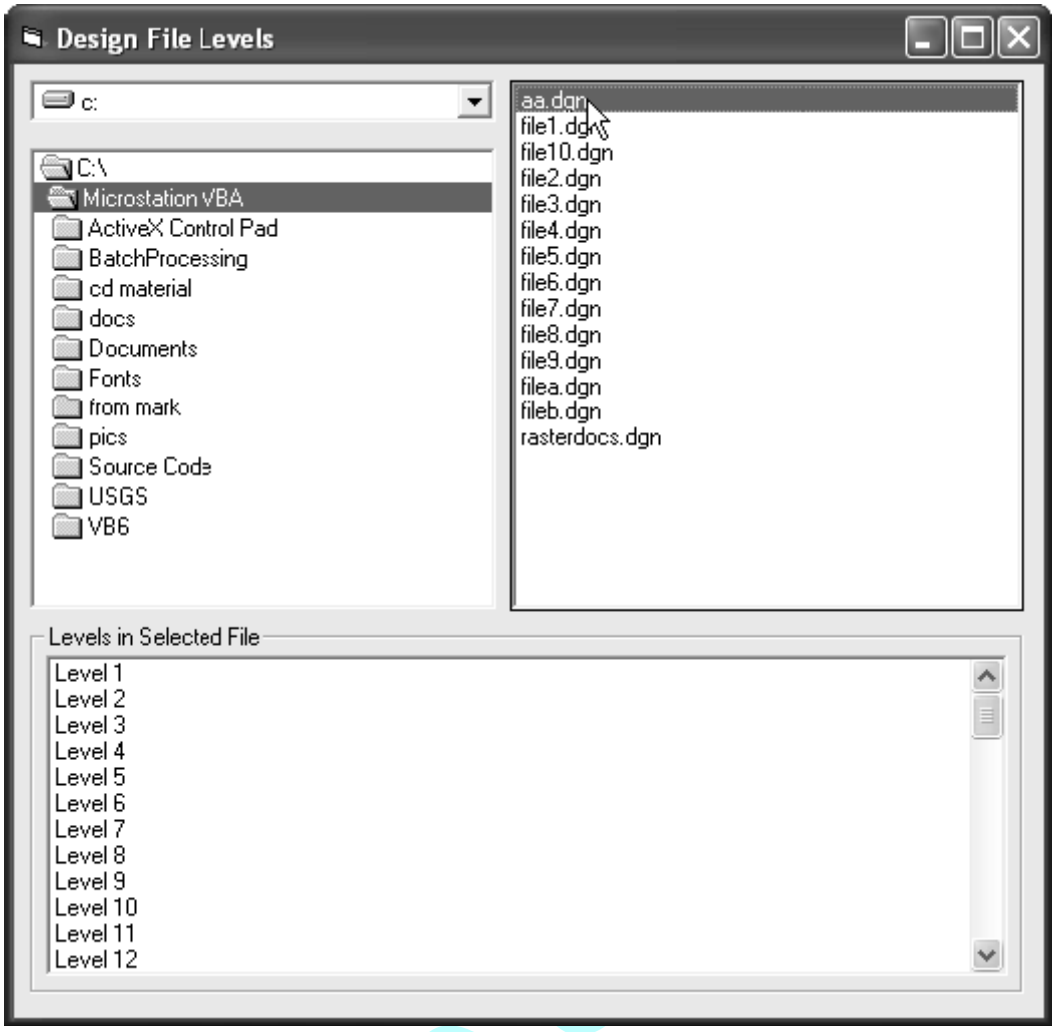
我们能在 VB6 中执行应用程序的测试运行。点击 VB6 工具栏中的 Start 按钮、使用 VB6 菜单 **Run > Start** 或按**<F5>**键在 VB6 中执行程序。我们也能像在 VBA 中那样反复用**<F8>**键单步跟踪代码。

现在让我们编译程序，**File > Make Project1.exe** 把工程编译为文件 **Project1.exe**。编译完成后，可以在 Windows 资源管理器中双击它来执行程序或在桌面添加一个快捷方式。

第一个 VB6 工程非常简单。让我们建立一个新工程，它利用一些标准 VBA 工具箱中没有的控件。命名该工程为“Project3”。

下面是窗体执行时的样子：

China BDN





当选择一个驱动器和文件夹时，被选文件夹下的 DGN 文件显示在文件列表中。当选择一个文件时，文件中的层显示在列表框中。列表框在 VBA 中可用，但驱动器列表框、目录列表框和文件列表框在 VBA 中不可用。

我们能在工具箱中看到驱动器、目录和文件列表框，这些是在 VB6 中可用的三个控件，它们在标准 VBA 工具箱中不存在。

现在让我们看一下代码，我们知道该程序将要干什么：

```
‘General Declarations Area
Dim myMStationC As MicroStationDGN.ApplicationObjectConnector
Dim myMStation As MicroStationDGN.Application
Private Sub Form_Load()
    Set myMStationC = GetObject(, _
```

```
                                "MicroStationDGN.ApplicationObjectConnector")
    Set myMStation = myMStationC.Application
End Sub
Private Sub Drive1_Change()
    Dir1.Path = Drive1.Drive
    List1.Clear
End Sub
Private Sub Dir1_Change()
    File1.Path = Dir1.Path
    List1.Clear
End Sub
Private Sub File1_Click()
    Dim myLevel As MicroStationDGN.Level
    Dim myDF As MicroStationDGN.DesignFile
    Set myDF = myMStation.OpenDesignFileForProgram( _
                                Dir1.Path & "\" & File1.FileName, True)
    List1.Clear
    For Each myLevel In myDF.Levels
        List1.AddItem myLevel.Name
    Next
    myDF.Close
End Sub
```

窗体装载后连接到 MicroStation。驱动器改变时我们更新目录列表框，目录列表框改变时我们更新文件列表框。当用户在文件列表框中点击一个.dgn 文件时，我们以“ForProgram”方式打开这个文件并将其层添加到层列表框中。用 OpenDesignFileForProgram 打开的文件不会出现在 MicroStation 编辑器中而只是在内存中打开。由于不对.dgn 文件进行任何修改，所以以只读方式打开。

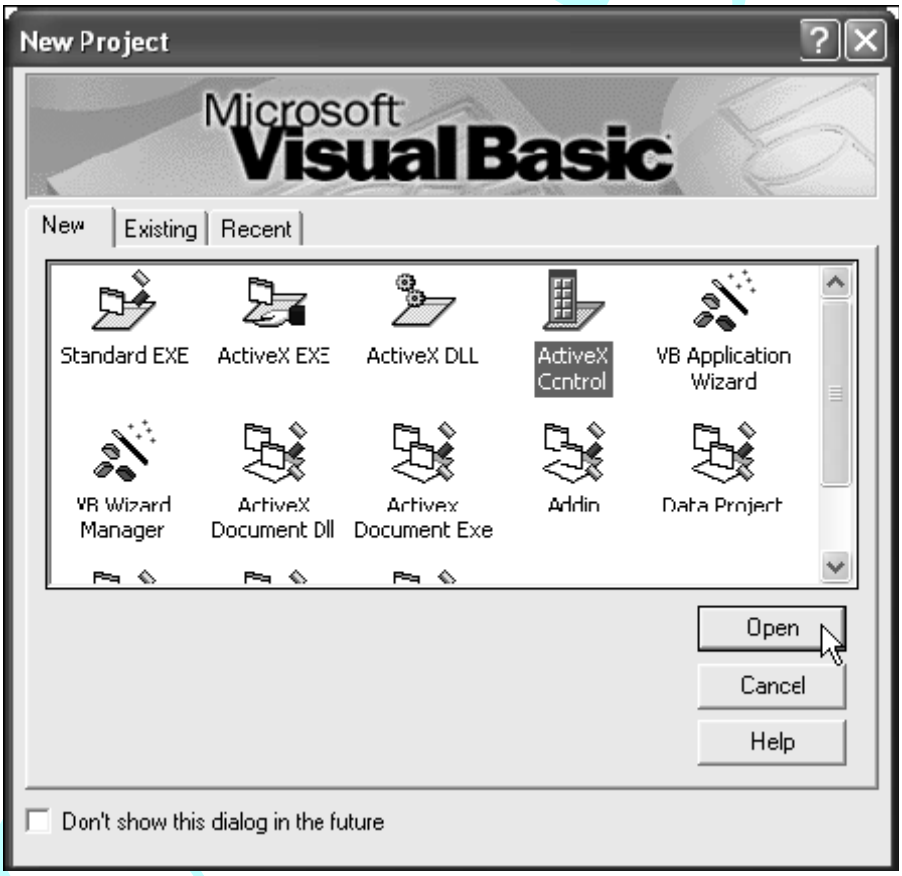
把窗体命名为“Form3”，把工程命名为“Project3”。保存工程后编译并运行它。

在 VB6 中建立 ActiveX 控件

VBA 为我们提供了许多标准控件：列表框、文本框、复合框、复选框、命令按钮等。在使用 MicroStation 的过程中，我们可能看到一些用户 GUI 控件，我们希望这些控件也能出现在 VBA 中。VB6 的优点之一就是能在 VB6 中创建自己的控件然后在 VBA 中使用。关于创建控件可能需要一整本书来单独讨论(实际上，已经有整本的书在讨论该主题)。我们在此只是建立一个简单的能用于 VB6 或 VBA 中的控件，该控件显示当前设计文件中的模型和层。我们将用 TreeView 控件来实现该功能。

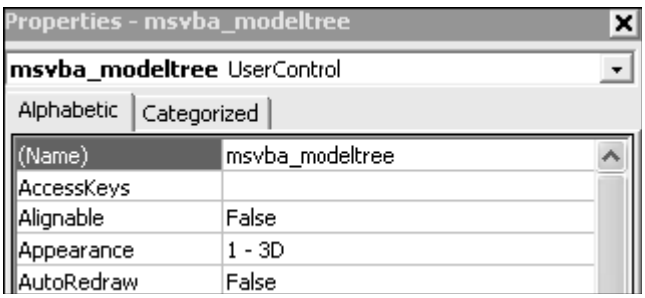
下面是建立该控件的步骤：

1、在 VB6 中建立一个新的“ActiveX 控件”工程；

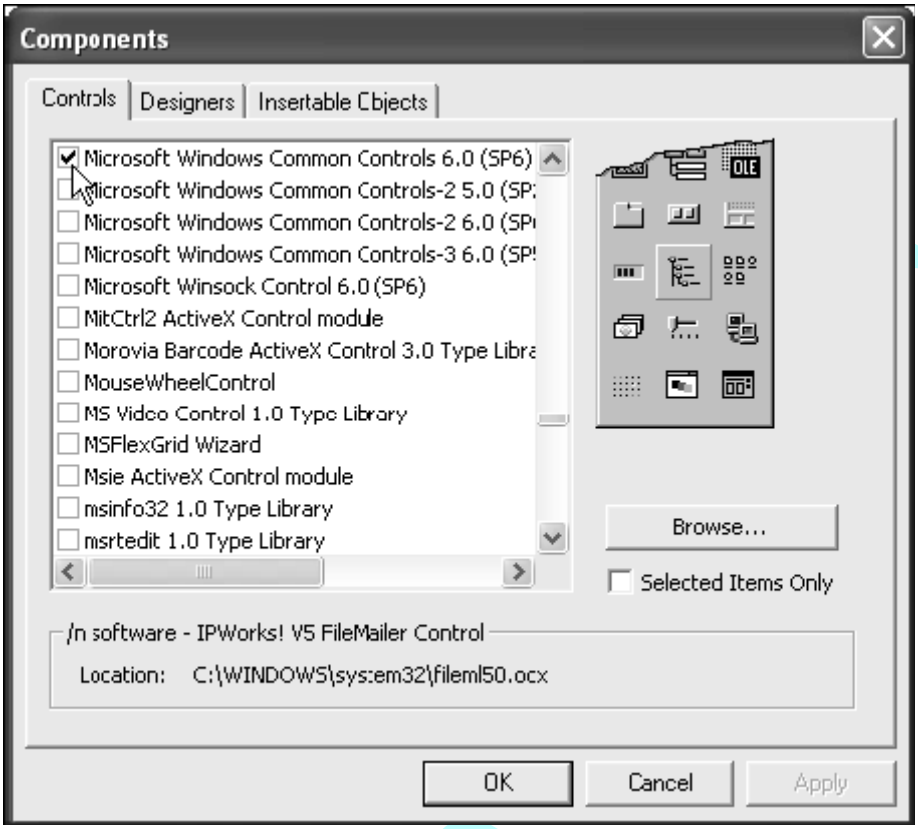


2、该控件名称为“msvba_modeltree”；

3、右击工具箱，点击“Component(组件)”，在列表中选择“Microsoft Windows



Common Controls 6.0”，然后点击 OK 按钮；

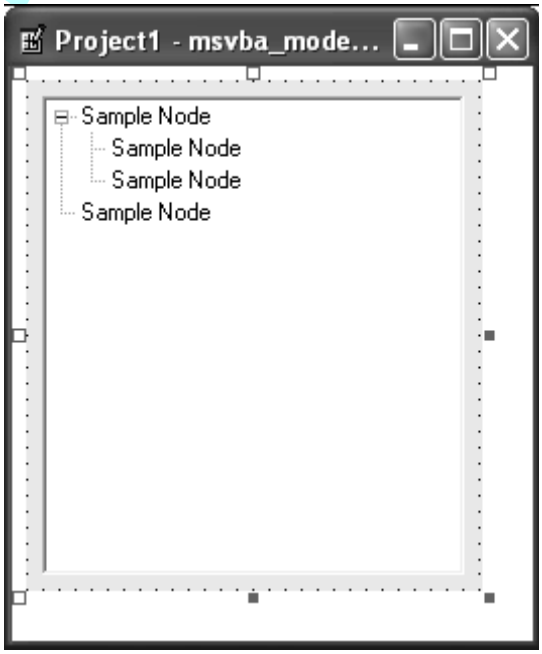


4、在用户控件区添加一个树视图(TreeView)控件并取名为“tv1”；

5、将树视图的“Style”属性改为“7 - tvwTreelinesPlusMinusPictureText”

6、在控件工程中选 **Project > References** 添加引用“Bentley MicroStation DGN 8.9 Object Library”；

现在有了新控件的框架。



至此，我们有了一个叫做“msvba_modeltree”的控件，该控件上有一个叫做“tv1”的树视图控件。我们想让这个控件做什么事情呢？

- 在树视图的顶级节点“Models”下显示当前设计文件中的模型；
- 在树视图的顶级节点“Levels”下显示当前设计文件中的层；
- 当用户在树视图的节点上点击时建立一个叫做“SelectionChanged”的事件并返回选择的节点对象；

➢ 创立叫做“GetModel”和“GetLevel”的方法，分别返回在树视图中选择的模型对象或层对象；

➢ 允许用户(其实是程序员)缩放控件，树视图也随之缩放。

下面是控件中的所有代码：

```
‘General Declarations Area
Dim myMSApp As MicroStationDGN.Application
Dim myMSAppCon As MicroStationDGN.ApplicationObjectConnector

Event SelectionChanged(SelectedNode As Node)

Private Sub UserControl_Initialize()
    Dim myModel As MicroStationDGN.ModelReference
    Dim myNode As Node
    Dim myLevel As MicroStationDGN.Level
    Set myMSAppCon = GetObject(, _
                    "MicroStationDGN.ApplicationObjectConnector")
    Set myMSApp = myMSAppCon.Application
    'Models
    Set myNode = tv1.Nodes.Add(, , "tv1Models", "Models")
    For Each myModel In myMSApp.ActiveDesignFile.Models
        tv1.Nodes.Add "tv1Models", tvwChild, “mod_” & myModel.Name, _
                    myModel.Name
    Next
    myNode.Sorted = True
    'Levels
    Set myNode = tv1.Nodes.Add(, , "tv1Levels", "Levels")
    For Each myLevel In myMSApp.ActiveDesignFile.Levels
        tv1.Nodes.Add "tv1Levels", tvwChild, “lvl_” & myLevel.Name, _
                    myLevel.Name
    Next
```


China BDN

```
    myNode.Sorted = True
End Sub
Private Sub UserControl_Resize()
    tv1.Width = UserControl.Width - tv1.Left * 2
    tv1.Height = UserControl.Height - tv1.Top * 2
End Sub
Private Sub tv1_Click()
    RaiseEvent SelectionChanged(tv1.SelectedItem)
End Sub
Function GetLevel() As MicroStationDGN.Level
    Dim xSplit As Variant
    xSplit = Split(tv1.SelectedItem.FullPath, "\")
    Select Case UCase(xSplit(0))
    Case "MODELS"
        Set GetLevel = Nothing
    Case "LEVELS"
        If UBound(xSplit) > 0 Then
            Set GetLevel = myMSApp.ActiveDesignFile.Levels(xSplit(1))
        Else
            Set GetLevel = Nothing
        End If
    End Select
End Function
Function GetModel() As MicroStationDGN.ModelReference
    Dim xSplit As Variant
    xSplit = Split(tv1.SelectedItem.FullPath, "\")
    Select Case UCase(xSplit(0))
    Case "MODELS"
        If UBound(xSplit) > 0 Then
            Set GetModel = myMSApp.ActiveDesignFile.Models(xSplit(1))
```


China BDN

```
            Else
                Set GetModel = Nothing
            End If
        Case "LEVELS"
            Set GetModel = Nothing
        End Select
End Function
Private Sub UserControl_Terminate()
        Set myMSApp = Nothing
        Set myMSAppCon = Nothing
End Sub
```

现在让我们看一下放入这段代码中的事件，以搞清楚都发生了什么。

Private Sub UserControl_Initialize()

控件初始化时，我们取得 MicroStation 的应用对象以及当前设计文件的模型和层，然后把模型和层放入树视图“tv1”中。

Private Sub UserControl_Resize()

当控件被缩放时，我们根据控件的宽和高以及树视图的 Left 和 Top 属性改变树视图的大小。

Private Sub tv1_Click()

当用户点击树视图的一个节点时，发出 SelectionChanged 事件，以树视图中选择的节点作为事件的参数。注意，SelectionChanged 事件在控件的通用声明区中声明。

函数 GetLevel()与 MicroStationDGN.Level

该控件添加到 VB6 或 VBA 中的一个窗体后，就能使用控件的 GetLevel 方法了，控件返回选择的层。如果树视图中选择的节点不是一个层就返回“Nothing”。

函数 GetModel()与 MicroStationDGN.ModelReference

与 GetLevel 类似，如果树视图中选择了一个模型，则 GetModel 函数返回模型对象。

调试 ActiveX 控件工程

我们刚刚建立了一个新的 ActiveX 控件框架，如何测试它看其是否工作正常呢？我们可以编译它，然后将其插入一个新的 VB6 或 VBA 工程中。但这样做会造成每次发现一个错误时就需要重新编译，而且还不能在开发中单步调试代码。

VB6 能够建立工程组。正如其名字所描述的那样，工程组是一组独立的 VB6 工程，它们在一个 VB6 会话期打开。在建立新工程组前应当保存我们的 ActiveX 控件工程，在保存工程前又需要在 **Project > Properties** 对话框中对它命名。我们命名控件工程为“msvbaControls”。

保存 ActiveX 控件工程时会同时保存工程(.vbp 文件)和工程中的控件(.ctl 文件)。让我们保存该工程和控件，控件的文件名叫做“msvba_modeltree”。

现在可以向控件工程中添加新工程了，这样就建立了一个工程组。

1、选择 **File > Add Project**(不要点击“New Project”，我们需要添加一个工程)；

2、选择“Standard EXE(标准 EXE)”；

3、在 VB6 菜单中选择 **File > Save ProjectGroup**，VB6 会提示一个工程组文件名；

4、保存工程为“msvba_modeltree”。VB6 还提问新工程和窗体的文件名；

5、保存新工程和窗体为“testingmodeltree”；

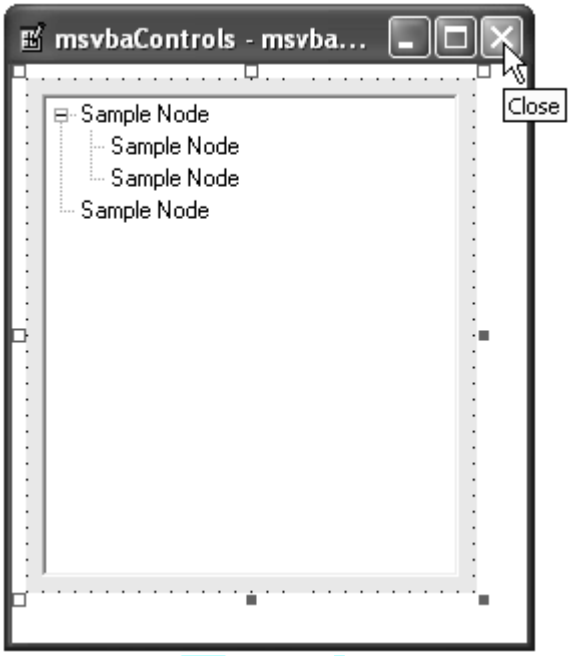


现在来查看 VB6 中的工程窗口，我们能看到有两个装载的工程。在工程组中工作时，其中的某一个工程要设置为“启动工程”。因为控件自己不能执行，所以将新的“testingmodeltree”工程设置为启动工程。在该工程上右击并选择“Set as Start Up(设置为启动)”完成设置。

在工程窗口中双击刚刚建立的控件以使其显示出来，接下来点击控件窗口顶

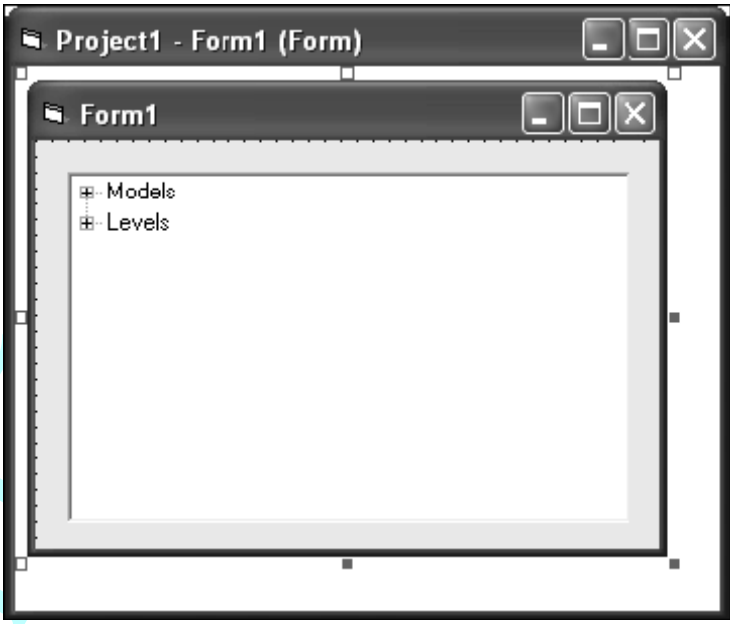
部的 Close(关闭)按钮来关闭控件。

为什么要关闭控件窗口呢？因为当控件打开时是不能将其添加到新工程中的。我们不需要卸载控件工程，仅需关闭控件窗口即可。



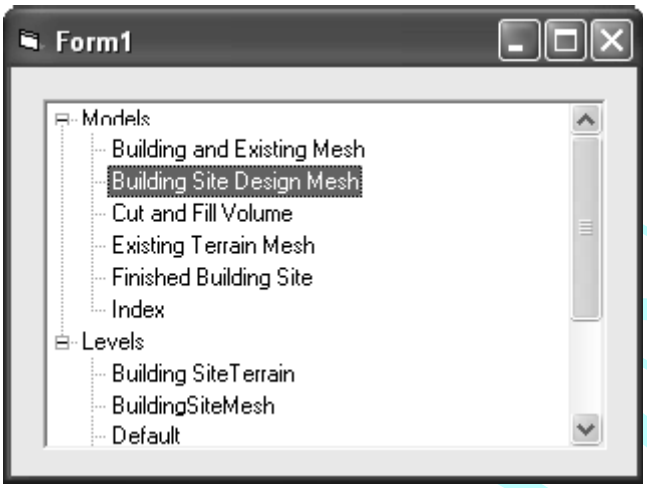


现在，当我们选择新工程的窗体并查看工具箱时，应当看到我们创建的控件出现在列表中。点击我们的新控件并将其拖拽到新窗体中能初始化该控件。



运行新工程，窗体显示出来且树视图可用。

至此，我们已经建立了一个新的 ActiveX 控件并且将其添加到了一个新的利用该控件的工程。现在让我们给新工程添加一点代码来测试控件的事件和方法。我们还需要添加引用“Bentley MicroStation DGN Object Library”。




```
Private Sub msvba_modeltree1_SelectionChanged( _
                                    SelectedNode As MSComctlLib.Node)
    Dim myLevel As MicroStationDGN.Level
    Dim myModel As MicroStationDGN.ModelReference
    MsgBox SelectedNode.Text & vbCr & SelectedNode.FullPath
    Set myLevel = msvba_modeltree1.GetLevel
    If myLevel Is Nothing = False Then
        MsgBox myLevel.Name & vbTab & myLevel.Number, , "LEVEL"
    End If
    Set myModel = msvba_modeltree1.GetModel
    If myModel Is Nothing = False Then
        MsgBox myModel.Name & vbTab & myModel.Description, , "MODEL"
    End If
End Sub
```

这就是在控件中建立的“SelectionChanged”事件。在该事件中显示了选择节点的 Text 属性和 FullPath 属性，然后使用了 GetLevel 和 GetModel 方法。

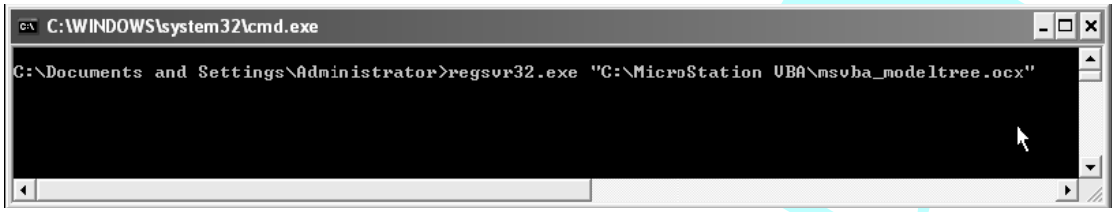
编译 ActiveX 控件

在将 ActiveX 控件编译成.ocx 文件前，必须将 ActiveX 控件工程设置为“启动”工程。设置完成后，从 VB6 菜单中选择 **File > Make msvba_modeltree.ocx**。让我们将 ActiveX 控件编译到控件工程文件夹中，这样可以容易找到它。

编译 ActiveX 控件与编译标准的 EXE 有点不同。文件扩展名当然不同了，

除此之外控件还在开发用的计算机上被“注册”。在要使用该控件的其它计算机上，这一注册步骤需要手工完成或用一个设置程序完成。在命令窗口中用“RegSvr32.exe”来注册控件。

在 Windows 的开始按钮下点击“运行”快捷方式打开运行对话框，在运行对话框中键入“cmd”并点击确定。

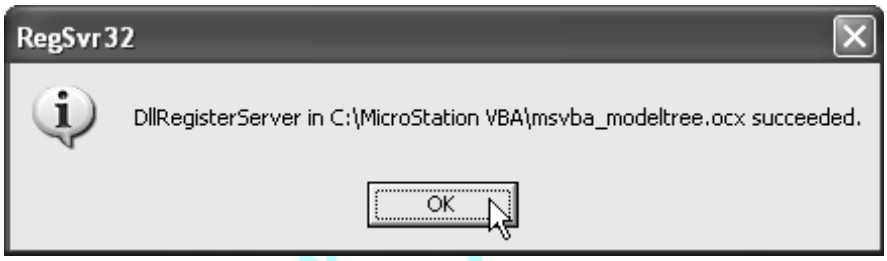


现在，键入：

regsvr32.exe “C:\MicroStation VBA\msvba_modeltree.ocx”

并按**<Enter>**键。输入的路径应指向.ocx 文件的位置，该.ocx 文件可能在一个别的文件夹下，所以路径可能需要调整。

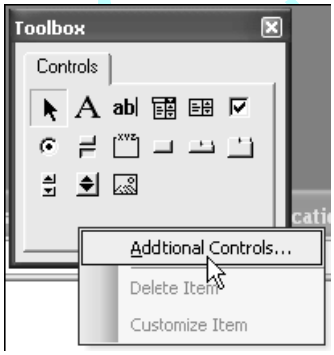
按**<Enter>**后，如果所有输入都正确，就会看到如下对话框：



这一注册步骤需要在使用该控件的任何计算机上执行。如果未执行该注册步骤，控件将不可用。

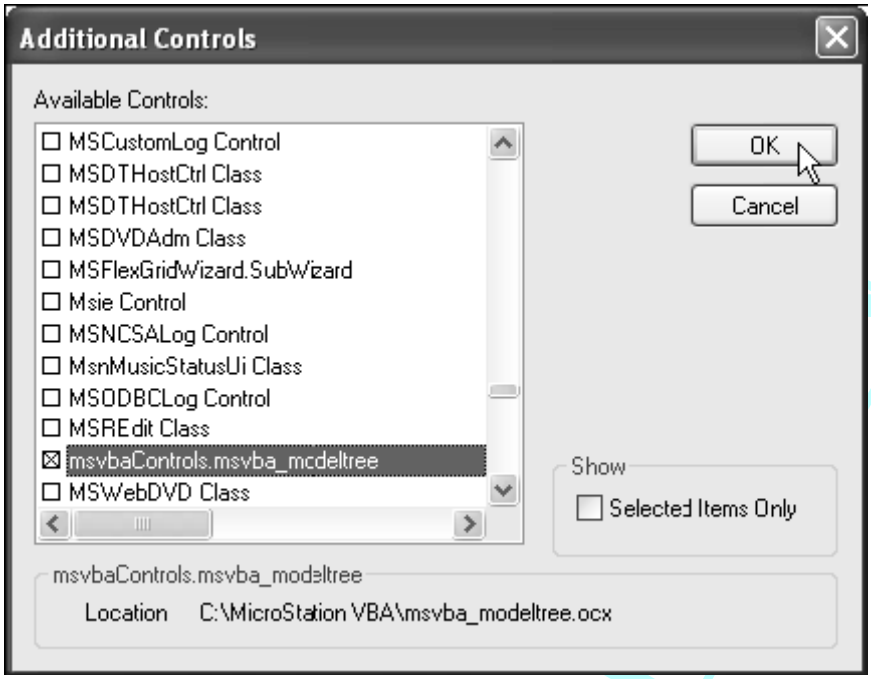
既然已经编译了这个 ActiveX 控件，那么让我们在 MicroStation 的 VBA 环境中看一看它。

在 MicroStation VBA 工程中插入一个新窗体后，右击工具箱并选择 Additional Controls(附加控件)。



我们新建的控件应当在列表中显示出来。选择它后会在 VBA 的工具箱中出现。

将控件拖拽到窗体中并加以缩放，然后按**<F5>**键来运行窗体。



此时会显示一个消息框，警告我们使用了这个 ActiveX 控件。跳过这个对话框后窗体显示出来，控件中赋有当前设计文件中的模型和层。

建立 ActiveX 控件相当简单而直接。我们已经建立了一个与 MicroStation 交互的控件。因此，下次在 MicroStation 或其它应用中看到想用的用户界面，就可以建立自己的带有完整属性、方法和事件的控件。这些自定义的控件能用于 VB6 和 VBA 中。

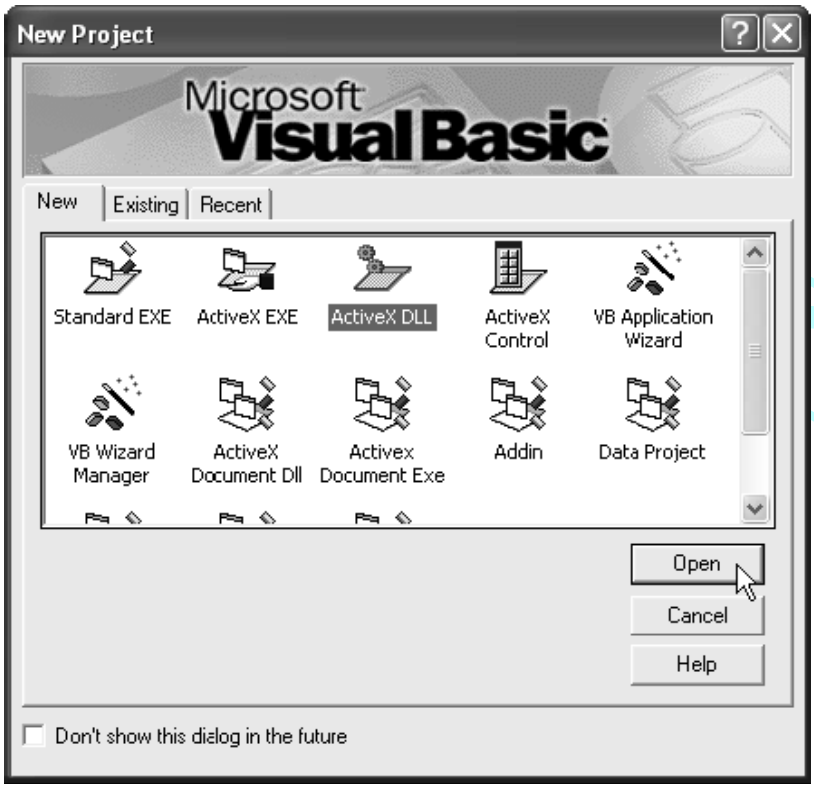
建立 ActiveX DLL

我们刚刚建立完一个 ActiveX 控件。ActiveX 控件能让我们基于现有的控件建立自定义用户界面的控件，有些这样的控件不能在 VBA 中使用。现在我们将考察如何建立一个 ActiveX DLL。

ActiveX DLL 是能通过 VB 和 VBA 应用存取的编译的.dll 文件。它们常用于将功能“包装”在单个.DLL 文件中，这样的.DLL 文件叫做“包装器”。建立 ActiveX DLL 的目的是把特定的逻辑或计算从总的应用程序逻辑中分离出来。

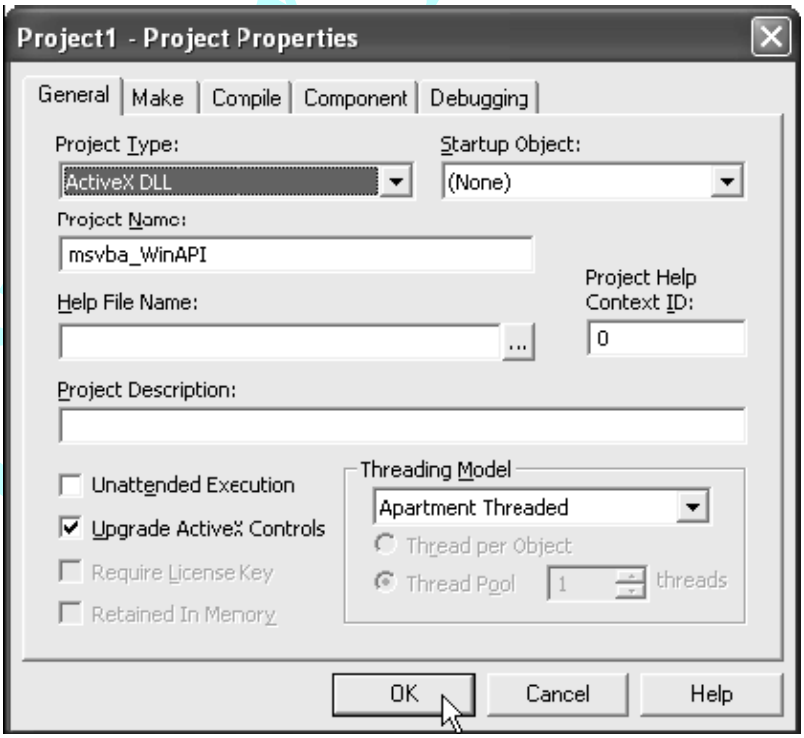
现在来建立一个 ActiveX DLL 文件。启动一个新的 VB6 会话，选择“ActiveX DLL”作为要建立的工程类型。

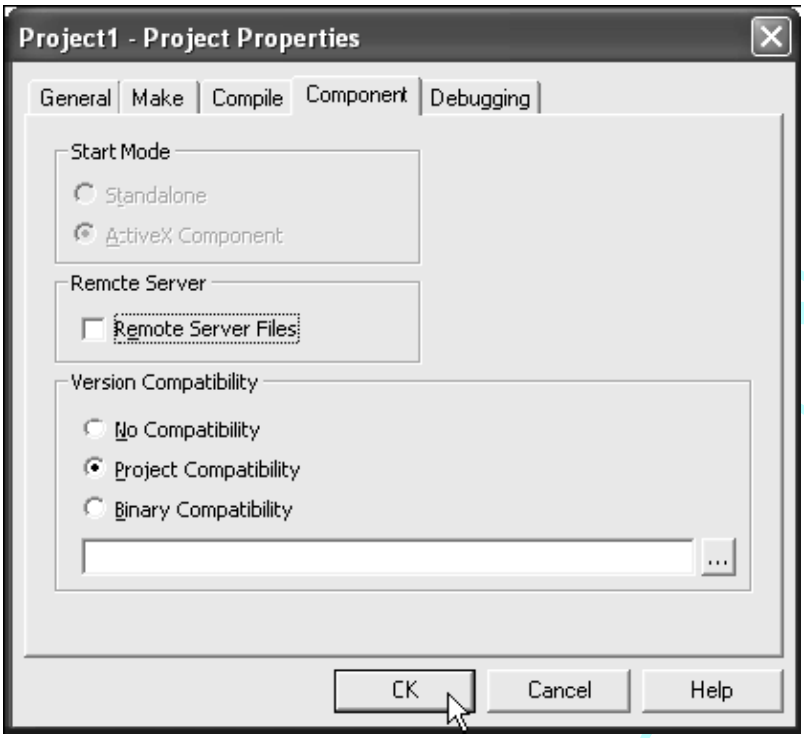
建立新的“标准 EXE”工程时，缺省建立了一个新的窗体。而建立新的“ActiveX DLL”时，缺省建立了一个新的类模块。在建立 ActiveX DLL 时，工程名(在工程属性窗口中设置)和类模块名极其重要。



当查看 MicroStation 的对象模型时，能看到顶级对象名为 MicroStationDGN，紧跟该对象之下的是 Application 对象。如果要建立类似这样的结构，工程名应当叫做“MicroStationDGN”，类模块名叫做“Application”。因为不想让新的 DLL 对象模型与 MicroStation 的混淆，所以将使用不同的命名约定。

我们的工程名叫做“msvba_WinAPI”，类模块名叫做“System”。





除了在工程属性对话框中命名工程外，还需要将版本兼容性(Version Compatibility)设置为“工程兼容性(Project Compatibility)”。

在叫做“System”的类模块中放置的代码和一般类模块中的代码相似。下面是代码的汇总：

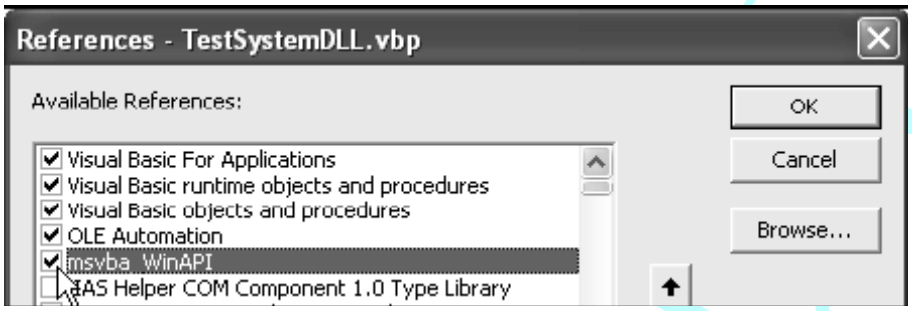
```
Private Declare Function GetSystemMetrics Lib "user32" (ByVal nIndex _
                As Long) As Long
Private Declare Function GetComputerName Lib "kernel32" Alias _
                "GetComputerNameA" (ByVal lpBuffer As String, _
                nSize As Long) As Long
Private Declare Function GetDiskFreeSpace Lib "kernel32" Alias _
                "GetDiskFreeSpaceA" (ByVal lpRootPathName As String, _
                lpSectorsPerCluster As Long, _
                lpBytesPerSector As Long, _
                lpNumberOfFreeClusters As Long, _
                lpTotalNumberOfClusters As Long) As Long
Private Declare Function GetLogicalDriveStrings Lib "kernel32" Alias _
                "GetLogicalDriveStringsA" (ByVal nBufferLength As Long, _
                ByVal lpBuffer As String) As Long
```

```
Private Declare Function LogonUser Lib "Advapi32" Alias "LogonUserA" _
                (ByVal lpszUsername As String, _
                ByVal lpszDomain As String, _
                ByVal lpszPassword As String, _
                ByVal dwLogonType As Long, _
                ByVal dwLogonProvider As Long, _
                phToken As Long) As Long
Private Declare Function ShellExecute Lib "shell32.dll" Alias _
                "ShellExecuteA" _
                (ByVal hwnd As Long, ByVal lpOperation As String, _
                ByVal lpFile As String, ByVal lpParameters As String, _
                ByVal lpDirectory As String, _
                ByVal nShowCmd As Long) As Long
Const SM_CXSCREEN = 0
Const SM_CYSCREEN = 1
Function Version () As String
    Version = App.Major & "." & App.Minor
End Function
Sub ScreenSize (ByRef x As Long, ByRef y As Long)
    x = GetSystemMetrics(SM_CXSCREEN)
    y = GetSystemMetrics(SM_CYSCREEN)
End Sub
Function ComputerName () As String
    Dim CompName As String
    CompName = Space(255)
    GetComputerName CompName, Len(CompName)
    ComputerName = Left(CompName, InStr(1, CompName, Chr(0)) - 1)
End Function
Function DiskFreeSpace (DriveLetter As String, DiskTotal As Variant) _
                As Variant
```

```
    Dim SectorsPerCluster As Variant
    Dim BytesPerSector As Variant
    Dim FreeClusters As Variant
    Dim TotalClusters As Variant
    GetDiskFreeSpace DriveLetter, SectorsPerCluster, _
                     BytesPerSector, FreeClusters, TotalClusters
    DiskTotal = BytesPerSector * SectorsPerCluster * TotalClusters
    DiskFreeSpace = BytesPerSector * SectorsPerCluster * FreeClusters
End Function
Function LogicalDrives() As Variant
    Dim LDrives() As String
    ReDim LDrives(0) As String
    Dim DriveBuff As String
    Dim XSplit As Variant
    DriveBuff = Space(255)
    GetLogicalDriveStrings Len(DriveBuff), DriveBuff
    XSplit = Split(DriveBuff, Chr(0))
    ReDim LDrives(0 To UBound(XSplit) - 2)
    For I = LBound(LDrives) To UBound(LDrives)
        LDrives(I) = XSplit(I)
    Next I
    LogicalDrives = LDrives
End Function
Function CheckLogon (UserName As String, Password As String) _
                     As Boolean
    CheckLogon = LogonUser(UserName, "", Password, 2, 0, 0)
End Function
Sub WinOpenFile (FileName As String)
    ShellExecute 0, "OPEN", FileName, "", "", 0
End Sub
```

China BDN

测试 ActiveX DLL 与测试 ActiveX 控件类似。在当前工程中添加一个新工程(是添加工程，不是新建工程)建立工程组。然后在新的“标准 EXE”工程中添加对刚建完的 ActiveX DLL 的引用：



一旦添加了引用，就能像本书许多其它引用一样来使用。在新的测试 EXE 工程的窗体中添加一个命令按钮，在该按钮的 Click(单击)事件中输入如下代码：

```
Private Sub Command1_Click()
    Dim myMsVBA As New msvba_WinAPI.System
    Dim XS As Long
    Dim YS As Long
    Dim XDrives As Variant
    Dim I As Long
    XDrives = myMsVBA.LogicalDrives
    MsgBox Join(XDrives, ", "), , "Drives On System"
    For I = LBound(XDrives) To UBound(XDrives)
        MsgBox XDrives(I) & vbCr & _
               myMsVBA.DiskFreeSpace(CStr(XDrives(I)), 0), , _
               "Drive Free Space"
    Next I
    MsgBox myMsVBA.ComputerName, , "Computer Name"
    myMsVBA.ScreenSize XS, YS
    MsgBox XS & ", " & YS, , "Screen Resolution"
    MsgBox myMsVBA.CheckLogon("Administrator", "adminpwd"), , _
           "LOGON"
    myMsVBA.WinOpenFile "c:\abcdefg.pdf"
```

End Sub

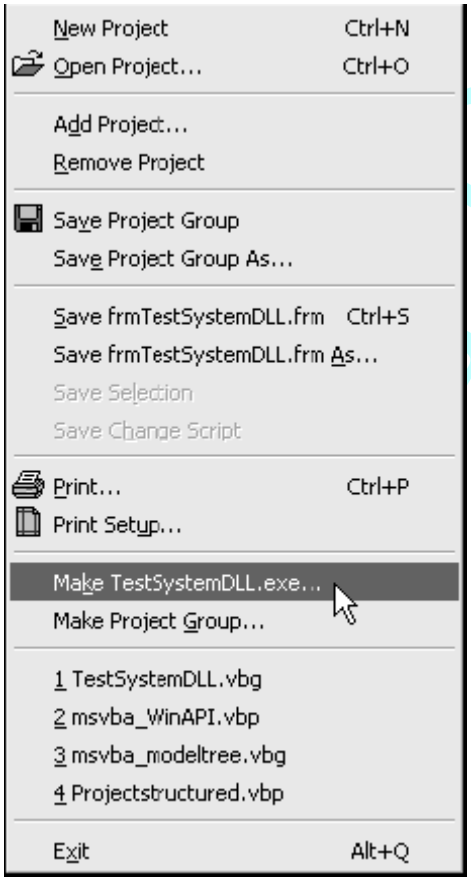
我们以前讨论过能在类模块中放置代码并在自己的代码中使用它们。当把类编译成 ActiveX DLL 后，使得代码很容易使用。多个 VB 和 VBA 工程都能“连接”到我们的 ActiveX DLL。

与 ActiveX 控件一样，该 DLL 文件在非开发该 DLL 的其它计算机上使用前必须用 Regsvr32.exe 注册。在 VB6 中编译该 DLL 文件时，VB6 自动将其注册到本机中。

编译和分发应用程序

在 MicroStation VBA 工程(mvba 文件)中编写代码后，只需简单地将工程文件交给用户即可。在大多数情况下，如果工程中未用自定义 ActiveX 控件的话，程序能很好地运行。但在 VB6 中编写代码时，需要编译应用程序，编译完应用程序后通常还需要做一些工作。让我们先讨论编译 VB6 程序，然后再讨论将 VB6 编写的应用程序分发给用户。

编译应用程序



要在 VB6 中编译一个应用时，请选择 **File > Make...**。“Make”一词后所看到的文字决定于正在使用的工程名和工程类型。除了看到“Make”和一个工程名外，还能看到“Make Project Group(制作工程组)”。Make Project Group 用于编译当前组中的所有应用程序。

编译应用非常简单，我们能在此看到菜单项。但什么时候应该编译呢？

在软件分发给用户之前，都有可能不编译而建立整个应用。对于 ActiveX 控件和 ActiveX DLL 这种情况，我们可以将工程添加到工程组中而不编译来测试控件或 DLL。然而，如果想在另外的像 MicroStation VBA 环

境这样的应用程序中测试控件或 DLL，就需要编译它了。当控件或 DLL 位于标准 EXE 工程中时，为了测试性能在开发过程中进行编译是有用的。VB6 应用在设计模式(能中断应用、单步调试等)下比其在编译后运行得要慢很多。缺省情况下，编译 VB6 应用能让编译器对代码进行非常全面的检查。编译一个工程时，在设计模式下运行时没有指出的、位于模块、窗体、过程、函数中的错误都将暴露出来。

编译 VB6 工程通常生成一个.exe 文件(标准 EXE)、.ocx (ActiveX 控件)或.dll(ActiveX DLL)文件。将这一文件电邮或分发给世界上的任何人，我们的程序似乎能在用户的计算机上运行就像在我们的计算机上运行一样。有时是这种情况，但并不总是如此。

分发 VB6 应用程序

当谈论分发 VB6 应用程序时，不仅意味着把文件放入另外一台计算机中，也意味着注册控件和 DLL 以及其它一些“设置”计算机的工作。

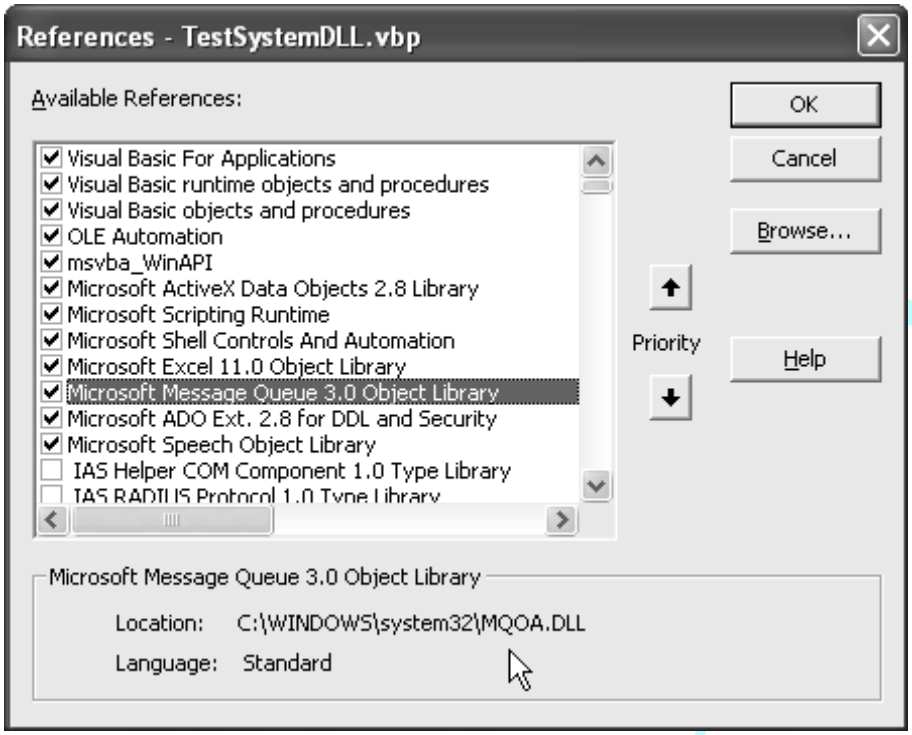
标准 EXE 应用

分发用 VB6 编写的一个标准 EXE 应用时，至少有两个文件应当被分发：EXE 文件本身和 VB6 虚拟机文件(msvbvm60.dll)。缺少 VB6 虚拟机文件，VB6 应用程序将不能运行。VB6 虚拟机 DLL 在大多数计算机中已经安装，但不能保证它在每台要安装我们软件的计算机上都安装了，因此它应当随应用一起分发。

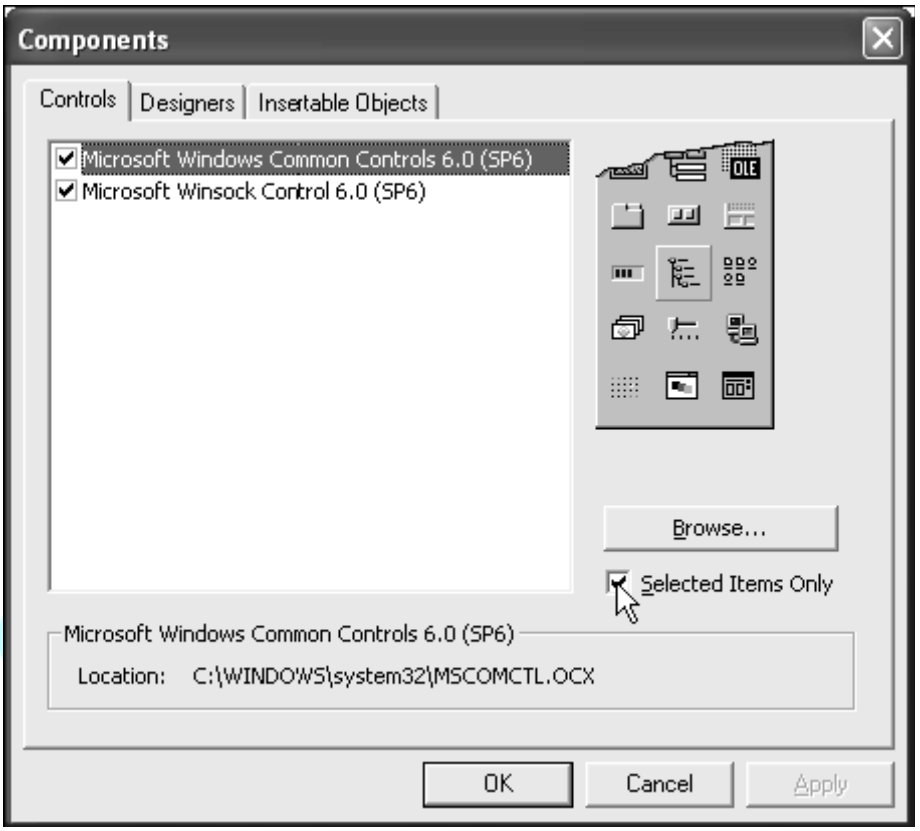
除了这两个基本文件外，还必须作出其它考虑。例如，假定我们建立一个利用互联网传输控制协议向 FTP 服务器传送文件的应用程序。该互联网传输控件(一个.ocx 文件)随 VB6 一起安装，它非常有用，利用它能建立功能强大的互联网程序。但如果将我们的程序(EXE 文件)给了某些人而他们的计算机上没有这个 ActiveX 控件，我们强大的程序也不能工作，只是显示错误消息框而已。

当要查找应用程序所依赖的文件时，有两个地方需要查看。第一个地方是引用对话框。

在下图中我们能看到大量引用。所显示的前四个引用对任何 VB6 程序都是标准的，不需要担心它们。被选择的其它那些引用应当给以关注。在引用列表中选择一项时，该项的完整路径在“Location(位置)”标签后显示。



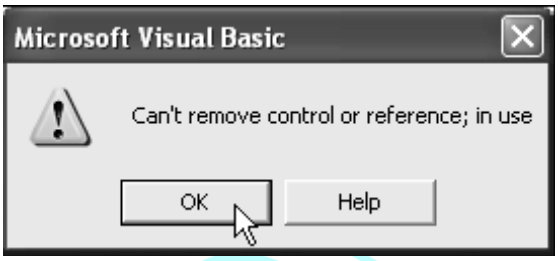
另一个应当查看的地方是工具箱。例如，如果我们使用了一个树视图控件，就需要确保包含树视图控件的文件也被分发。



右击工具箱并选择“Components(组件)”打开组件对话框。当在对话框中选择“Selected Items Only(仅为选择的条目)”时，只有那些已经被“选择”的

控件才显示出来。一个控件被选择并不意味着将它插入我们的工程。如果控件未被插入，它就不需要随应用程序一起分发。这个观点对吗？

如果想试图“取消选择”插入的一项引用，将看到这个消息框：

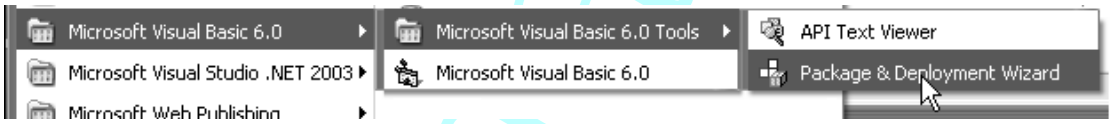


任何“未选择”的文件实际上都没有在我们的程序中使用，所以也就不需要分发它们。

好了，从引用和控件区能知道需要哪些文件。还知道需要在宿主计算机上注册 DLL 和 ActiveX。怎样才能最好地完成这些工作呢？

我们已经使用过了 RegSvr32.exe 注册过程。我们可以在一个批处理文件(.bat 文件)中放入一系列的注册命令，然后将这些源文件和注册批处理文件压缩成一个 zip 文件。虽然这个解决方案能工作，但离完美和用户友好相差甚远。

让我们考察一下随 VB6 带来的解决方案：Microsoft Package and Deployment Wizard(微软打包部署向导)。从“Microsoft Visual Basic 6.0”菜单开始能找到它。

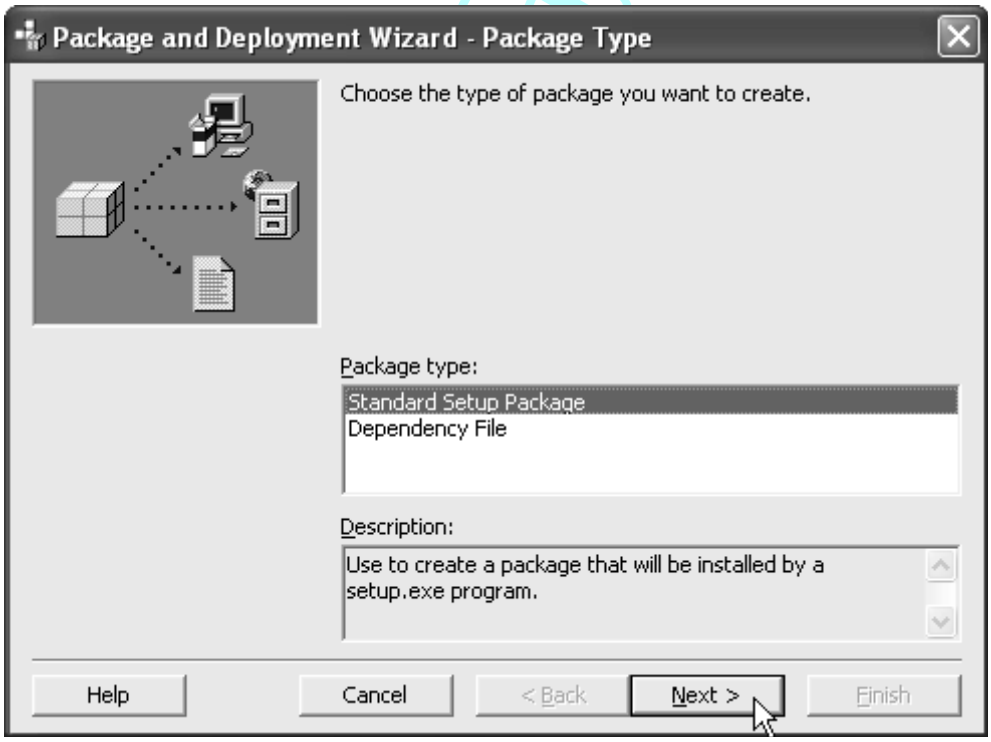


在运行该打包部署向导前，最好关闭 VB6。

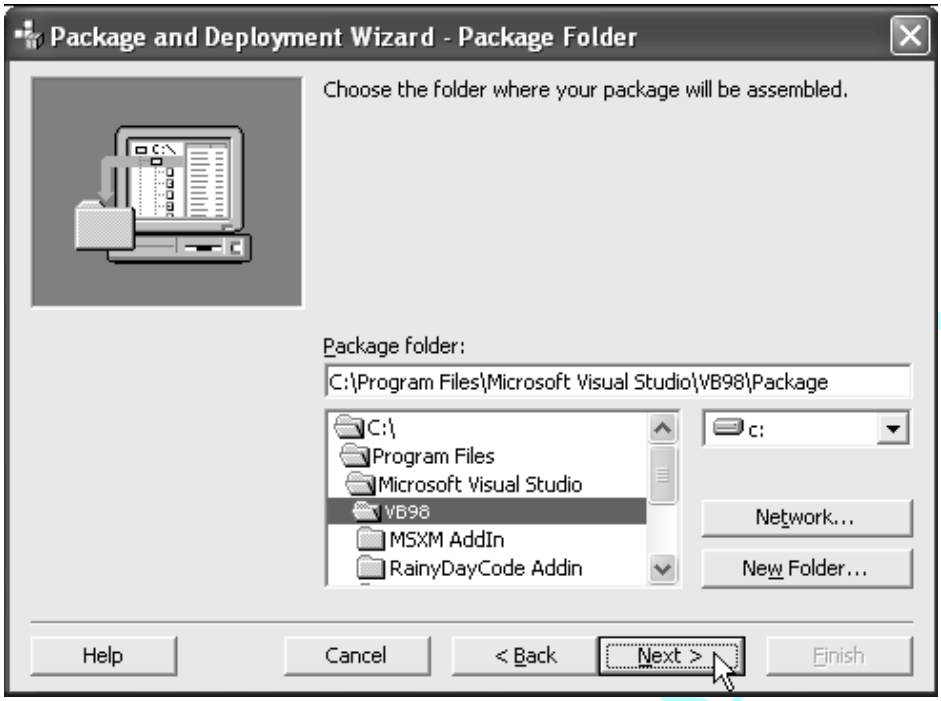
1、使用打包部署向导的第一步是选择要分发的 Visual Basic 工程(.vbp)文件。然后点击 Package(打包)按钮；



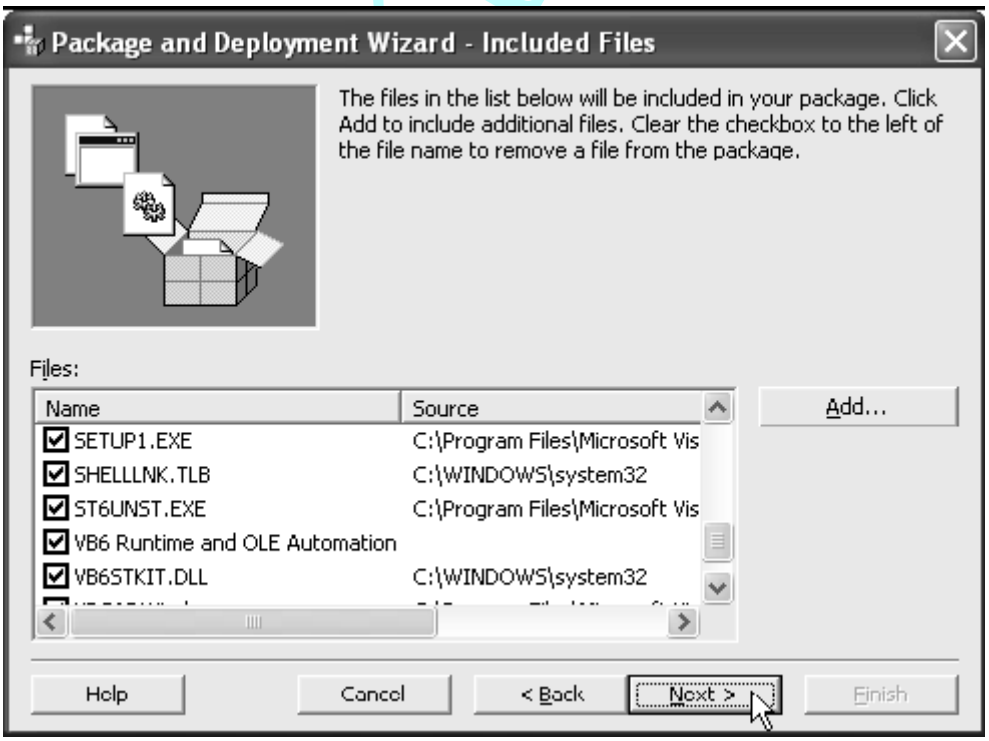
2、出现下一个对话框，需要在“Standard Setup Package(标准设置打包)”和“Dependency File(依赖文件)”中选择。我们选择 Standard Setup Package；



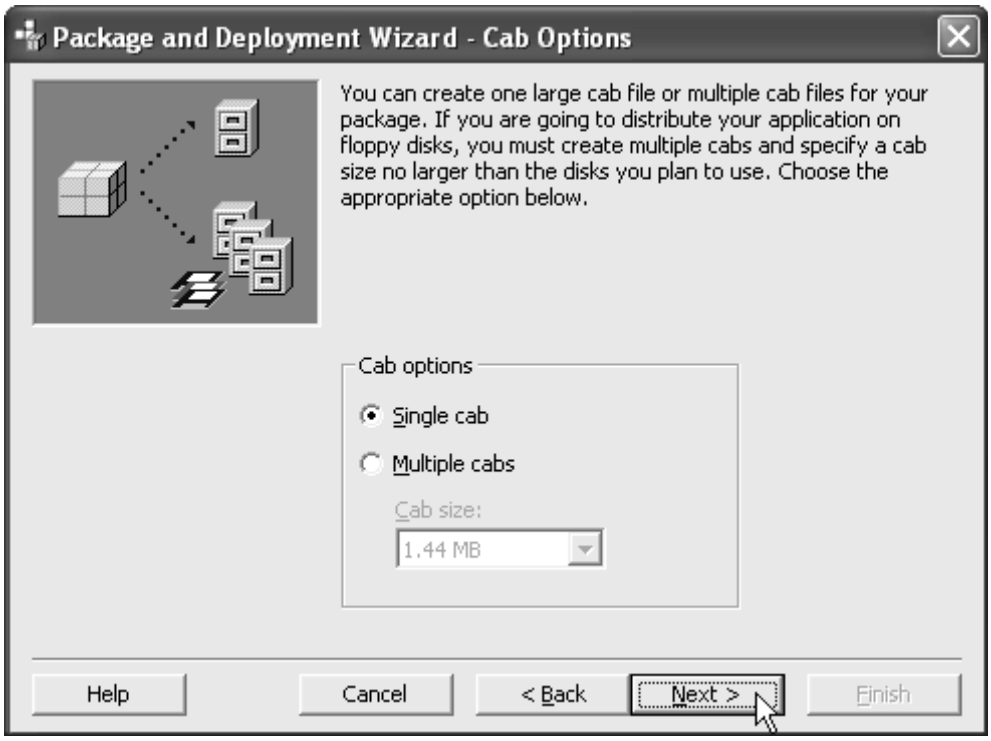
3、下一步提示选择一个建立 Setup.exe 文件的文件夹；



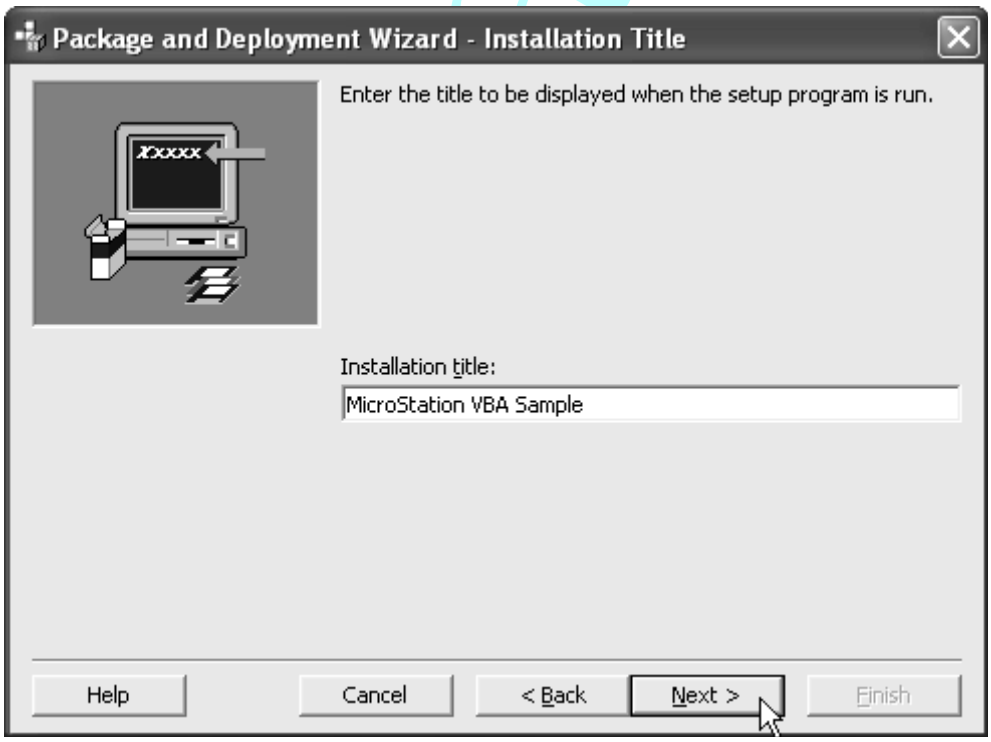
4、选完文件夹后，向导提问要在 Setup 文件中包含哪些文件。这一步用来确定需要随应用程序分发的文件。所以，如果使用打包部署向导，我们不需要担心所添加的引用和控件。这一步还很重要，因为能向 Setup 包中添加可能不是自动加入的那些文件。例如，如果我们编写了一本 PDF 格式的用户手册，就可以在这一步将手册文件添加进来，它将与其它工程文件一起被压缩并“安装”到用户的计算机中；



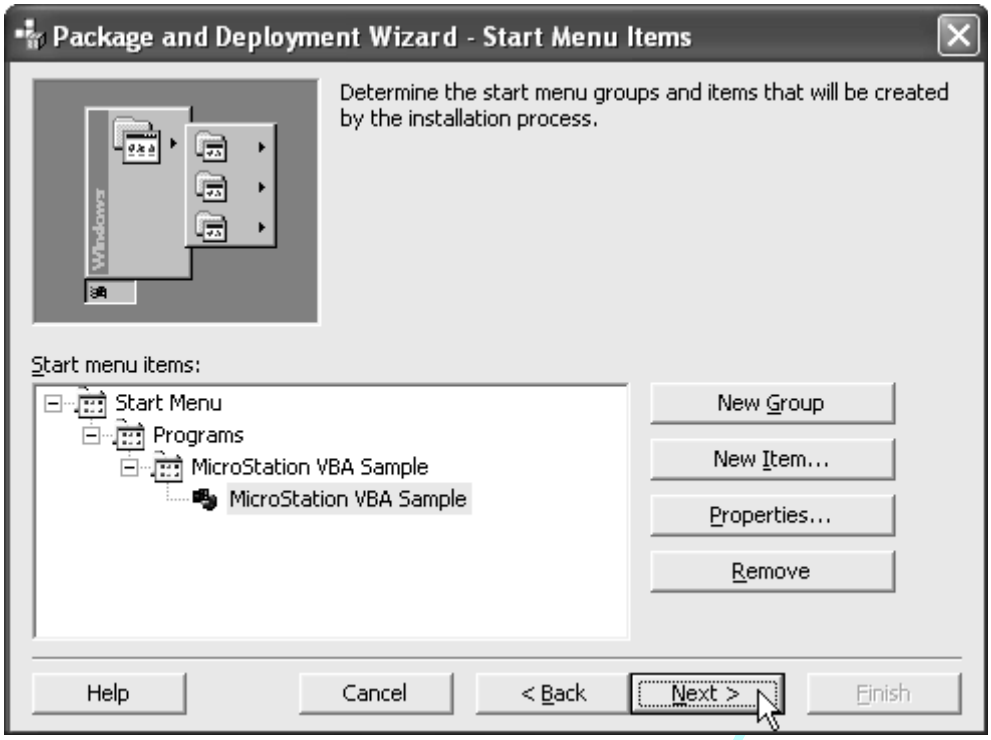
5、验证完 Setup 文件中要包含的文件后，向导提示我们是否要将 Setup 作为单个文件分发或放在多个软盘(虽已过时但仍然是一个可选项)上分发；



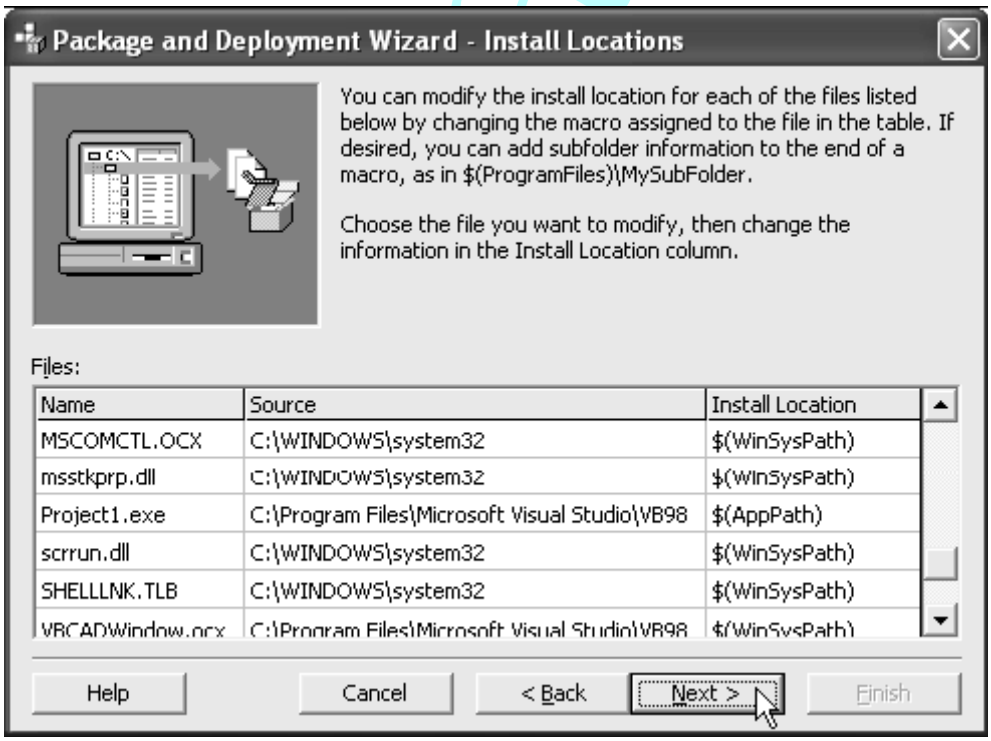
6、下一步是为 Setup 包指定标题：



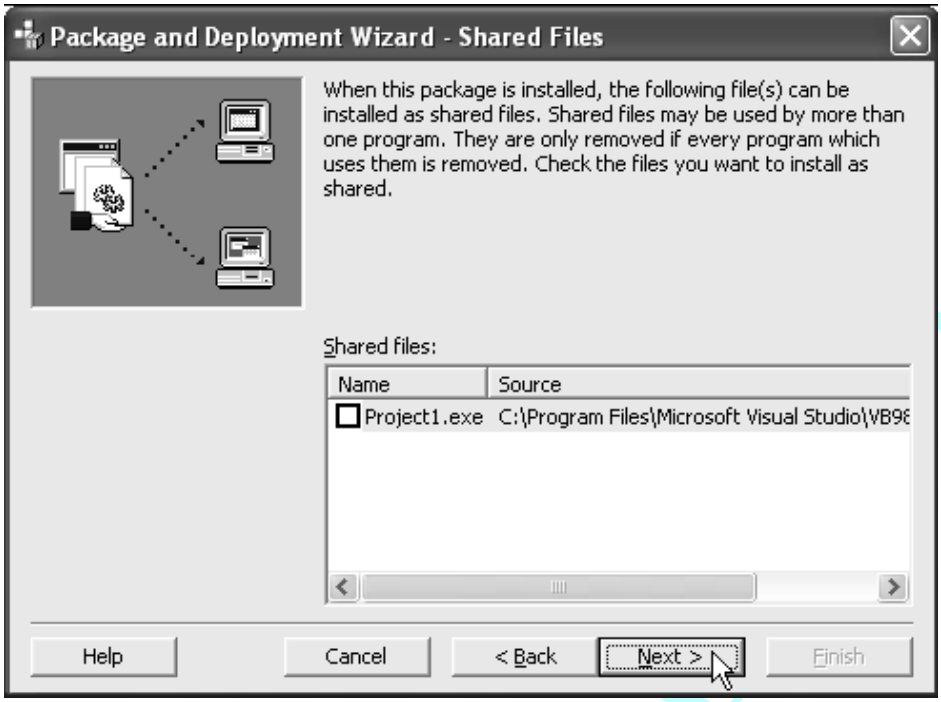
7、可以在开发菜单或其它地方放置快捷方式；



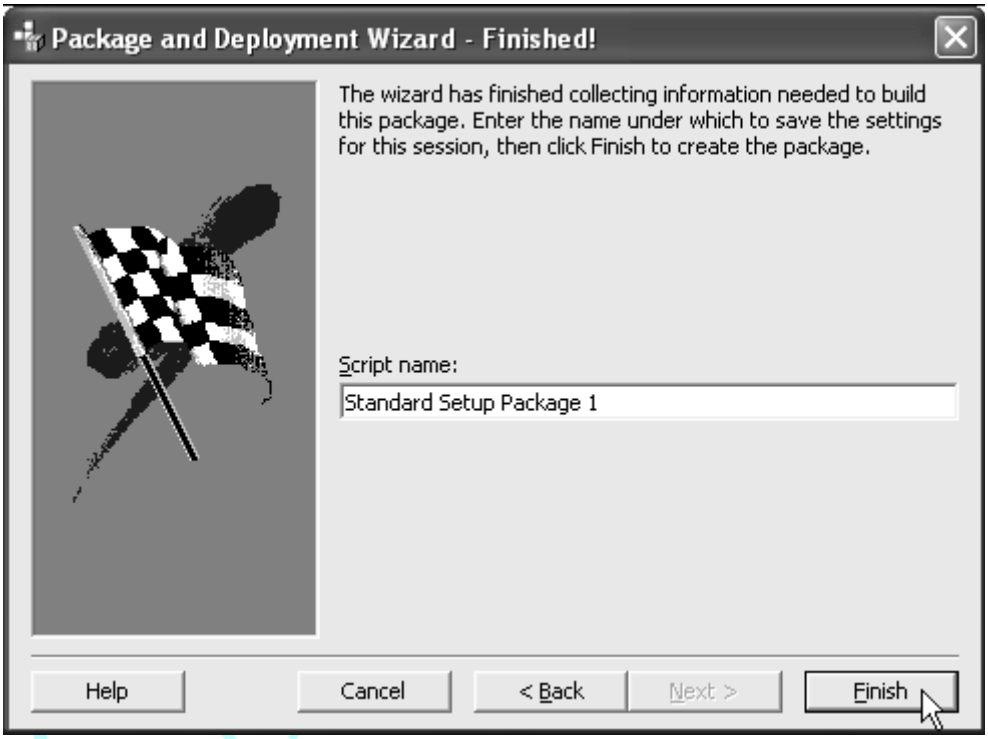
8、包含在 Setup 包中的文件必须放入用户计算机的某个地方。缺省情况下，DLL 和 OCX 文件放入 System32 文件夹。然而，在下个对话框中可以改变这些文件的位置：



9、不选择工程可执行文件就不会将工程测试用可执行文件指定为“共享文件”，这样可以继续进入到下个对话框；



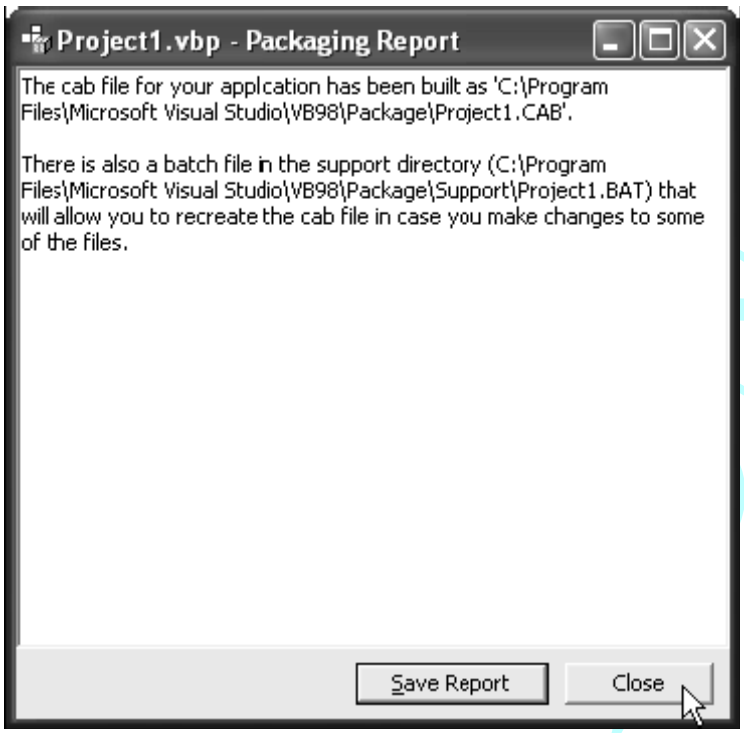
10、要处理的最后一个对话框能让我们指定脚本名。

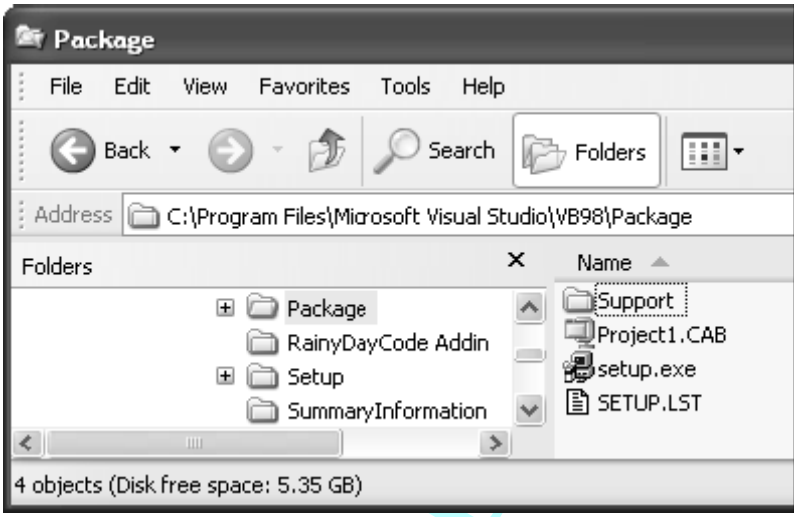


点击 Finish(完成)按钮后建立了 Setup 文件。



Setup 程序完成后生成了一个报告，我们可以保存报告或关闭向导。





我们已经完成了打包部署向导。还需要说明什么？在打包过程早期，向导要求我们指定放置部署文件的位置。现在浏览到选择的那个位置，我们能看到由向导建立的文件。

向导为我们建立了一个 **Project1.CAB** 文件、一个 **setup.exe** 文件和一个 **SETUP.LST** 文件。.CAB 文件包含要分发的所有文件，它们(所有这些文件)被压缩到单个文件中。.LST 文件指定每个文件存放的位置，建立完 CAB 和 setup.exe 文件后也能修改该文件。对于 setup.exe 文件，其唯一目的就是提取.CAB 中的文件并将这些文件放入.LST 文件指定的文件夹中，它也按.LST 文件中的指令注册组件。Support 文件夹包含未压缩状态的所有文件。

微软打包部署向导在打包 VB6 开发的程序方面做得相当好。当然，还有其它产品也能建立安装程序，其中之一就是 Macrovision 公司的 InstallShield

Express。

本章回顾

VB6 具有一个看起来和感觉起来都很像 VBA 的环境。VB6 除了能建立独立运行的程序外，还能建立我们自己的 ActiveX 控件和 ActiveX DLL。当完成了程序编制和编译后，可以用随 VB6 带来的微软打包部署向导分发我们的应用。

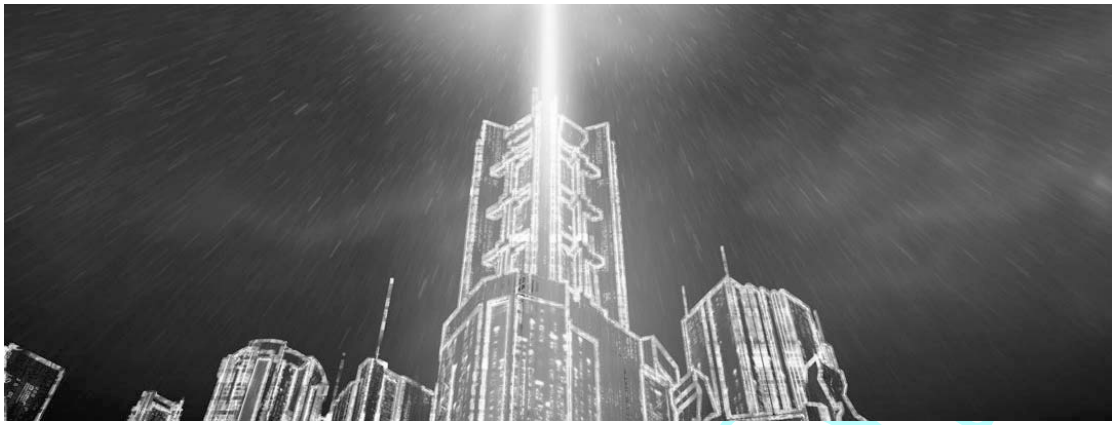
一旦“连接”到 MicroStation，在 VB6 中开发与在 MicroStation 的 VBA 环境中开发几乎一样。在 MicroStation VBA 中建立和使用的所有代码都能毫不费力地“移植”到 VB6 中。

在把 VBA 程序“移植”到 VB6 时，需要考虑一件事情：在 VBA 中工作时，有几个对象本身就公开了，如 Application 对象。在 MicroStation VBA 环境中可以键入

Application.ActiveDesignFile . . . . .

这行代码就能运行。但在 VB6 中工作时，每个 MicroStation 对象都必须明确设置。然而，如果我们在代码模块中将一个代表 MicroStation 应用的变量声明为 Public，那么只需要对该变量设置一次，随后就能不需“设置”而使用它。

> **注意：**Microsoft Visual Studio 6 已经被 Visual Studio .NET 2003 和 Visual Studio .NET 2005 取代。.NET 家族的开发产品能生成独立的.EXE 文件，也能创建 COM 可存取的 DLL 文件。然而，.NET 不能用来建立 ActiveX 控件。所以，如果 MicroStation VBA 环境需要一个新控件时，还是应该在 VB6 中建立。



39. 使用 VB.NET

从表面上看，VBA、VB6 和 VB.NET 好像都一样，它们的名字中每个都含有“Visual Basic”部分并使用相同的“Visual Basic”基础。是的，它们非常相像，但也有不同点。相比较于 VB.NET 来说，VBA 和 VB6 有更多的共同点，它们(VBA 和 VB6)与 VB.NET 之间有很大的不同。微软在 2003 年引入 VB.NET 环境，目的之一是使 Visual Basic 更加面向对象。这一点是 VB.NET 与 VBA 或 VB6 的根本不同点。

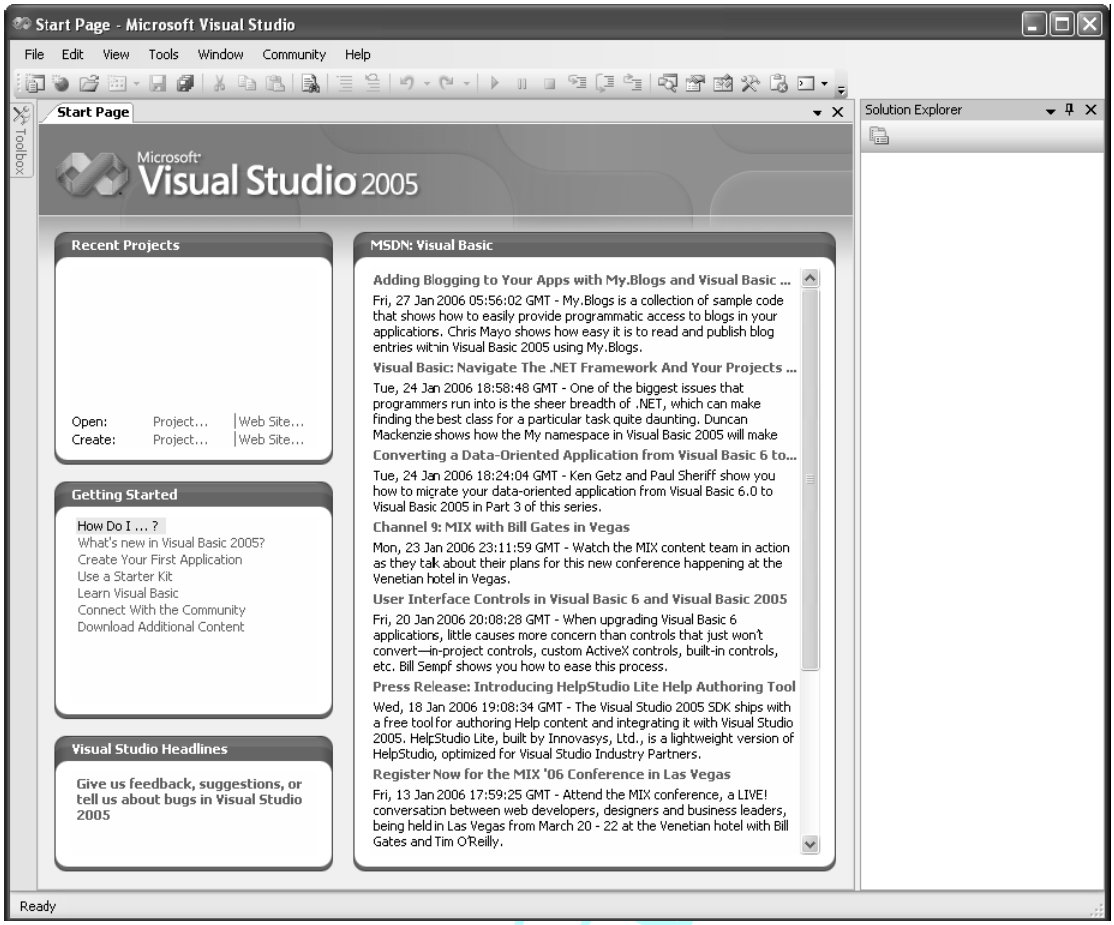
本章不仅涉及在 VB.NET 应用中如何控制 MicroStation，而且还关注这些环境之间的不同，这样能使我们对 VB.NET 环境和语言更加熟悉。我们将使用 Visual Studio 2005 来作讨论。

本章包括：

- 对 VB.NET 环境的介绍
- 在 VB.NET 可以完成的工作
- VBA/VB.NET 交叉引用
- 分发 VB.NET 应用
- MicroStation 中 VB.NET 的未来

VB.NET 介绍

让我们看一下 VB.NET 环境。应当说明的第一件事是，尽管 VB6 是一个应用，VC++是其应用本身，但.NET 环境用于在 VB、C#(读作 C Sharp)、J#等中开发应用。

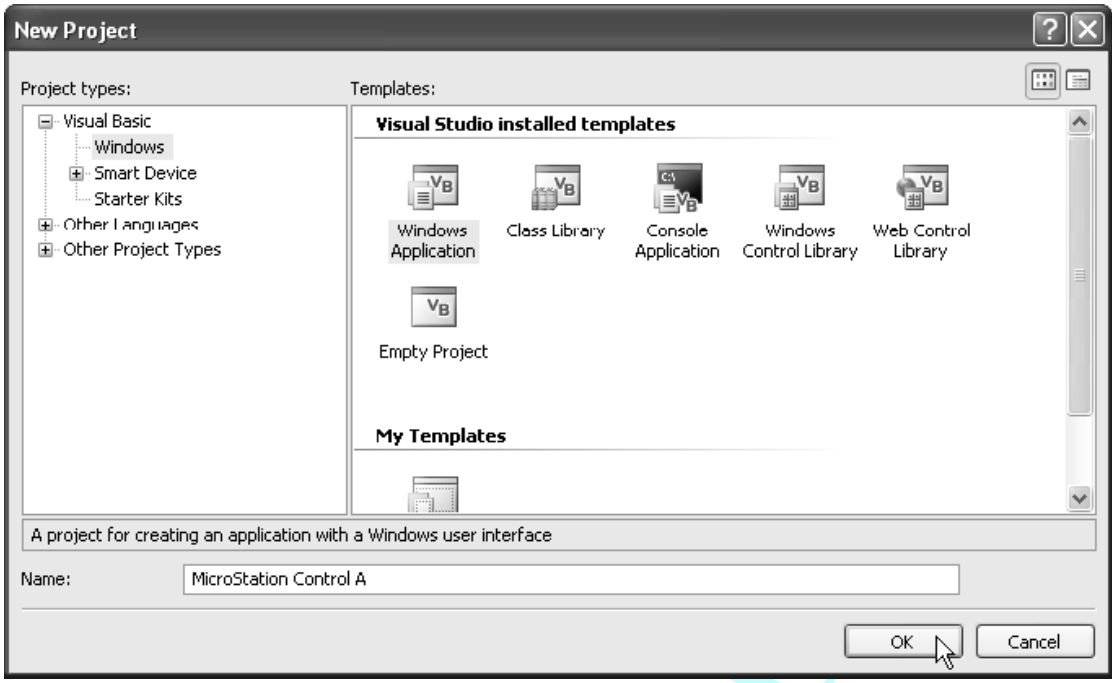


（Visual Studio 2005 环境启动页面）

从 Visual Studio 2005 启动页面中能建立或打开新工程。除了建立新工程和打开现有工程外，还能看到来自微软 MSDN Visual Basic 主页的 RSS 传播(在早先的一章中讨论过 RSS 技术)。

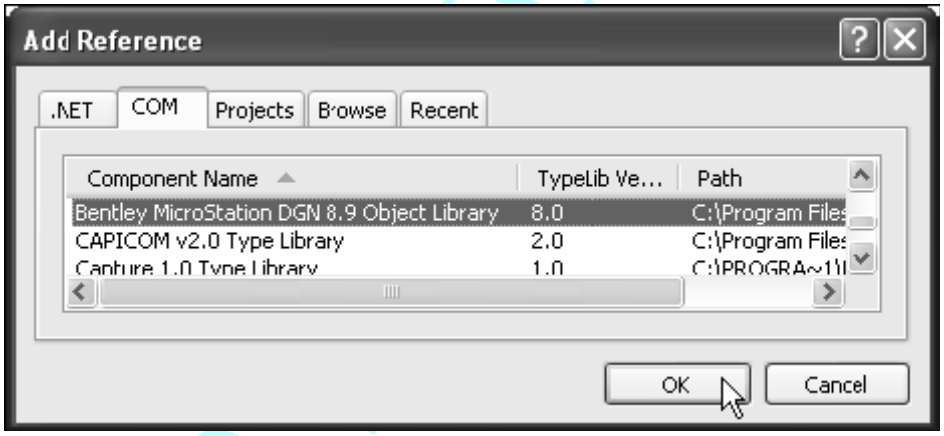
建立一个新工程

我们将建立一个叫做“MicroStation Control A”的新工程，它是一个 Windows 应用。与 VB6 一样，新建的工程会带有一个添加的窗体。



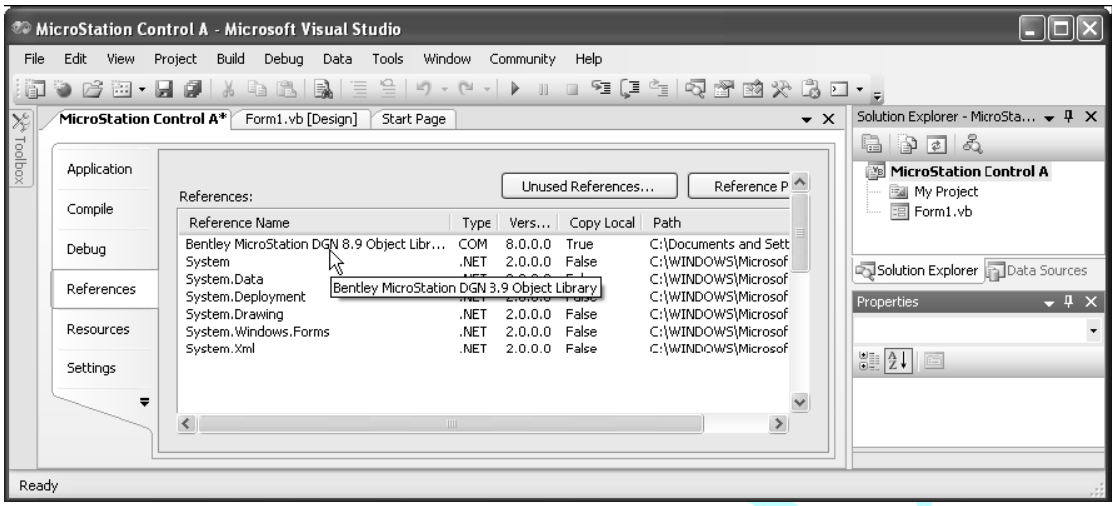
我们知道这个工程要用于控制 MicroStation。下一步是什么？当然是添加一个引用。

1、从 Project(工程)菜单中选择 Add Reference(添加引用)；



2、选择 COM 选项卡并向下滚动到“Bentley MicroStation DGN #.# Object Library”。点击 OK 按钮添加引用；

3、双击“Solution Explorer(解决方法管理器)”中我们的工程名，然后选择 Reference(引用)选项卡来显示当前工程中的所有引用；

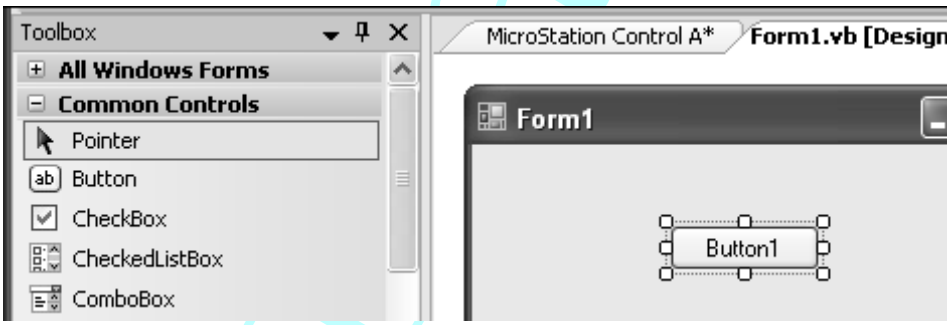


从这个窗口中能看到一个引用的命名系统，一个是 System.Data，另一个是 System.Drawing 还有其它的，它们的类型是“.NET”。我们能看到已经被添加的 Bentley MicroStation 引用，其类型是“COM”。

现在来编写一些代码。

4、切换到 Form1.vb 选项卡，通过点击工具箱顶部的图钉图标来固定工具箱；

5、从工具箱中拖拽一个按钮到窗体上；



6、接下来在该按钮上双击进入按钮的 Click(单击)事件；

7、既然进入了按钮的 Click 事件，就输入一些非常简单的代码来“连接”到 MicroStation 并在一个消息框中显示应用的标题。下面是含有这个基本功能的全部代码列表。

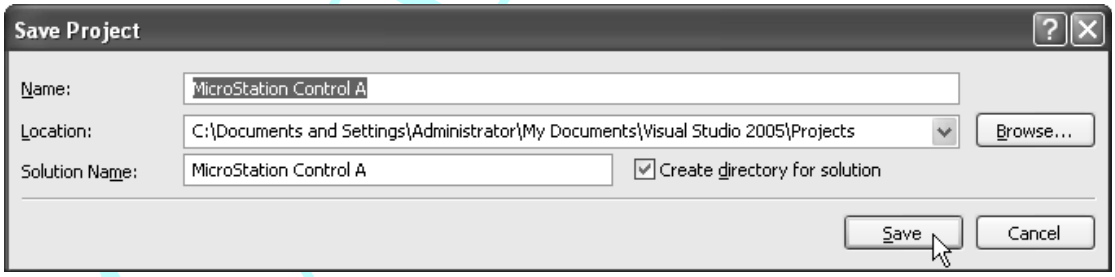
```
Public Class Form1
    Private Sub Button1_Click( _
                ByVal sender As System.Object, _
                ByVal e As System.EventArgs) Handles Button1.Click
        Dim myMSAppCon As MicroStationDGN.ApplicationObjectConnector
```

```
        Dim myMSApp As MicroStationDGN.Application
        myMSAppCon = GetObject(, _
                              "MicroStationDGN.ApplicationObjectConnector")
        myMSApp = myMSAppCon.Application
        MsgBox(myMSApp.Caption)
        myMSApp = Nothing
        myMSAppCon = Nothing
    End Sub
End Class
```

这段代码看起来非常熟悉。它很像在 VB6 中建立的代码，也很像为 Microsoft Excel 创建的代码。在这里看到的代码与以前使用的代码主要不同点之一是，在使用对象时缺少“Set”语句。为什么是这样？因为在 VB.NET 中所有变量都是对象。String 是对象，Integer 是对象。所以，在给变量赋值或赋对象时不需要使用“Set”。

如果仔细看消息框语句能看到在 VB.NET 中使用了圆括号，在 VB6 中除了要返回值以外都不这样用。任何时候在 VB.NET 中使用带有参数的函数或过程时，都要在参数两侧加上圆括号；

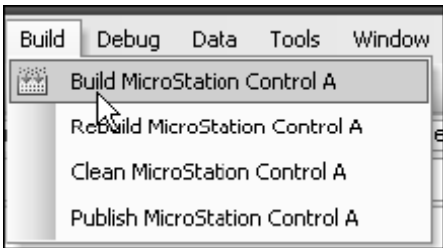
8、如果运行代码，将发现这段代码运行良好。但在做任何事之前应当先保存工程。选择 **File > Save All(文件 > 保存所有)**显示 Save Project(保存工程)对话框。



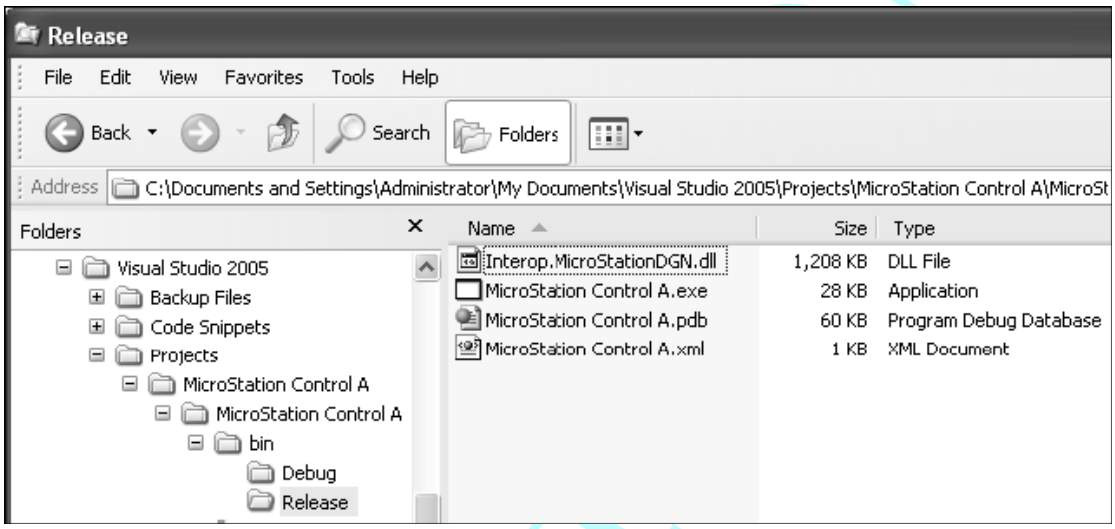
使用如上所示的设置，一个新的叫做“MicroStation Control A”的目录被建立。

编译应用

1、选择菜单 **Build > Build MicroStation Control A** 开始编译我们的应用。



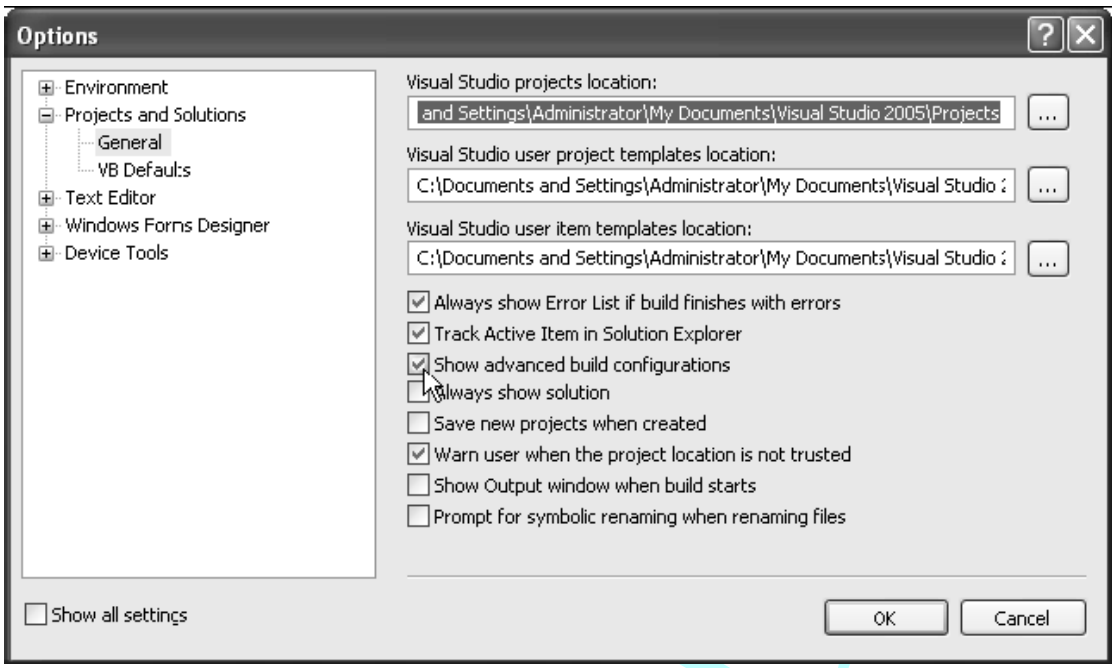
“Build(生成)”应用能生成一个.exe 文件。看一眼下面的图形，然后我们来讨论它。



在 VB.NET 中编译应用时，缺省情况下它们被编译到两个文件夹中的一个：Debug 或 Release。“Debug(调试)”和“Relase(发布)”是在开发和编译应用时可用的编译器配置。这些配置具有属于调试、性能选项和处理器预选项方面的不同设置。在“配置管理器”(**Build > Configuration Manager**)中指定要使用的配置。

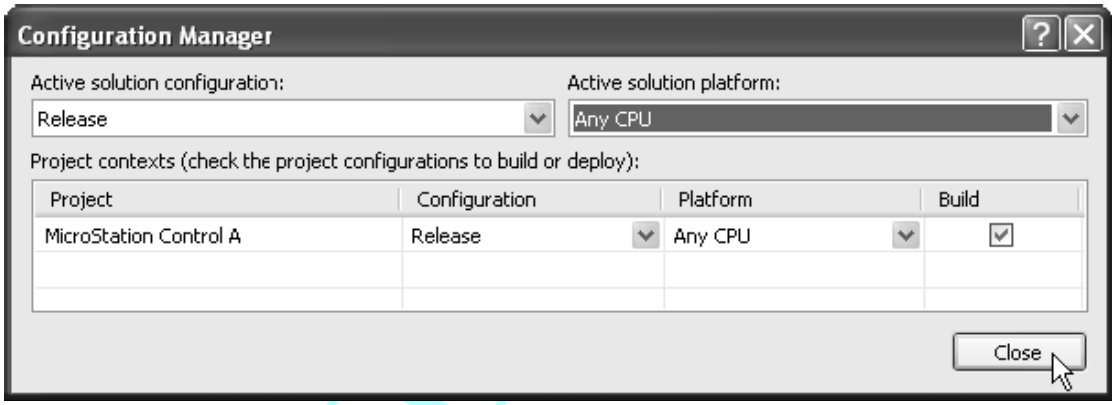
> **注意：**如果在 VB.NET 中要去调试一个应用但不能用**<F8>**进入单步状态，那就要查看配置管理器以检查当前的配置设置。碰巧可能选择了发布配置(这种配置取消许多 VB.NET 的调试能力)而想去调试应用程序。

2、如果配置管理器菜单项没有出现在菜单中，可以在 **Tools > Options(工具 > 选项)**中设置选项来打开它。



3、在 Build(生成)菜单组中选择“Show advanced build configurations(显示高级建立配置)”来打开配置管理器。

现在来看一看配置管理器。



Build(生成)应用时，结果文件被放入当前配置指定的文件夹中。缺省情况下，“Release”编译配置把文件编译到 Release 文件夹，“Debug”编译配置把文件编译到 Debug 文件夹。我们使用配置管理器来指定当前配置，建立应用时，编译是基于当前配置的。

以上这些内容是对 Microsoft Visual Studio 2005 中 VB.NET 环境的主要介绍。

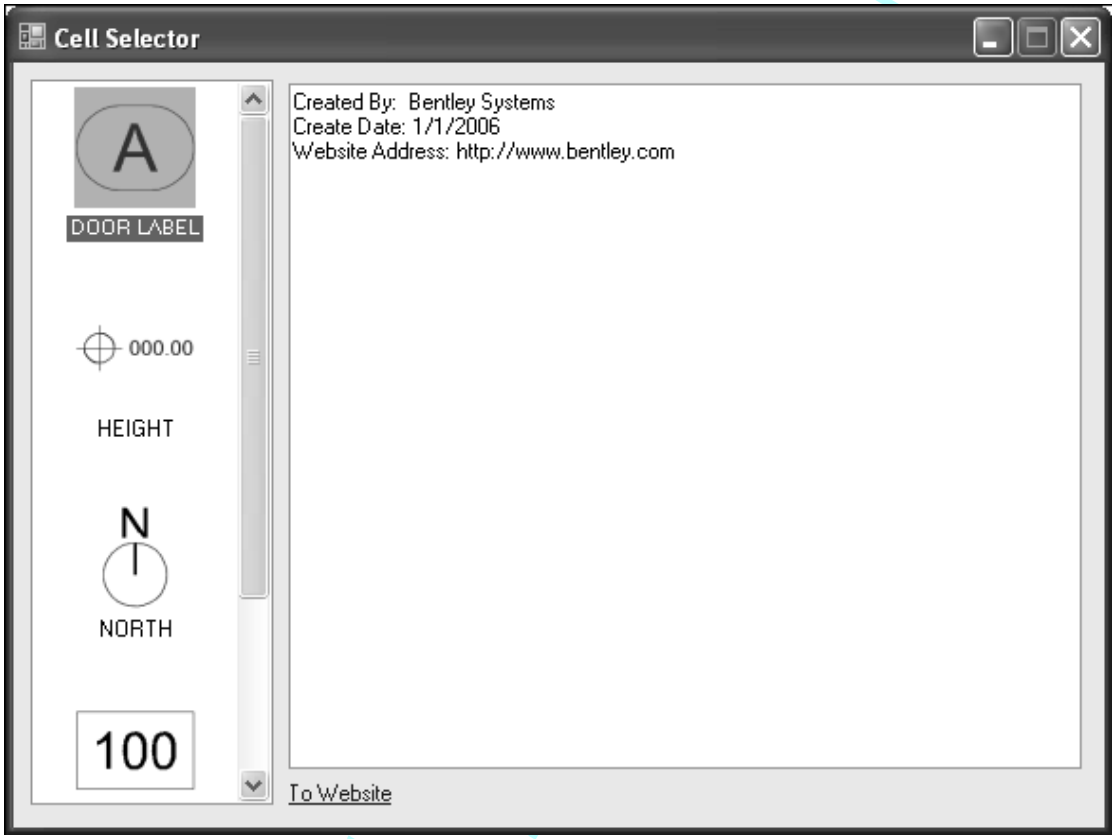
在 VB.NET 可以完成的工作

我们刚刚在 VB.NET 中创建了一个非常简单小巧的应用程序。在 VB.NET 的 IDE(集成开发环境)与 VBA 和 VB6 的 IDE 之间存在一些不同。在此之前我们

学习了如何使用 VB.NET 的 IDE，现在开始查看在 VB.NET 中能做什么。如果多探究一点的话，就会发现几个在 VBA 中不存在的“工具”。让我们创建几个利用 VB.NET 特性功能的应用程序。

下个将要建立的工程是名字为“MicroStation Control B”的 Windows 应用工程。源文件在随书附赠的 CD 上。

让我们看一下运行中的应用程序。先讨论功能后看代码。



该应用程序的概念相当简单。我们看到了一个能被插入到 MicroStation 中的单元列表，每个单元以伴有单元名的略图显示。在窗口右侧的文本框中还能看到被选单元的信息，这一附加“信息”存储在与预览图像同名的.info 文件中。在.info 文件中如果有以“Website Address:”文本开头的行，则就在窗口中显示一个“To Website”超链接。当用户点击该链接后，程序就打开一个新的网页浏览器窗口并打开在.info 文件中的网页地址。假如.info 文件中没有网页地址或.info 文件无效，就不显示“To Website”链接。当用户在左侧的列表视图控件中双击一项时，被双击的单元就被插入到 MicroStation 中。为简化起见，我们将单元插入到当前视图的中心处。

我们不是硬编码单元名并强迫用户建立特定大小的略图图像，而是基于应用

文件夹中可用图像(.bmp)文件完整列表的内容。每个图像文件名对应于一个单元名，信息文件(.info)匹配图像文件名并含有关于单元的参考信息。略图基于每个图像文件被自动地建立于内存之中为列表视图所用。每个动态建立的略图为 64 点阵宽和 64 点阵高，源图像文件可以是任意大小但应当是正方形的，因为被建立的略图图像不在高宽比上作不同的调整。

下面是本工程的代码。请记住，我们已经添加了一个指向 Bentley MicroStation DGN #.# Object Library.的引用。

```
Public Class Form1
    Public ExePath As String
    Public FixedHeight As Long
    Private Sub Form1_Load(ByVal sender As System.Object, _
                ByVal e As System.EventArgs) Handles MyBase.Load
        Dim myFile As System.IO.FileInfo
        Dim myFolder As New _
                System.IO.DirectoryInfo(Application.ExecutablePath)
        Dim myLstVI As ListViewItem
        Dim myImg As Image
        Dim myThumb As Image
        ExePath = myFolder.Parent.FullName
        For Each myFile In myFolder.Parent.GetFiles("*.bmp")
            myImg = Image.FromFile(myFile.FullName)
            myThumb = myImg.GetThumbnailImage( _
                        ImageList1.ImageSize.Width, _
                        ImageList1.ImageSize.Height, _
                        Nothing, Nothing)
            myThumb.Tag = myFile.Name.ToUpper.Replace(".BMP", "")
            ImageList1.Images.Add( _
                    myFile.Name.Replace(".bmp", ""), myThumb
            myLstVI = lstvCells.Items.Add(myThumb.Tag, _
                                ImageList1.Images.Count - 1)
```


China BDN

```
        Next
    End Sub
    Private Sub lstvCells_ItemSelectionChanged(ByVal _
      sender As Object, ByVal e As _
      System.Windows.Forms.ListViewItemSelectionChangedEventArgs) _
      Handles lstvCells.ItemSelectionChanged
        Dim myFileInfo As IO.FileInfo
        Dim myFileReader As IO.StreamReader
        TextBox1.Text = ""
        LinkLabel1.Visible = False
        myFileInfo = New IO.FileInfo(ExePath & "\" & e.Item.Text & ".info")
        If myFileInfo.Exists = True Then
            myFileReader = myFileInfo.OpenText
            TextBox1.Text = myFileReader.ReadToEnd
            myFileReader.Close()
            If InStr(TextBox1.Text, "Website Address:") > 0 Then
                LinkLabel1.Visible = True
            End If
        End If
    End Sub
    Private Sub lstvCells_DoubleClick(ByVal sender As Object, _
                ByVal e As System.EventArgs) Handles _
                lstvCells.DoubleClick
        Dim myMSAppCon As _
                    MicroStationDGN.ApplicationObjectConnector
        Dim myMSApp As MicroStationDGN.Application
        Dim myCellElem As MicroStationDGN.CellElement
        Dim CellOrigin As MicroStationDGN.Point3d
        ProgressBar1.Visible = True
        ProgressBar1.Value = 10
```


China BDN

```
        myMSAppCon = GetObject(, _
                    "MicroStationDGN.ApplicationObjectConnector")
        ProgressBar1.Value = 20
        myMSApp = myMSAppCon.Application
        ProgressBar1.Value = 40
        CellOrigin = myMSApp.CommandState.LastView.Center
        ProgressBar1.Value = 60
        myCellElem = myMSApp.CreateCellElement3( _
                    lstvCells.SelectedItems(0).Text, CellOrigin, True)
        ProgressBar1.Value = 80
        myMSApp.ActiveModelReference.AddElement(myCellElem)
        ProgressBar1.Value = 100
        ProgressBar1.Visible = False
    End Sub
    Private Sub LinkLabel1_LinkClicked(ByVal sender _
            As System.Object, ByVal e As _
            System.Windows.Forms.LinkLabelLinkClickedEventArgs) _
            Handles LinkLabel1.LinkClicked
        Dim xSplit As String()
        Dim strWebsite As String
        xSplit = Split(TextBox1.Text, vbCrLf)
        For Each strWebsite In xSplit
            If strWebsite.StartsWith("Website Address:") = True Then
                Dim a As ProcessStartInfo = New ProcessStartInfo( _
                        Mid(strWebsite, InStr(strWebsite, ": ") + 2))
                Process.Start(a)
            End If
        Next
    End Sub
    Private Sub Form1_Resize(ByVal sender As Object, _
```

```
                ByVal e As System.EventArgs) Handles Me.Resize
        If FixedHeight = 0 Then
            FixedHeight = Me.Height
        Else
            Me.Height = FixedHeight
        End If
    End Sub
End Class
```

代码放入了五个事件中，这些事件要么被应用启动触发，要么被用户交互触发。让我们逐个讨论这些事件以及它们所完成的功能。

Private Sub Form1_Load

该应用程序使用了一个列表视图(ListView)控件，该控件能让我们在列表中显示图像和描述。列表视图控件使用图像列表(ImageList)控件来存储显示的图像。我们在应用(.exe 文件)所在的文件夹中查找所有 Windows 位图(.bmp)文件，每个位图文件代表 MicroStation 中的一个单元。这些位图文件可以是 “CAD 绘制”或实拍或物体插图，它们都是用来表达单元的。我们使用 Image 对象的 GetThumbNailImage 方法在内存中为每个位图建立一个略图。虽然同样的功能也能在 VBA 中复制，但 VB.NET 完成这一步骤非常容易。在把每个略图添加到图像列表中后，我们用刚“建立”的略图图像和单元描述在列表视图中添加一项。

Private Sub lstvCells_ItemSelectionChanged

每当用户改变了列表视图中的选择时，我们查看是否有一个.info 文件对应于选择的位图。如果找到了一个.info 文件，就用.info 文件的内容赋值文本框。如果在.info 文件中发现含有“Website Address:”的链接，就使链接标签控件可见，这样用户就能点击该控件了。

Private Sub lstvCells_DoubleClick

当用户在列表视图上双击时，我们将双击的单元插入到 MicroStation 中去。有许多途径能确定单元的原点，本例把单元放置在了 MicroStation 当前视图的中心。代码运行时用一个进度条显示其进度。

Private Sub LineLabel1_LinkClick

在链接标签(LinkLabel)上点击的唯一途径是要能看见它。只有所选单元的.info 文件中有“Website Address:”时该标签才可见。用户点击链接标签后，我们取得.info 文件中的“Website Address:”并用网页地址启动一个新的进程。Process.Start 开始一个新进程(启动一个应用)并打开应用中提供的文件。Process.Start 等同于前几章中讨论过的 Windows API 调用 ShellExecute。

Private Sub Form1_Resize

VB6 和 VB.NET 中的用户窗体可被缩放。本例仅想让窗体在宽度方向上缩放，因此在窗体最初建立(Resize 事件和 Load 事件都在窗体显示时触发)时使用了一个叫做 **FixedHeight** 的变量，每当窗体缩放时继续使用该变量作为窗体的高度。

当我们查看上面所示的代码或者在 Visual Studio 中打开这个工程时，会发现缺少点什么，到底是什么呢？缺少错误处理。作为独立运行的.EXE 文件，有可能会在 MicroStation 未首先运行的情况下就打开该程序。通读代码后能看到，只有在 lstvCells_DoubleClick 事件中可能会出问题。让我们以 VB.NET 的方式添加一点错误处理，这与 VB6 和 VBA 的方式有些不同。

```
Private Sub lstvCells_DoubleClick(ByVal sender As Object, _
                ByVal e As System.EventArgs) Handles lstvCells.DoubleClick
    Dim myMSAppCon As MicroStationDGN.ApplicationObjectConnector
    Dim myMSApp As MicroStationDGN.Application
    Dim myCellElem As MicroStationDGN.CellElement
    Dim CellOrigin As MicroStationDGN.Point3d
    ProgressBar1.Visible = True
    ProgressBar1.Value = 10
    Try
        myMSAppCon = GetObject(, _
                "MicroStationDGN.ApplicationObjectConnector")
        ProgressBar1.Value = 20
        myMSApp = myMSAppCon.Application
        ProgressBar1.Value = 40
        CellOrigin = myMSApp.CommandState.LastView.Center
```

```
            ProgressBar1.Value = 60
            myCellElem = myMSApp.CreateCellElement3( _
                        lstvCells.SelectedItems(0).Text, CellOrigin, True)
            ProgressBar1.Value = 80
            myMSApp.ActiveModelReference.AddElement(myCellElem)
            ProgressBar1.Value = 100
        Catch ex As Exception
            Select Case Err.Number
            Case 429 'MicroStation not started
                MsgBox("MicroStation is not started." & vbCr & _
                        "Please start MicroStation and try again.")
                ProgressBar1.Visible = False
                Exit Sub
            Case Else
                MsgBox(ex.Message & vbCr & ex.StackTrace, , "ERROR")
                ProgressBar1.Visible = False
                Exit Sub
            End Select
        End Try
        ProgressBar1.Visible = False
End Sub
```

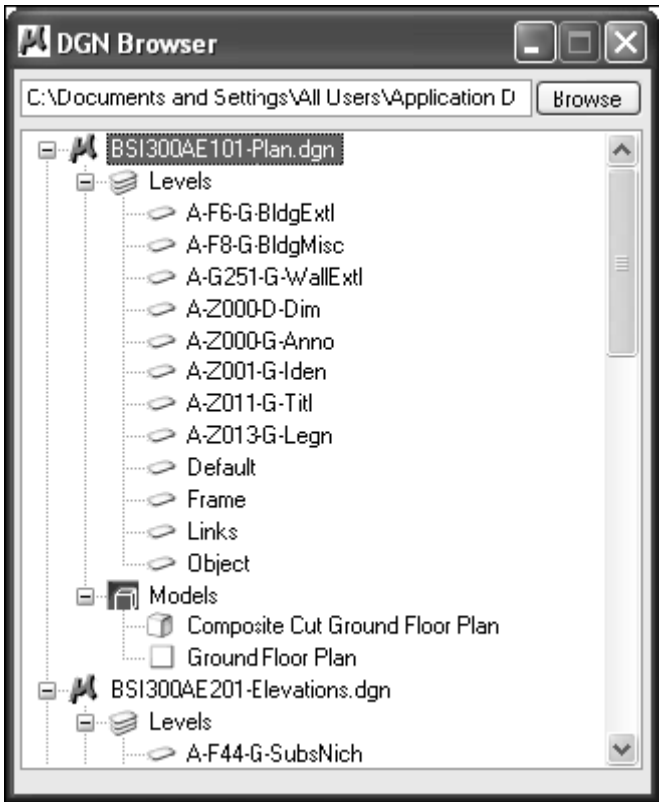
在 VB.NET 中执行错误检查时，我们预计某个代码行可能会引起错误。**Try(试)**着执行该代码行或一些行，在其后就 **Catch(捕获)**可能出现的错误。随后我们会看到几个附加的错误处理的例子。

DGN 浏览器应用程序

下一个应用程序仍然是一个 Windows 应用，名字叫做 DGN 浏览器。它利用了一个树视图控件、一个命令按钮、一个文本框、一个文件夹浏览对话框、一个通知图标、一个图像列表和一个工具提示控件。让我们看一下它运行时的 GUI。

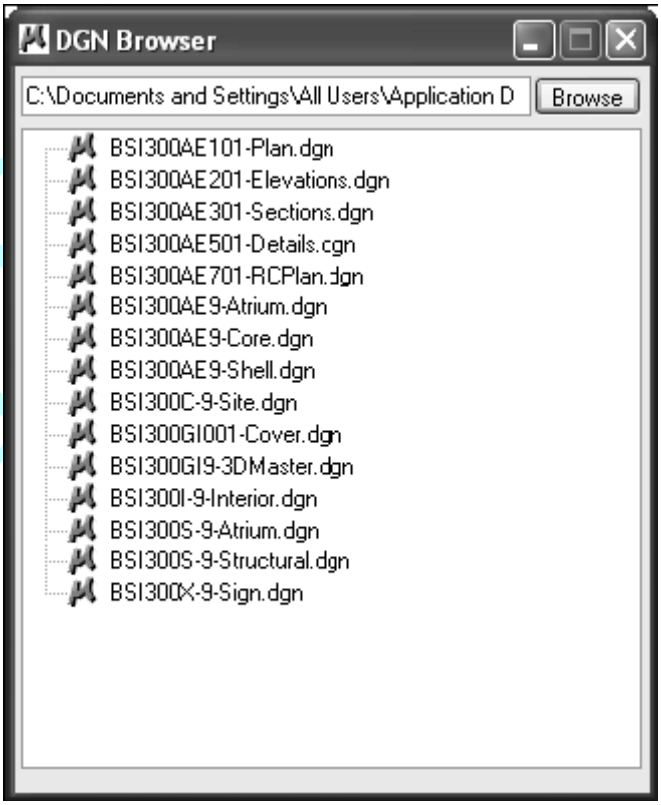
China BDN

用户浏览一个想查看的文件夹。文件夹中的每个.dgn 文件被装载到树视图控件中，而且每个.dgn 文件都在 MicroStation 中以“ForProgram”方式打开。层和模型被提取并显示在每个设计文件下的“Levels”和“Model”图标下。3D 和 2D 模型靠其图标来区分。



再强调一次，要使本程序工作正常，MicroStation 必须处于运行状态。我们不想让程序被庞大的列表撑爆了，所以，只在树视图中简单地显示了.dgn 文件而没有显示其中的层和模型。

右图是 MicroStation 未运行时本程序的样子：



先假设已经生成了该应用，而且自认为用户是想通过快速点击鼠标来启动应用。如何能让程序以这种方式被访问呢？途径之一就是将其放入任务栏的状态通

知区。



在 VB.NET 中实现这一功能很困难吗？只需简单地在窗体中添加一个“NotifyIcon(通知图标)”控件，它就会显示在任务栏中。我们当然不会让这一绝妙的程序显示一个标准 VB.NET 的图标，所以我们使用了 MicroStation V8 XM 的图标。

现在总结一下：

1、本程序通过双击 Windows 状态通知区的图标打开；

2、点击 Browse(浏览)按钮选择一个文件夹，然后程序在选择的文件夹中查找所有.dgn 文件并在树视图中显示文件名；

3、如果 MicroStation 处于打开状态，我们用 GetObject 连接到 MicroStation 并以“ForProgram”方式打开列表中的每个文件，把文件中的层和模型添加到树视图控件中。

就这么简单？使用 VB.NET 时这是真的。

下面是代码：

```
Public Class Form1
    Private LastPath As String
    Private myMSApp As MicroStationDGN.Application
    Private myMSAppCon As MicroStationDGN.ApplicationObjectConnector
    Private MicroStationError As Boolean
    Private MicroStationOpen As Boolean
    Private Sub NotifyIcon1_MouseDoubleClick(ByVal _
            sender As System.Object, _
            ByVal e As System.Windows.Forms.MouseEventArgs) _
            Handles NotifyIcon1.MouseDoubleClick
        Me.WindowState = FormWindowState.Normal
    End Sub
    Private Sub Button1_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles Button1.Click
        FBD1.SelectedPath = LastPath
        FBD1.Description = "DGN Explorer Path:"
```

```
        FBD1.ShowDialog()
        TextBox1.Text = FBD1.SelectedPath
        DisplayDGNs(FBD1.SelectedPath)
    End Sub
    Private Sub DisplayDGNs(ByVal PathIn As String)
        Dim myDI As IO.DirectoryInfo
        Dim myFI As IO.FileInfo
        Dim myNode As TreeNode
        Dim FileCounter As Long
        tv1.Nodes.Clear()
        myDI = New IO.DirectoryInfo(PathIn)
        If Not myDI.Exists Then
            MsgBox("The path " & PathIn & " does not exist.", _
                    MsgBoxStyle.Critical)
            TextBox1.Text = ""
            Exit Sub
        End If
        LastPath = PathIn
        ToolTip1.SetToolTip(TextBox1, PathIn)
        FileCounter = 0
        ProgressBar1.Visible = True
        For Each myFI In myDI.GetFiles
            FileCounter = FileCounter + 1
            ProgressBar1.Value = FileCounter * 10
            ProgressBar1.Refresh()
            Select Case myFI.Extension.ToUpper
            Case ".DGN"
                myNode = tv1.Nodes.Add(myFI.Name)
                myNode.ImageIndex = 0
                myNode.SelectedImageIndex = 0
```


China BDN

```
                If MicroStationError = False Then
                    GetFileComps(myFI.FullName, myNode)
                End If
            End Select
            If FileCounter = 10 Then FileCounter = 0
        Next
        ProgressBar1.Visible = False
        tv1.Sort()
    End Sub
    Private Sub Form1_Load(ByVal sender As System.Object, _
                    ByVal e As System.EventArgs) Handles MyBase.Load
        LastPath = "C:\MicroStation VBA"
    End Sub
    Private Sub GetFileComps(ByVal FileIn As String, _
                                ByVal FileNode As TreeNode)
        Dim myLevel As MicroStationDGN.Level
        Dim myModel As MicroStationDGN.ModelReference
        Dim LevelNode As TreeNode
        Dim ModelNode As TreeNode
        Dim tmpNode As TreeNode
        If MicroStationOpen = False Then
            Try
                myMSAppCon = GetObject(, _
                    "MicroStationDGN.ApplicationObjectConnector")
                myMSApp = myMSAppCon.Application
            Catch
                MicroStationError = True
                Exit Sub
            End Try
            Dim DGNFile As MicroStationDGN.DesignFile
```


China BDN

```
        Try
            DGNFile = myMSApp.OpenDesignFileForProgram( _
                                    FileIn, True)
        Catch
            FileNode.ForeColor = Color.Red
            Exit Sub
        End Try
        LevelNode = FileNode.Nodes.Add("Levels")
        LevelNode.ImageIndex = 1
        LevelNode.SelectedImageIndex = 1
        ModelNode = FileNode.Nodes.Add("Models")
        ModelNode.ImageIndex = 3
        ModelNode.SelectedImageIndex = 3
        For Each myLevel In DGNFile.Levels
            tmpNode = LevelNode.Nodes.Add(myLevel.Name)
            tmpNode.ImageIndex = 2
            tmpNode.SelectedImageIndex = 2
        Next
        For Each myModel In DGNFile.Models
            tmpNode = ModelNode.Nodes.Add(myModel.Name)
            If myModel.Is3D Then
                tmpNode.ImageIndex = 4
                tmpNode.SelectedImageIndex = 4
            Else
                tmpNode.ImageIndex = 5
                tmpNode.SelectedImageIndex = 5
            End If
        Next
        DGNFile = Nothing
    End If
```

China BDN

```
    End Sub
End Class
```

图像列表图标在随书附赠的 CD 上。

还能使本程序更加强大吗？当然能。在树中双击时自动打开文件；允许用户从 Windows 资源管理器中拖拽文件夹到我们的 DGN 管理器中；用一个能存储最近选择文件夹的复合框取代现在的显示选择路径的文本框；选择文件夹时有一个是否选择子文件夹选项。对这个工程还能做许许多多的工作，但在此就不做了，我们把这些润色工作留给读者来完成。

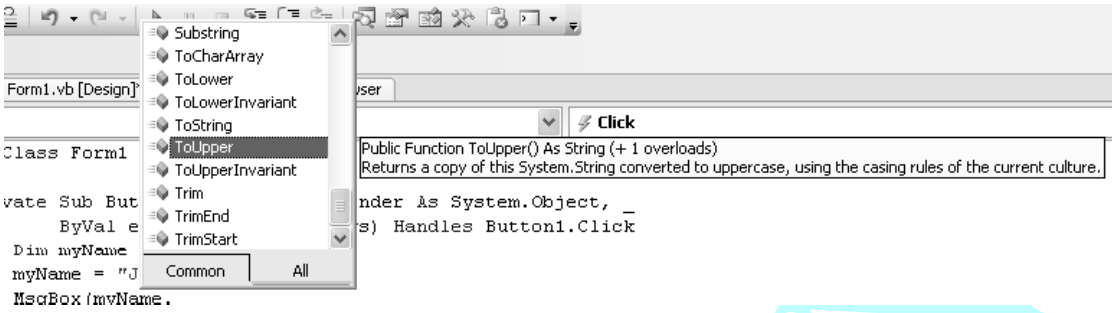
VBA 到 VB.NET 引用

经常被问及的一个问题是：“我应该学习哪种编程语言？VBA？VB6？VB.NET？”这是一个好问题。如果您向 100 个专家求教这个问题，可能会听到 100 种不同的答案。我们在此之前讨论过一些为何要用 VBA、VB6 或 VB.NET 进行开发的原因。假如在 VBA 中开发有非常明确的好处，那么当然就只在 VBA 中开发了。如果 VBA 是您基本的开发环境，那么过渡到 VB6 比直接过渡到 VB.NET 会更自然。然而，正如专家们所说的：“大事不妙了”，VB6 正在过时，VB.NET 才是未来。所以，最好还是要熟悉 VB.NET 环境、了解在 VB.NET 中如何工作。对于 VBA，微软已经为应用程序开发了一个类似.NET 的 VBA 环境，在未来的几年中可能会采用。

本书的这一节不是用来列出 VBA 到 VB.NET 调用的完整列表的。本节的目的是提供一些信息，告诉您在本书中已经介绍过的一些通用内容如何在 VB.NET 中完成。有关 VB.NET 环境的更多信息参见本书的“其它参考资料”一节。

一切皆对象

在 VBA 或 VB6 中开发时，可以把变量声明为一个 String(字符串)。在 VBA 和 VB6 中，String 是数据类型而不是对象。在 VB.NET 中，String 是一个具有其属性和方法的对象。例如，当我们键入声明为 String 的一个变量名并按句点键时，将看到：



正如在此看到的，我们能用 String 对象的 ToUpper 方法代替 UCase(如下)。

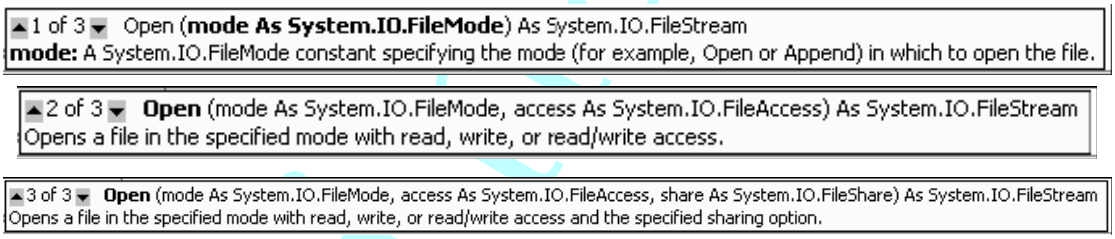
Ucase(myName) ‘VB6 and VBA

myString.ToUpper ‘VB.NET

ToUpper 是 String 对象的一个方法。String 对象的属性又如何呢？是的，它也有属性。在 VB.NET 中，我们使用 String 对象的 Length 属性代替 Len 函数(在 VB6 和 VBA 中的用法)。

重载

重载(生活中的超载)这个词引出了需要注意的逼真画面。在 VB.NET 中，它意味着一个过程、方法、函数等可以具有多个实现，每个实现都有其自己的一套参数。下面就是使用 FileInfo 对象的 Open 方法的例子：



当要使用 FileInfo 对象的 Open 方法时，最先看到的智能提示是“1 of 3”。可以用箭头键或点击工具提示中显示的上下箭头来浏览每个重载方法。这些方法使用了不同个数和类型的参数但都完成的是同一项任务 — 把文件作为 FileStream 对象打开。

过程和函数

在 VB.NET 中，圆括号放在过程调用和函数调用两端。这样的一个例子就是 MessageBox。

在 VBA 和 VB6 中：

MsgBox “This is a test.”

在 VB.NET 中：

MsgBox(“This is a test.”)

存取 ASCII 文件

在 VBA 和 VB6 中读取和写入 ASCII 文件是容易的，在 VB.NET 中也很容易。

在 VBA 中读取文件

```
Sub ReadFileVBA()
    Dim FFile As Long
    Dim strLine As String
    FFile = FreeFile
    Open "C:\MicroStation VBA\points.txt" For Input As #FFile
    While EOF(FFile) = False
        Line Input #FFile, strLine
        Debug.Print strLine
    Wend
    Close #FFile
End Sub
```

在 VB.NET 中读取文件

```
Sub ReadFileA()
    Dim myFS As New IO.StreamReader( _
                    "C:\MicroStation VBA\points.txt")
    While myFS.EndOfStream = False
        Debug.Print(myFS.ReadLine)
    End While
    myFS.Close()
End Sub
Sub ReadFileB()
    Dim myFS As New IO.StreamReader( _
                        "C:\MicroStation VBA\points.txt")
```

China BDN

```
        Debug.Print(myFS.ReadToEnd)
        myFS.Close()
    End Sub
```

在 VB.NET 中也能像 VBA 中那样每次一行地读取文件，还有一个 ReadToEnd 的选项。当使用 ReadToEnd 方法时，一次性将整个文件在一个字符串中返回。

在 VBA 中写入文件

```
    Sub WriteFileVBA()
        Dim FFile As Long
        Dim strLine As String
        FFile = FreeFile
        Open "C:\MicroStation VBA\points2.txt" For Output As #FFile
        Print #FFile, "1,1,0,1"
        Print #FFile, "2,2,0,2"
        Print #FFile, "3,3,0,3"
        Print #FFile, "4,4,0,4"
        Close #FFile
    End Sub
```

在 VB.NET 中写入文件

```
    Sub WriteFileA()
        Dim myFW As New IO.StreamWriter( _
                                "C:\MicroStation VBA\points2.txt")
        myFW.WriteLine("1,1,0,1")
        myFW.WriteLine("2,2,0,2")
        myFW.WriteLine("3,3,0,3")
        myFW.WriteLine("4,4,0,4")
        myFW.Close()
    End Sub
```

在 VB.NET 中写入文件与 VBA 中非常相似，也是每次写一行。

China BDN

VB.NET 中发送电子邮件

用 VBA 发送电子右击的例子参考前些章的相应部分。下面是 VB.NET 的几个例子：

```
Sub SendMailA()
    Dim mySMTP As New System.Net.Mail.SmtpClient
    mySMTP.Host = "yoursmtpserver.com"
    mySMTP.Send("me@bentley.com", "you@bentley.com", _
                "MicroStation VBA", "Emailing using .NET is easy.")
End Sub
Sub SendMailB()
    Dim mySMTP As New System.Net.Mail.SmtpClient
    Dim myMessage As New System.Net.Mail.MailMessage
    Dim myAttachment As New System.Net.Mail.Attachment( _
                                "C:\MicroStation VBA\points.txt")
    mySMTP.Host = "yoursmtpserver.com"
    myMessage.To.Add("you@bentley.com")
    myMessage.From = New _
                System.Net.Mail.MailAddress("me@bentley.com")
    myMessage.Subject = "Emailing an Attachment"
    myMessage.Body = "<b>HTML</b> Email is also easy."
    myMessage.IsBodyHtml = True
    myMessage.Attachments.Add(myAttachment)
    mySMTP.Send(myMessage)
End Sub
```

第一个例子 SendMailA 为我们显示了仅用三行代码就能发送一封电子邮件。SendMailB 以稍微不同的方式完成发送，它为正要发送的邮件添加了一个附件。当然了，这些例子中的 SMTP.Host、From 和 To 字段需要修改成合法的电子邮件服务器和邮箱设置。

China BDN

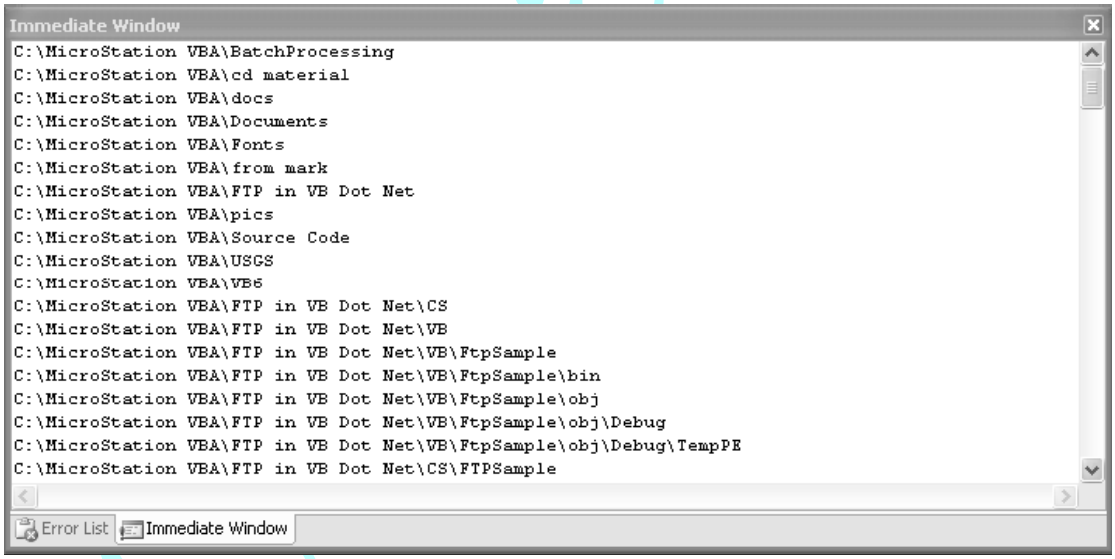
遍历文件夹及其子文件夹

我们已经在 VBA 中看过如何用文件系统对象实现这个功能，现在来看一看怎样在 VB.NET 中来实现。第一个例子取得指定目录的所有子目录：

```
Sub TraverseFolders()
    Dim myDI As New IO.DirectoryInfo("C:\MicroStation VBA")
    Dim mySubDI As IO.DirectoryInfo
    For Each mySubDI In myDI.GetDirectories("*.*", _
                                IO.SearchOption.AllDirectories)
        Debug.Print(mySubDI.FullName)
    Next
End Sub
```

GetDirectories 有三种不同的实现。这里显示的一种能让我们指定查找的模式，还能指定“AllDirectories(所有目录)”或“TopDirectoryOnly(仅顶级目录)”。

在 VBA 中，使用了一个递归过程取得子文件夹，而在 VB.NET 中仅用一行代码就取得了所有文件夹。



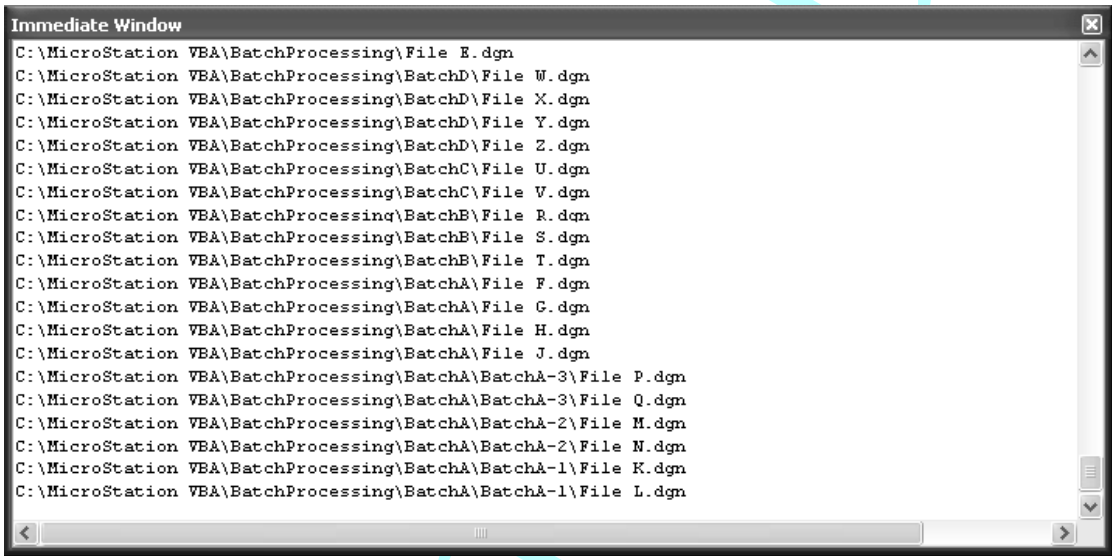
取得一个路径中的所有文件

正像 VB.NET 很容易地遍历指定文件夹下的子文件夹那样，遍历指定文件夹(及其子文件夹)下的文件也只需几行代码就能完成：

```
Sub FindFilesA()
```

```
    Dim myDI As New IO.DirectoryInfo("C:\MicroStation VBA")
    Dim myFI As IO.FileInfo
    For Each myFI In myDI.GetFiles("*.DGN", _
                        IO.SearchOption.AllDirectories)
        Debug.Print(myFI.FullName)
    Next
End Sub
```

本例取得 C:\MicroStation VBA 路径及其子目录下的所有.dgn 文件。



返回函数值

当函数需要返回一个值时，VB.NET 要求使用 Return 语句。这与 VBA 和 VB6 不同，在那里是使用函数名，然后是等号，再后来是要返回的值或对象。

Windows API 调用

.NET 框架公开了许多具有方法和属性的对象，这些对象以前是专属于 Windows API 调用领域的。例如，访问整个 Windows 注册表是一个有些乏味的过程。当然了，我们可以建立包装有访问注册表代码的类。尽管在 VB.NET 中访问注册表相当容易，但并非所有的 Windows API 调用都唾手可得，下一个例子就说明了这一点。

在 Windows 资源管理器中查看时，文件以其名字和图标显示。图标通常反映了用于打开或编辑该文件的应用程序。在注册表中能找到几处存有特定文件类

型所用“DefaultIcon(缺省图标)”的地方。下个例子将查看文件夹中每个文件的扩展名，在注册表中查找每个文件扩展名所用的缺省图标，然后把图标保存为一个位图(.bmp)文件。为了实现这些功能，我们将使用两个 Windows API 调用以及.NET 内置的注册表访问对象。

下面是 VB.NET 中紧跟“Public Class”之下声明的 Windows API 调用：

```
Declare Function ExtractIcon Lib "shell32.dll" Alias _
        "ExtractIconA" (ByVal hInst As Integer, _
        ByVal lpszExeFileName As String, _
        ByVal nIconIndex As Integer) As Integer
Public Declare Function FindExecutable Lib "shell32.dll" Alias _
        "FindExecutableA" (ByVal lpFile As String, _
        ByVal lpDirectory As String, ByVal lpResult As String) As Long
```

第一个调用 ExtractIcon 从一个.exe 或.dll 文件从提取图标。FindExecutable 让我们指定一个文件名，它返回注册用来打开该文件的程序的路径。稍后一会儿将看到这些函数的使用。

让我们查看程序开始的主过程。

1、一开始取得指向路径 C:\MicroStation VBA 的 DirectoryInfo 对象；

2、在查看文件夹中的每个文件时，检查是否已经为该扩展名类型的文件建立了位图文件。如果还未建立，就开始指定要建立位图所用的图标文件；

3、指定图标的第一步就是在 Windows 注册表中根据文件扩展名来查找。如果失败，就从该文件注册的应用中取得图标。如果还是失败，就取得缺省的 Windows 图标(来自于 shell32.dll 文件)；

4、在成功地取得一个图标后，将其保存为位图。然后在代码中将图标放入一个 PictureBox 控件并存成一个文件。使用 PictureBox 控件的原因是为了保持图标文件中的一些透明度信息。

```
Sub SaveFileIcons()
    Dim myDI As New IO.DirectoryInfo("C:\MicroStation VBA")
    Dim myFI As IO.FileInfo
    Dim IconFile As IO.FileInfo
    Dim myIcon As Bitmap
```

```
Dim tmpImage As Image
Dim myPictureBox As New PictureBox
For Each myFI In myDI.GetFiles
    IconFile = New IO.FileInfo(myDI.FullName & "\" & _
                    myFI.Extension.Replace(".", "") & ".bmp")
    If IconFile.Exists = False Then
        'First look at extension
        myIcon = IconFromExtension(myFI.Extension)
        If myIcon Is Nothing = False Then
            myIcon.Save("c:\MicroStation VBA\" & _
                    myFI.Extension.Replace(".", "") & ".bmp", _
                    System.Drawing.Imaging.ImageFormat.Bmp)
        Else
            'Now look at file
            myIcon = IconFromFile(myFI.FullName)
            If myIcon Is Nothing = False Then
                myIcon.Save("c:\MicroStation VBA\" & _
                        myFI.Extension.Replace(".", "") & ".bmp", _
                        System.Drawing.Imaging.ImageFormat.Bmp)
            Else
                'Use Default Windows Icon
                myIcon = GetDefWinIcon()
                myIcon.Save("c:\MicroStation VBA\" & _
                        myFI.Extension.Replace(".", "") & ".bmp", _
                        System.Drawing.Imaging.ImageFormat.Bmp)
            End If
        End If
        If myIcon Is Nothing = False Then
            myPictureBox.Image = myIcon
            tmpImage = myPictureBox.Image
```


ChinaBDN

```
                    tmpImage.Save("C:\MicroStation VBA\" & _
                                  myFI.Extension.Replace(".", "") & ".abmp")
                End If
            End If
        Next
End Sub
```

函数 IconFromExtension、IconFromFile 和 GetDefWinIcon 都使用了另一个函数 IconFromValue，该函数分析从注册表中检索到的 DefaultIcon 值的结果。

```
Function IconFromExtension(ByVal FileExtension As String) As Bitmap
    Dim MyKey As Microsoft.Win32.RegistryKey
    Dim MyDefaultKey As Microsoft.Win32.RegistryKey
    Dim myCR As Microsoft.Win32.RegistryKey
    Dim myDefaultIcon As Microsoft.Win32.RegistryKey
    Dim DefValue As String
    myCR = Microsoft.Win32.Registry.ClassesRoot
    MyKey = myCR.OpenSubKey(FileExtension)
    If MyKey Is Nothing Then
        Return Nothing
        Exit Function
    End If
    'First look for Value
    DefValue = MyKey.GetValue("DefaultIcon")
    If DefValue Is Nothing = False Then
        Return IconFromValue(DefValue)
        Exit Function
    End If
    'Next look for Key
    DefValue = MyKey.GetValue("")
    If DefValue Is Nothing = False Then
        MyDefaultKey = myCR.OpenSubKey(DefValue)
```


China BDN

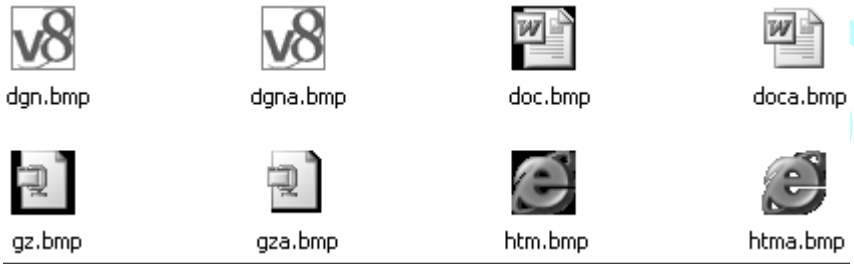
```
            If MyDefaultKey Is Nothing = False Then
                'Look for Key
                myDefaultIcon = MyDefaultKey.OpenSubKey("DefaultIcon")
                If myDefaultIcon Is Nothing = False Then
                    Return IconFromValue(myDefaultIcon.GetValue(""))
                    Exit Function
                Else
                    'Look for Value
                    Return _
                    IconFromValue(MyDefaultKey.GetValue("DefaultIcon"))
                    Exit Function
                End If
            End If
        End If
        Return Nothing
End Function
Function IconFromFile(ByVal FileIn As String) As Bitmap
        Dim myDI As IO.DirectoryInfo
        Dim myFI As New IO.FileInfo(FileIn)
        Dim myExe As String
        Dim nIcon As Long
        Dim tmpBitmap As Bitmap
        myDI = New IO.DirectoryInfo(myFI.DirectoryName)
        myExe = Space(255)
        FindExecutable(myFI.Name, myDI.FullName, myExe)
        nIcon = ExtractIcon(0, myExe.Substring(0, _
                                myExe.IndexOf(Chr(0))), 0)
        If nIcon > 0 Then
            tmpBitmap = Bitmap.FromHicon(nIcon)
            Return tmpBitmap
```


China BDN

```
    Else
        Return Nothing
    End If
End Function
Function GetDefWinIcon() As Bitmap
    Dim nIcon As Long
    Dim tmpBitmap As Bitmap
    nIcon = ExtractIcon(0, "C:\Windows\System32\shell32.dll", 0)
    tmpBitmap = Bitmap.FromHicon(nIcon)
    Return tmpBitmap
End Function
Function IconFromValue(ByVal ValueIn As String) As Bitmap
    Dim xSplit() As String
    Dim tmpBitmap As Bitmap
    Dim nIcon As Long
    tmpBitmap = Nothing
    xSplit = Split(ValueIn, ",")
    If xSplit(0) = "" Then
        Return Nothing
        Exit Function
    End If
    Select Case xSplit.Length
    Case 1 '.ico file
        If xSplit(0).Contains("%") = False Then
            If xSplit(0) <> "" Then
                tmpBitmap = Bitmap.FromFile(xSplit(0))
            End If
        End If
    Case 2 '.exe or .dll file
        nIcon = ExtractIcon(0, xSplit(0), xSplit(1))
```


China BDN

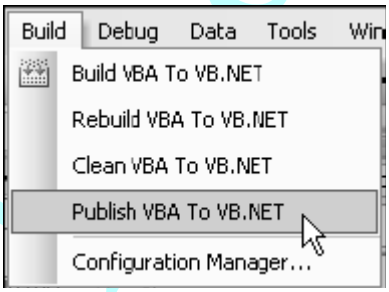
```
        If nIcon > 0 Then
            tmpBitmap = Bitmap.FromHicon(nIcon)
        End If
    Case Else
        Return Nothing
    End Select
    Return tmpBitmap
End Function
```



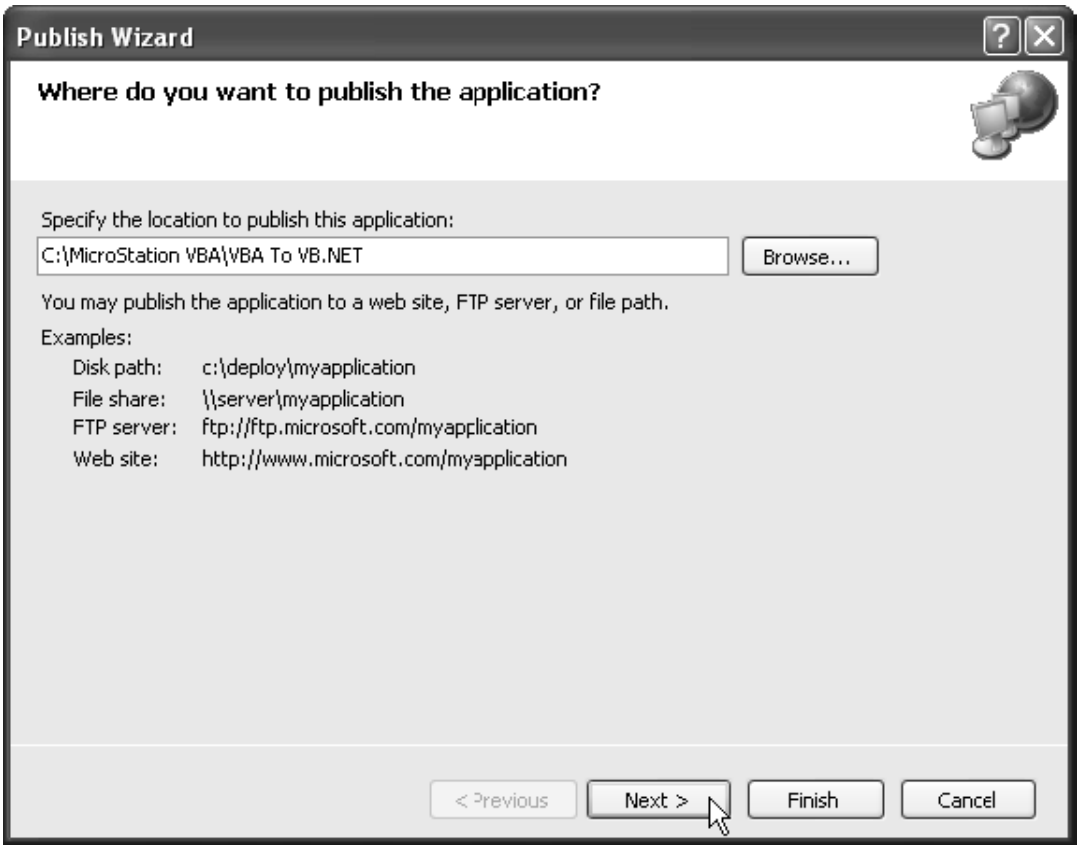
该程序运行后，为指定文件夹中找到的每种文件扩展名建立位图文件。

分发 VB.NET 应用程序

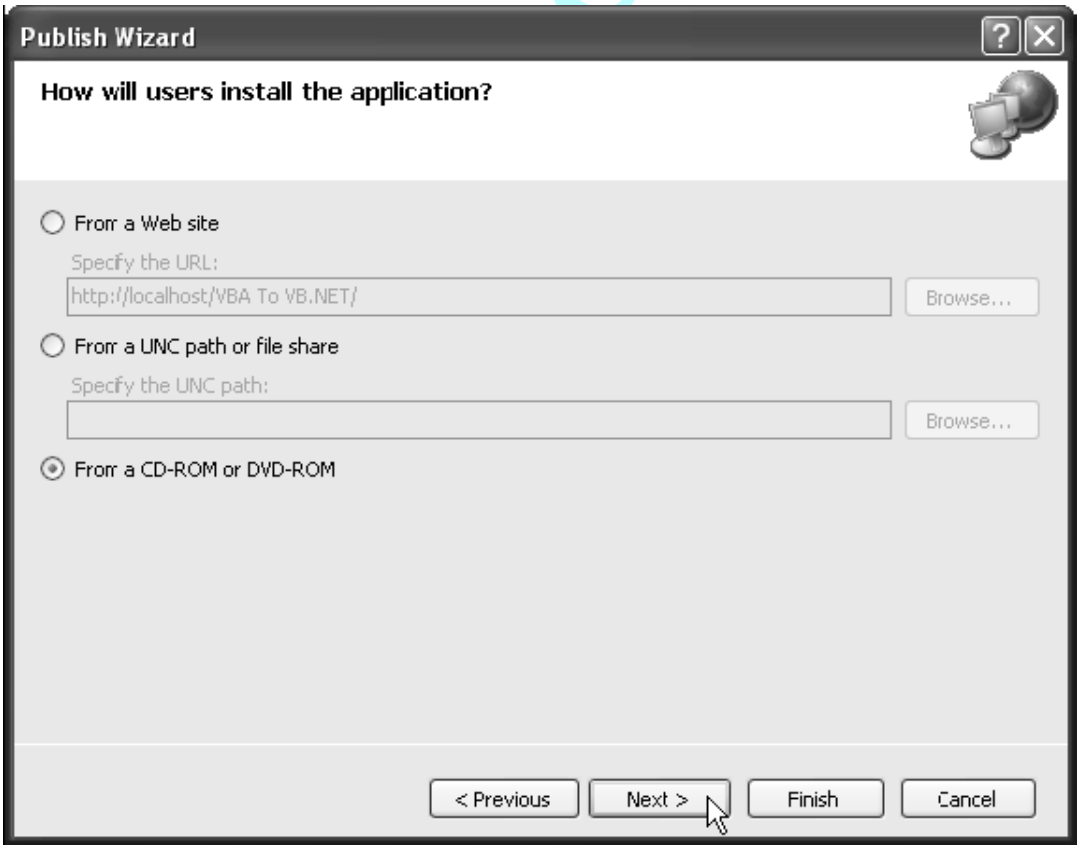
分发一个 VB.NET 应用只是点几下鼠标而已，因为它被生成为 VB.NET。



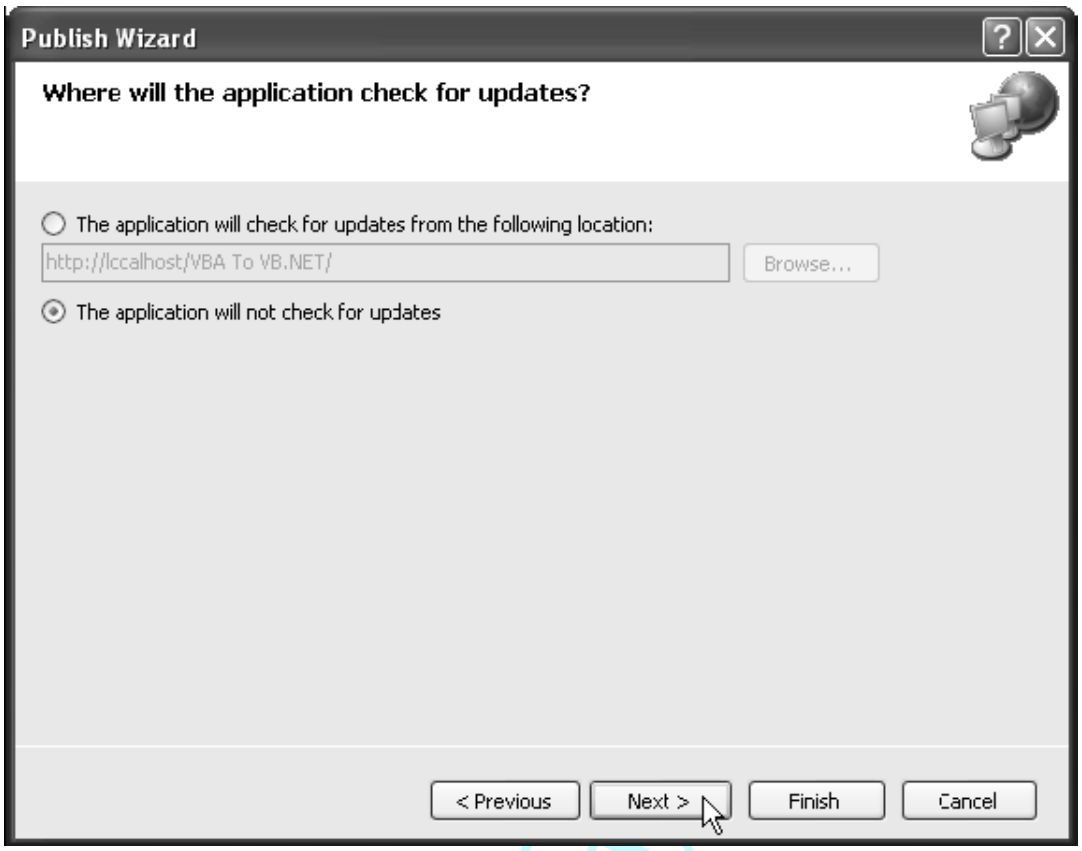
1、使用 Build > Publish XXXXXXX 开始“发布”过程；



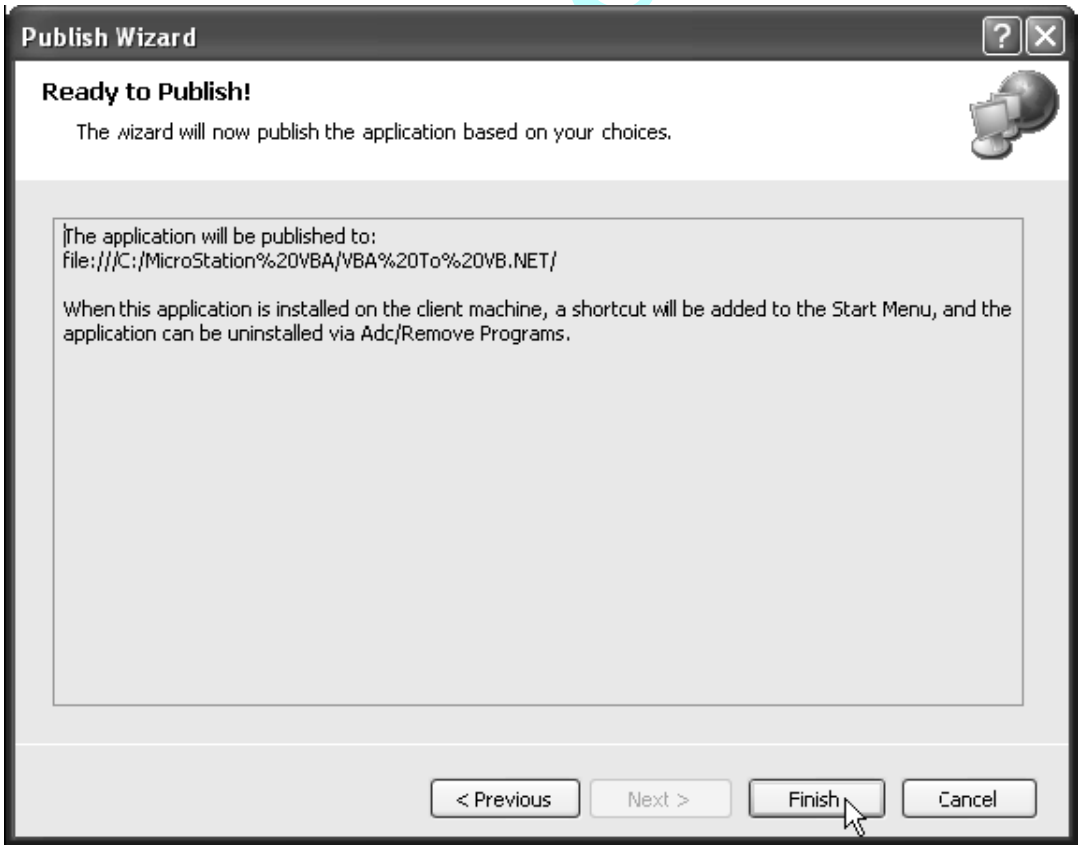
先要指定应用发布的位置。除了发布到硬盘上以外，还可以发布到一个 FTP 服务器或 Web 服务器上。



2、本例将建立一个从 CD-ROM 或 DVD-ROM 运行的 Setup 程序；

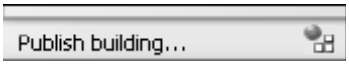


为简单起见，不使用更新特性。

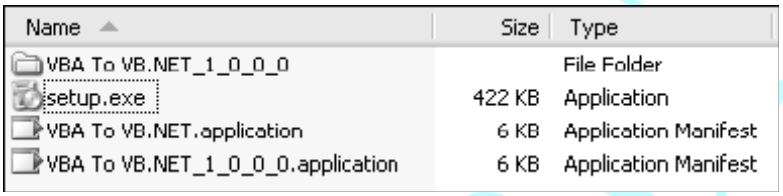


3、点击 Finish(完成)使发布向导建立 Setup 文件。

建立 Setup 文件要花上一会儿时间。“Pubish building”会显示在 VB.NET IDE 的状态栏，还带有一个动画图标。



下面是发布向导生成的文件。



本章回顾

与 Microsoft Excel 或 VB6 相比，使用 VB.NET 来控制 MicroStation 的特殊 API 没有什么不同。我们仍然添加了“Bentley MicroStation DGN #.# Library”引用，仍然使用了 GetObject。在 VB.NET 与 VB6/VBA 之间也有一些不同。这些不同常常发生在文件和文件夹访问的极大简化方面，还有其它一些以前编程困难且乏味的方面。有一个小问题就是，VB.NET 是 VB 编程的未来。虽然我们可以继续在 MicroStation VBA 环境中开发强大的应用，但逐渐熟悉.NET 世界是个好主意。

China BIM

附加资源

通用 VBA 资源

http://msdn.microsoft.com/vba

(VBA 概述，白皮书等)

http://bentleyinstitute.bentley.com/catalog.aspx?discipline=10

(由 Bentley 提供的编程课程)

Mastering Microsoft VBA(《掌握 Microsoft VBA》)；ISBN: 0782144365

VBA Developer's Handbook(《VBA 开发者手册》)；ISBN: 0782129781

在 Google、Yahoo 或其它网上搜索“VBA”或“Visual Basic for Applications”

http://discussion.bentley.com

(查找 bentley.microstation.v8xm.vba 讨论组)

VB and VBA in a Nutshell(《VB 和 VBA 简介》)；ISBN: 1565923588

SQL 语句

http://msdn.microsoft.com/library/en-us/tsqlref/ts_sases_9sfo.asp?frame=true

(来自微软的 SQL 语句解释和例子)

VB.NET

http://msdn.microsoft.com/vbasic/

Visual Basic 2005 Programmer's Reference (Programmer to Programmer) (《Visual Basic 2005 程序员参考》(程序员写给程序员的书))；ISBN: 0764571982

MATHCAD

www.mathcad.com

XML

XML in a Nutshell, Third Edition(《XML 简介，第三版》)；ISBN: 0596007647

XML Programming Bible(《XML 编程经典》)；ISBN: 0764538292

China BDN