0839459

中文书 30839459

AutoCAD VBA 二次开发教程

张 帆 主编

郑立楷 卢择临 王成煌 编著

清华大学出版社

北 京

内 容 简 介

本书以 AutoCAD 2005 中文版为开发平台，通过 73 个精彩的编程实例，向读者展示了 AutoCAD VBA 开发的高级技术。

全书共 16 章，将所有的实例归入基础相关、控制 AutoCAD 环境、创建和编辑二维对象、对象特性、文字、标注和公差、块和属性、处理 AutoCAD 中的事件、三维空间的操作、打印、高级绘图组织、使用组件扩展程序功能、和 Excel 交换数据、数据库应用程序、发布 VBA 应用程序等大类，从不同的角度分析 VBA 编程的技术和技巧。本书提供了多个精彩实例的源码，涉及 VBA 的基础操作、VBA 的应用领域、VBA 的特殊功能，以及 CAD 软件开发的一些基础知识。

本书层次清晰、实例丰富，适用于建筑、机械、电子等多个领域的工程设计人员和软件开发工程师，同时也可供对计算机辅助设计有兴趣的大学生、研究生以及各工程单位相关技术人员使用。对于 VBA 的读者来说，这是一本具有较强指导性和实用性的图书。

版权所有，翻印必究。举报电话：010-62782989　13501256678　13801310933
本书封面贴有清华大学出版社防伪标签，无标签者不得销售。
本书防伪标签采用特殊防伪技术，用户可通过在图案表面涂抹清水，图案消失，水干后图案复现；或将表面膜揭下，放在白纸上用彩笔涂抹，图案在白纸上再现的方法识别真伪。

图书在版编目(CIP)数据

AutoCAD VBA 二次开发教程/张帆主编；郑立楷，卢择临，王成煌编著. —北京：清华大学出版社，2006.1
ISBN 7-302-11982-1

Ⅰ. A… Ⅱ. ①张… ②郑… ③卢… ④王… Ⅲ. 计算机辅助设计—应用软件，AutoCAD Ⅳ. TP391.72

中国版本图书馆 CIP 数据核字(2005)第 118850 号

出 版 者：清华大学出版社　　地　　址：北京清华大学学研大厦
　　http://www.tup.com.cn　　邮　　编：100084
　　社 总 机：010-62770175　　客户服务：010-62776969
责任编辑：魏江江
封面设计：杨　兮
印 装 者：北京鑫海金澳胶印有限公司
发 行 者：新华书店总店北京发行所
开　　本：185×260　印张：31　字数：773 千字
版　　次：2006 年 1 月第 1 版　2006 年 1 月第1 次印刷
书　　号：ISBN 7-302-11982-1/TP·7760
印　　数：1～3500
定　　价：49.00 元(含光盘)

前　言

读者对象

本书适用于使用 VBA 和 ActiveX 技术对 AutoCAD 进行二次开发的工程技术人员。

如果对 AutoCAD VBA 开发毫无概念，建议首先阅读清华大学出版社出版的《AutoCAD VBA 开发精彩实例教程》。

本书内容

本书的内容分为 16 章，另有一个附录，从各个方面向读者展示 AutoCAD VBA 开发的知识。

第 1 章深入分析了 AutoCAD VBA 开发中两个重要的知识点——宏和集合对象。宏是 AutoCAD VBA 开发中一个最基础但又非常重要的概念，而集合对象在开发过程中使用非常频繁，有必要在学习的开始阶段打下坚实的基础。

第 2 章包括文档操作、缩放视图、命名视图，以及和用户交互的操作。“文档操作”部分可以看作是集合对象使用的继续深入探讨，缩放视图使用 ActiveX 模型中提供的缩放方法模拟了 AutoCAD 自身的几种缩放功能，“和用户交互”一节提出了在 GetPoint 方法中使用关键字的解决方案。

第 3 章对选择集的使用进行了全面的剖析，为 SendCommand 提供了更广泛的应用，分析了直线、圆、圆弧和样条曲线向多段线的转化，提供了一些常用的多段线操作，提出制图法解决问题的新思路，编写了根据起点、终点和弧长创建圆弧的函数，最后又给出绘制放样展开图的方法。

第 4 章包含了两个有用的程序。按图层缩放提供了显示整个图层上所有实体的功能；使用 ObjectDBX 技术导入其他图形中的信息，相当于提供了一种后期“绑定”样板文件的手段。

第 5 章介绍了与文字使用有关的程序设计。创建文字样式能够获得当前计算机中可用的字体列表；文字按插入点自动对齐提供了对自定义数据类型的数组进行排序的一个思路；提取多行文字中的文字内容对 AutoCAD 中多行文字的格式代码进行了分析。使用 ObjectDBX 实现的批量文字替换，与上一个版本比较起来，修正了一些 Bug，并且是在不打开文件的情况下完成操作的，速度有了明显的提高。

第 6 章介绍了标注和公差的使用。新的圆弧长度标注程序涉及到多行文字格式代码的使用；锁定标注文字的程序对于创建局部放大图形时非常有用，技术方面涉及到匿名块、尺寸标注对象的组成、GetSubEntity 方法的使用等；列举所有的公差符号程序仍然涉及到多行文字的格式代码，从程序中还可以发现访问公差内容的方法。

第 7 章介绍了块和属性相关的程序设计，简单总结了简单块和属性块的使用，提供了将已经存在的实体添加到块定义中的方法，探讨了匿名块的使用。块和属性的结合使用非常灵活，自动标题栏和块属性生成表格两个程序演示了这种应用。实现“斜切”功能应该属于

块应用的一个副产品，利用插入块参照时可以指定不同 X、Y、Z 坐标的特点，实现特殊的图形变换。

第 8 章介绍了 AutoCAD 事件处理的方法。利用几个典型的实例介绍了应用程序层事件(禁用命令)、文档层事件(双击事件和快捷菜单事件)和对象层事件(临时"反应器")的处理方法，然后用一个序号球的实例模拟了永久"反应器"。

第 9 章处理在三维空间中的程序设计。在三维操作中，提供了在 UCS 中创建基本图元的方法，让编程人员也能享受 UCS 带来的便利。三点法创建 UCS、在任意平面上创建图案填充、创建正多面体和创建偏心圆台都以数学知识为基础，解决几个在三维建模中应用的典型问题。

第 10 章讨论打印和布局的问题。本章仅提供了批量打印的程序。

第 11 章介绍了扩展数据、扩展字典、对象命名字典、SummaryInfo 对象和绘图顺序等高级绘图组织技术。其中，扩展数据和扩展字典是标识实体的一种手段，SummaryInfo 对象提供了访问图形特性的接口，绘图顺序则用于调整图形对象的显示层次。

第 12 章探讨了在 VBA 开发应用程序过程中可能会用到的几种技术和技巧。调试应用程序和应用程序的错误处理对于程序开发来说必不可少，这里用作者自身的经历来介绍其中的技巧；DwgThumbnail、Accont、Slide、MSScript 是几个很有特色的组件，能够为 VBA 程序增加一些特殊的功能；创建和使用定时器一节直接用 Windows API 函数在类模块中自己创建定时器。

第 13 章将注意力集中在 Excel、Word 和 AutoCAD 的数据交换，以及在 VBA 使用 Windows API 实现特殊效果。VBA 和 Excel 的数据交换似乎一直是一个热点问题，本章提供了 Excel 表格向 AutoCAD 的转换、使用 Excel 数据绘图、转换 AutoCAD 数据到 Word 和 Excel 中几个实例。在使用 Windows API 方面，提供了去掉对话框的关闭按钮和添加快捷菜单的实例。

第 14 章介绍在 VBA 中创建数据库应用程序的方法。由于 ADO 技术是比较理想的一种技术，因此本章着重介绍 ADO 和 SQL 语句结合开发数据库应用程序。此外，数据记录和 AutoCAD 实体的对应一节探讨了实现属性数据和地理数据的对应，这对于在 AutoCAD 上实现 GIS 的读者非常有用。

第 15 章提供发布 VBA 应用程序的各种技术，包括菜单宏的使用、在工具栏中使用自定义图标、基本图库的实现(动态修改菜单文件)、在命令行或菜单中执行 VBA 程序、实现 VBA 程序的先选择再操作、更改 AutoCAD 应用程序的图标和标题栏、使用 InstallShield 完全发布 VBA 应用程序。

第 16 章介绍了使用 VB 开发 AutoCAD 的一些知识(将 VBA 代码转换到 VB 中，VB 窗体和 AutoCAD 的焦点切换，使用 ActiveX DLL 封装带有窗体的 VB 程序)、搜索某个文件夹中所有的文件和 VBA IDE 的编程，利用这些知识，完全有可能开发出功能更为强大的应用程序。

附录

本书的附录包含了 AutoCAD 2005 帮助文档中《ActiveX 和 VBA 参考》(英文版)部分的内容，并且已经翻译为中文，供读者在学习过程中参考。

配套光盘

本书的配套光盘给出了除附录内容之外的所有源程序代码，以及在编程中可能用到的资源文件。

使用配套光盘，就可以直接将代码复制到读者本人的计算机中学习，而不必键入源程序代码。此外，读者可以自己写代码，将程序运行结果与光盘中程序执行结果相对照。

本书适用版本

本书所有程序都能够运行在 AutoCAD 2005 中文版和英文版平台上。

AutoCAD 2005 VBA 中新增了一些对象和方法，但是在不使用这些新增功能的情况下，仍然可以在低版本的 AutoCAD 中运行这些程序。本书作者不保证所有的程序都能在 AutoCAD 2004 及其以下版本中使用（很明显的一个例子，创建表格对象的程序不能在 AutoCAD 2004 及其以下的版本中使用，因为那些版本中根本没有表格对象）。

关于明经通道

“明经通道”是国内著名的 CAD 技术网站（http://www.mjtd.com），它提供了大量的学习资料和 CAD 程序。网站论坛现有注册用户 30 000 以上，每天都有很多高手在论坛中交流各种技术。

明经通道是免费的技术站点，欢迎各位读者来做客。请记住明经通道的服务宗旨：你的需要，我们创造。

本书的技术支持

可以通过以下三种方式获得关于书中问题的技术支持：

- 在明经通道论坛（http://www.mjtd.com/bbs）的“VBA 和 ActiveX 技术”版块发帖。
- 发送邮件到作者的邮箱（统一发送到 zf9568@263.net）。
- 在作者的个人网站“CAD 大观园”论坛（http://www.cadhelp.net/bbs）的相关版块发帖。

关于本书作者

本书是集体劳动的结晶！下面是每一位作者所撰写的内容：

- 郑立楷（mccad）（明经通道站长，VBA 和 Visual LISP 开发方面的专家）：3.2、3.3、3.5、3.9、5.2、5.3、第 6 章、7.7、11.3、15.4、15.5。
- 王成煌（efan2000）（明经通道“VBA 和 ActiveX 技术”板块版主）：12.3～12.6、13.1、13.3、13.4、15.6。
- 卢择临（ahlzl）（明经通道贵宾，安徽省淮北工业学校教师）：3.1、3.10、3.11、7.6、第 9 章。
- 李倚伏（leeyeafu）（明经通道“数据库/EXCEL/ObjectDBX 编程技术”板块版主）：14.1～14.4。

- 李昭鸿(lzh741206)(明经通道“VBA 和 ActiveX 技术”板块版主):8.4、16.1、配套光盘中的“块内按点生成填充”程序。
- 张帆(zfbj)(明经通道“编程申请”板块版主,“CAD 大观园”站长):第 1 章、第 2 章、3.4、3.6、3.7、3.8、第 4 章、5.1、5.4、7.1～7.5、8.1～8.3、第 10 章、11.1、11.2、11.4、12.1、12.2、12.7、13.2、14.5、15.1～15.3、15.7、16.2、16.3、附录,并负责全书统稿。

本书和《AutoCAD VBA 开发精彩实例教程》

同《AutoCAD VBA 开发精彩实例教程》一书相比较,这本书并不是内容的重复,而是一个提高,前者专注于基础和中级应用,本书则更多地介绍高级技巧,很多内容在帮助系统和市场上同类的图书中是找不到答案的。

致谢

本书能够顺利出版,绝对是集体劳动的成果,明经通道几位在 VBA 方面有造诣的版主和贵宾(有几位因为工作的忙碌没能参与这次的写作)都拿出自己擅长的程序,并亲自撰写文档,确保文档能够最好地反映出程序设计的思想。感谢几位合作者能够在工作忙碌的情况下抽时间高质量地完成写作任务,并且为我的统稿工作提供技术支持,没有你们的努力,这本书不可能有这么高的含金量。

在图书撰写过程中,明经通道的多位网友为本书的编写提出了宝贵的意见并提供了素材资料,例如 wyj7485 网友提供了创建定时器控件的原始素材,更多的网友不再一一列举,在此一并表示感谢,你们的支持是我们前进最重要的动力!

最后要感谢的是本书的读者,感谢您选购这本图书,这是对我们辛苦工作的最好的认可,希望这本书能对你有所帮助,祝读书愉快!

编 者

目　录

第 1 章　AutoCAD VBA 开发基础 …… 1
1.1　宏 …… 1
1.2　集合对象的使用 …… 4

第 2 章　控制 AutoCAD 环境 …… 8
2.1　文档操作 …… 8
2.2　缩放视图 …… 12
2.3　命名视图 …… 14
2.4　和用户交互 …… 17

第 3 章　创建和编辑二维图形对象 …… 25
3.1　根据起点、终点和弧长创建圆弧 …… 25
3.2　利用制图法通过三点创建圆 …… 28
3.3　获得点到多段线的最短距离 …… 31
3.4　选择集 …… 36
3.5　使用 SendCommand 方法 …… 48
3.6　多段线 …… 53
3.7　直线、圆和圆弧转化为多段线 …… 60
3.8　样条曲线转化为多段线 …… 67
3.9　在 AutoCAD 2004 以前版本中计算优化多段线长度 …… 70
3.10　圆管的放样展开 …… 74
3.11　天圆地方的放样展开 …… 78

第 4 章　对象特性 …… 84
4.1　按图层进行缩放 …… 84
4.2　使用 ObjectDBX 技术导入其他图形中的信息 …… 87

第 5 章　文字 …… 98
5.1　创建文字样式 …… 98
5.2　文字按插入点自动对齐 …… 105
5.3　提取多行文字中的文字内容 …… 111
5.4　使用 ObjectDBX 实现批量文字替换 …… 116

第 6 章　标注和公差 …… 124
6.1　标注圆弧长度 …… 124

目　录

第 1 章　AutoCAD VBA 开发基础 …… 1
1.1　宏 …… 1
1.2　集合对象的使用 …… 4

第 2 章　控制 AutoCAD 环境 …… 8
2.1　文档操作 …… 8
2.2　缩放视图 …… 12
2.3　命名视图 …… 14
2.4　和用户交互 …… 17

第 3 章　创建和编辑二维图形对象 …… 25
3.1　根据起点、终点和弧长创建圆弧 …… 25
3.2　利用制图法通过三点创建圆 …… 28
3.3　获得点到多段线的最短距离 …… 31
3.4　选择集 …… 36
3.5　使用 SendCommand 方法 …… 48
3.6　多段线 …… 53
3.7　直线、圆和圆弧转化为多段线 …… 60
3.8　样条曲线转化为多段线 …… 67
3.9　在 AutoCAD 2004 以前版本中计算优化多段线长度 …… 70
3.10　圆管的放样展开 …… 74
3.11　天圆地方的放样展开 …… 78

第 4 章　对象特性 …… 84
4.1　按图层进行缩放 …… 84
4.2　使用 ObjectDBX 技术导入其他图形中的信息 …… 87

第 5 章　文字 …… 98
5.1　创建文字样式 …… 98
5.2　文字按插入点自动对齐 …… 105
5.3　提取多行文字中的文字内容 …… 111
5.4　使用 ObjectDBX 实现批量文字替换 …… 116

第 6 章　标注和公差 …… 124
6.1　标注圆弧长度 …… 124

12.4 显示停靠窗口——Accont 组件 …… 279
12.5 显示幻灯片——Slide 组件 …… 285
12.6 解析表达式——MSScript 组件 …… 289
12.7 创建和使用定时器 …… 294

第 13 章 与其他应用程序和 Windows API 交互 …… 300
13.1 转换 Excel 表格到 AutoCAD 中 …… 300
13.2 使用 Excel 数据绘图 …… 313
13.3 转换 AutoCAD 数据到 Word 和 Excel 中 …… 317
13.4 去掉关闭按钮(X)和添加快捷菜单 …… 326

第 14 章 创建数据库应用程序 …… 332
14.1 创建数据库的 ADO 连接 …… 332
14.2 创建数据表 …… 336
14.3 使用记录集从数据表中读取数据 …… 341
14.4 使用 ADODC 控件 …… 346
14.5 数据记录和 AutoCAD 实体的对应 …… 356

第 15 章 VBA 程序的发布 …… 365
15.1 菜单宏 …… 365
15.2 在工具栏中使用自定义图标 …… 368
15.3 基本图库的实现 …… 375
15.4 在命令行或菜单中执行 VBA 程序 …… 383
15.5 实现 VBA 程序的先选择后操作和透明命令 …… 392
15.6 更改 AutoCAD 应用程序的图标和标题栏 …… 398
15.7 发布 VBA 应用程序 …… 403

第 16 章 VBA 开发相关技术 …… 430
16.1 在 VB 中创建应用程序 …… 430
16.2 搜索某个文件夹中所有的文件 …… 441
16.3 VBA IDE 的编程 …… 448

附录 A AutoCAD ActiveX 和 VBA 参考 …… 457

第 1 章　AutoCAD VBA 开发基础

本章将对 AutoCAD VBA 开发中两个重要的基础知识进行分析，确保用户在后面的程序中能自由地使用 VBA 管理器来创建和管理宏，并且深入理解 AutoCAD 中集合对象的含义，掌握访问集合对象及其元素的方法。

1.1　宏

1.1.1　实例说明

开始学习 AutoCAD VBA 之前，必须熟悉宏及其应用。本节介绍宏的创建、编辑、运行，以及宏的管理等知识。

1.1.2　编程思路

宏是在选择工具栏某个选项时执行特定操作的一系列命令，可以只是完成任务的按键记录，也可以是复杂的命令，例如，AutoLISP、DIESEL 或 ActiveX 编程代码等的组合。宏的定义方式与 VB 中过程(Sub)的定义相一致，一般用来实现某些特定功能，也可作为 VBA 应用程序的启动过程。只有用 Public 关键字声明的过程才能被作为宏。

在 AutoCAD 的“宏”对话框的宏列表中，仅会显示 ThisDrawing 或者标准模块中的宏。ThisDrawing 和标准模块中的代码有何区别？ ThisDrawing 模块中无法创建全局变量，但是可以放置 AutoCAD 事件处理的代码；标准模块与之相反。

严格来说，本节的内容并不在于编程，而是在于介绍与 AutoCAD VBA 开发关系密切的一些概念和开发工具的使用。

1.1.3　步骤

(1) 在 AutoCAD 2005 中，选择“工具”|“宏”|“VBA 管理器”菜单项，系统会弹出“VBA 管理器”对话框，如图 1.1 所示。目前工程列表中未包含任何项目，表示 AutoCAD 2005 没有加载任何的 VBA 工程。

(2) 新建 VBA 工程。单击“新建”按钮，系统会自动添加一个名称为 ACADProject 的全局工程。单击“Visual Basic 编辑器”按钮，进入 VBA 集成开发环境，在“工程资源管理器”窗口中，显示了当前加载的 VBA 工程，如图 1.2 所示。

(3) 修改工程名称。在“工程资源管理器”窗口中选择未命名的工程，然后在“属性”窗口的“名称”一栏中输入“宏的基本概念”，并按 Enter 键，完成工程名称的修改。此时，在 AutoCAD 2005 中打开“VBA 管理器”对话框，就能在工程列表中看到具体的工程名称，如图 1.3 所示。

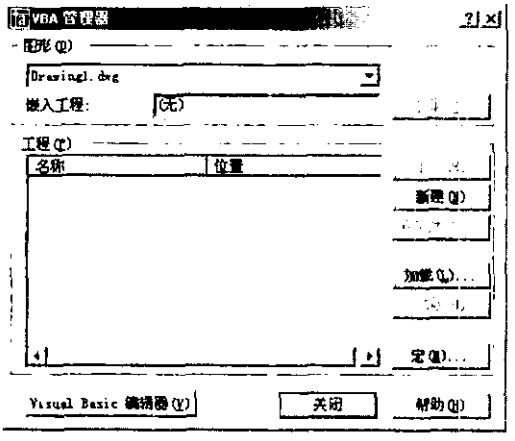


图 1.1 “VBA 管理器”对话框



图 1.2 显示当前加载的 VBA 工程

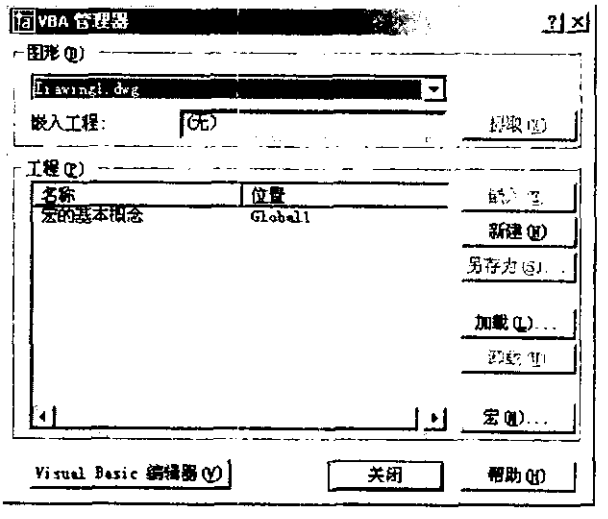


图 1.3 显示修改后的工程名称

提示：在创建新工程之后，应该为其指定新的名称，这样在加载多个工程之后，仍可在“VBA 管理器”对话框和“工程资源管理器”窗口中区分不同的工程。

(4) 保存和另存工程。如果当前的工程尚未保存过，可以在 VBA 集成开发环境中选择“文件”|“保存 Global1”菜单项，或者单击“标准”工具栏中的“保存”按钮，在系统弹出的“另存为”对话框中指定工程的位置，完成工程的保存。

此外，在“VBA 管理器”对话框中，从工程列表中选择所要保存的工程，单击“另存为”按钮，在弹出的“另存为”对话框中，同样可以完成工程的保存。要实现工程的换名保存(另存)，就必须使用这种方法。

(5) 打开已有的 VBA 工程。在“VBA 管理器”对话框中，单击“加载”按钮，在系统弹出的“打开 VBA 工程”对话框中，选择所要加载的 VBA 工程，单击“打开”按钮，即可将其加载到 AutoCAD 中。

(6) 关闭 VBA 工程。在“VBA 管理器”对话框中，选择所要关闭的工程，单击“卸载”按

钮，就能关闭指定的 VBA 工程。

(7) 输入示例代码。进入 VBA 集成开发环境，双击工程资源管理器的 ThisDrawing 图标，在弹出的代码窗口中输入如下所示的代码。

```
Public Sub MacroDemo( )
    MsgBox "Hello,Welcome to AutoCAD VBA!"
End Sub
```

(8) 运行宏。在 VBA 集成开发环境中，将光标定位到要运行的宏内部(也就是位于 Sub 和 End Sub 语句之间的部分)，按 F5 键就可以执行宏。另外，也可选择“工具”|“宏”菜单项，在系统弹出的“宏”对话框中运行宏。

1.1.4　实例效果

在 VBA 集成开发环境中，选择“工具”|“宏”菜单项，系统弹出如图 1.4 所示的“宏”对话框。从宏列表中选择本节创建的 MacroDemo 宏，单击“运行”按钮，执行该宏。

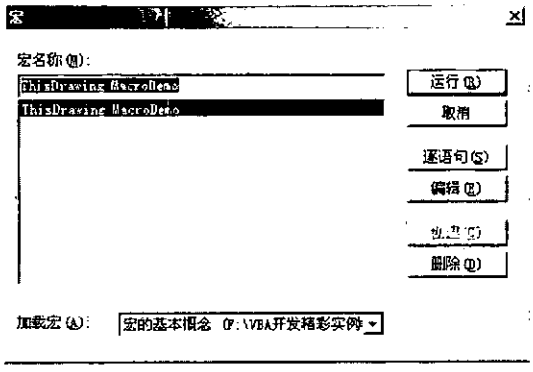


图 1.4　选择所要执行的宏

系统会弹出一个信息提示框，如图 1.5 所示。



图 1.5　运行宏的结果

1.1.5　小结

宏是 VBA 中重要的概念，在 AutoCAD VBA 开发中，一般被用作某个功能的入口点，类似于 VB 中启动对象(Sub Main)的概念。

除了在 VBA 集成开发环境中创建和执行宏之外，还可以在 AutoCAD 2005 中选择“工具”|“宏”|“宏”菜单项，打开如图 1.6 所示的“宏”对话框。可以看出，在“宏”对话框

中同样可以创建和运行宏，但是这多用于应用阶段，也就是说程序的使用者多用这种方法来运行宏。

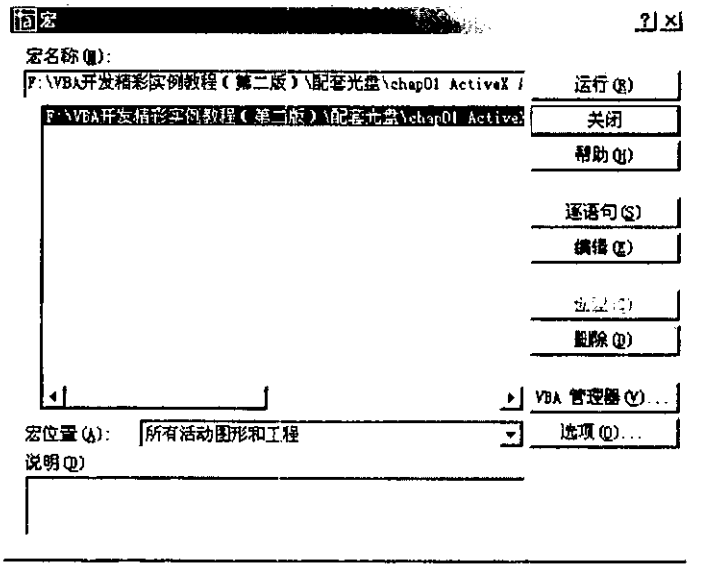


图 1.6 在“宏”对话框中运行宏

1.2 集合对象的使用

1.2.1 实例说明

集合对象是一种预定义的对象，包含相似对象的所有实例(或是相似对象所有实例的上级对象)。在 ActiveX 接口中，提供了大量的集合对象，重要的集合对象包括 Documents(当前 AutoCAD 任务中打开的所有文档)、ModelSpace(模型空间中的所有图形对象)、Blocks(图形中所有的块)、Groups(图形中的所有编组)、Layers(图形中的所有图层)、Linetypes(图形中的所有线型)、SelectionSets(图形中所有的选择集)、TextStyles(图形中所有的文字样式)、UCSs(图形中所有的用户坐标系)、Views(图形中所有的视图)和 Viewports(图形中所有的视口)。

本节介绍访问集合、集合对象，添加和删除集合对象，以及遍历集合对象的知识。

1.2.2 编程思路

大部分的集合对象都是通过 Document 对象访问的，对于每一个集合对象，Document 对象都有一个相应的特性。例如，可以定义一个变量，并将其设置为当前图形的 Groups 集合：

```
Dim groupCollection As AcadGroups
Set groupCollection = ThisDrawing.Groups
```

还有一些集合对象是通过 Application 对象来访问的，例如 Documents、MenuBar 和

MenuGroups。对于这些对象，在 Application 对象中包含了相应的特性。

要向集合中添加新的元素，可以使用 Add 方法，例如：

```
Dim newGroup As AcadGroup
Set newGroup = ThisDrawing.Groups.Add("MyGroup")
```

要访问集合中特定的某个元素，可以使用集合的 Item 方法，既可以通过集合元素的名称来访问，也可以通过元素的索引值来访问特定的元素，例如：

```
Dim objGroup As AcadGroup
Set objGroup = ThisDrawing.Groups.Item("MyGroup")
```

要从集合中删除某个元素，可以使用元素本身的 Delete 方法，例如：

```
Dim objGroup As AcadGroup
Set objGroup = ThisDrawing.Groups.Item("MyGroup")
objGroup.Delete
```

遍历集合对象是指依次访问集合中的每一个元素，一般可以使用 VB 中的 For...Each 语句来完成。需要注意的是，不要在使用 For...Each 机制遍历集合的同时对任意对象使用图元编辑方法(例如复制、阵列、镜像等)。应在遍历完成后再尝试编辑集合中的对象，或者创建临时数组，并将其设置为等于该集合，然后遍历复制的数组并执行所需的编辑操作。

1.2.3　步骤

下面以图层的操作为例，介绍一下集合对象的使用。首先创建一个新工程，本节涉及到的所有代码均在 ThisDrawing 模块中添加。

(1) 新建一个图层，也就是向图层集合添加一个元素。使用 Add 方法，向当前图形的 Layers 集合添加一个新元素，其相关代码为：

```
Public Sub NewLayer( )
    '新建一个图层
    Dim objLayer As AcadLayer
    Dim layerCollection As AcadLayers

    Set layerCollection = ThisDrawing.Layers
    Set objLayer = layerCollection.Add("轴线")
End Sub
```

(2) 修改“轴线”层的名称。要修改某个特定图层的特性，必须首先声明一个变量，并且将该变量指向这个图层，其相关代码为：

```
Public Sub ChangeLayerName( )
    Dim objlayer As AcadLayer

    Set objlayer = ThisDrawing.Layers.Item("轴线")
    objlayer.Name = "中心线"
End Sub
```

(3) 统计图形中名称的前两个字符为“图层”的图层的数量。要实现这个功能，必须遍

历图形中所有的图层，将图层名称的前两个字符与“图层”字符串相比较，两者相等时将统计值加 1。其相关代码为：

```
Public Sub ChangeAllName( )
    Dim objlayer As AcadLayer
    Dim i As Integer

    For Each objlayer In ThisDrawing.Layers
        If StrComp(Left$(objlayer.Name, 2), "图层") = 0 Then
            i = i + 1
        End If
    Next objlayer

    Debug.Print "有" & i & "个图层需要修改名称"
End Sub
```

StrComp 函数能够比较两个字符串的内容，如果两字符串相同，返回值为 0。Left$ 函数返回一个字符串，其中包含字符串中从左边算起指定数量的字符。Debug.Print 语句会在立即窗口中打印指定的文本。在 VBA 集成开发环境中，选择“视图”|“立即窗口”菜单项，能够显示立即窗口。

(4) 删除某个图层。如果图形中存在“轴线”层，就将其删除，其相关代码为：

```
Public Sub DeleteLayer( )
    On Error Resume Next

    Dim objlayer As AcadLayer
    Set objlayer = ThisDrawing.Layers.Item("轴线")

    If Not IsEmpty(objlayer) Then
        objlayer.Delete
    End If
End Sub
```

Not 运算符用来对表达式进行逻辑否定运算，IsEmpty 函数判断指定的变量是否已经初始化。之所以要使用 On Error Resume Next 语句进行错误处理(如果遇到错误，程序会转到紧接着发生错误的语句后面的语句，继续执行)，是因为可能在当前图形中并不存在“轴线”图层，那么访问 Layers 集合中名为“轴线”的元素就会发生错误。

1.2.4 实例效果

在 VBA 集成开发环境中，将光标定位在要执行的宏内部，按 F5 键运行该宏，然后选择“格式”|“图层”菜单项，在图层特性管理器中观察宏的运行结果。

1.2.5 小结

在 VBA 编程中，经常用的操作是遍历图形数据库，获得图形数据库中满足特定条件的实体。例如，要将图形中所有的直线颜色变为红色，就可以使用下面的一段程序。

```
Public Sub ChangeLineColor( )
    Dim ent As AcadEntity
    Dim color As New AcadAcCmColor

    color.ColorIndex = acRed

    For Each ent In ThisDrawing.ModelSpace
        If TypeOf ent Is AcadLine Then
            ent.TrueColor = color
        End If
    Next ent
End Sub
```

第2章　控制 AutoCAD 环境

本章包括文档操作、缩放视图、命名视图，以及和用户交互的操作。文档操作是指图形文件的创建、打开、关闭和管理等，是典型的集合对象操作；缩放视图使用 ActiveX 模型中提供的缩放方法模拟了 AutoCAD 自身的几种缩放功能；“命名视图”一节探讨了在 VBA 程序中保存当前图形窗口的视图和设置当前视图的方法；“和用户交互”部分提出了在 GetPoint 方法中使用关键字的解决方案。

2.1　文 档 操 作

2.1.1　实例说明

在 VBA 开发中，经常需要进行图形的创建、打开、关闭和管理，这些操作都是通过 Documents 集合和 Document 对象来实现的。本节的实例演示创建新图形、打开一个图形、保存图形、关闭图形和设置当前文档。

2.1.2　编程思路

在 VBA 中对图形文档的操作，主要是通过 Documents 集合和 Document 对象的特性来实现的。

Documents 集合的 Add 方法能够创建一个新图形，并将其添加到 Documents 集合中；Open 方法能够打开现有的图形；Close 方法可以关闭 AutoCAD 打开的所有图形。

Document 对象的 Save 和 SaveAs 方法用于保存图形；在关闭图形之前，可以使用 Saved 属性来判断当前图形中是否有未保存的修改。

2.1.3　步骤

创建新工程之后，打开 ThisDrawing 窗口，本节所有的代码均在该模块中添加。

(1) 创建新图形。使用 Documents 集合的 Add 方法可以创建新图形，并将图形添加到 Documents 集合中，其实现代码为：

```
Public Sub NewDocument( )
    Dim doc As AcadDocument
    '新建一个文档
    Set doc = ThisDrawing.Application.Documents.Add( )
End Sub
```

(2) 保存一个图形。Document 对象的 Save 和 SaveAs 方法用于保存图形，使用 Saved 属性可以检查当前图形是否存在未保存的修改。保存图形的示例代码为：

```
Public Sub SaveDocument( )
    Dim doc As AcadDocument
    Set doc = ThisDrawing.Application.Documents.Add( )

    '创建一条直线
    Dim ptStart(0 To 2) As Double
    Dim ptEnd(0 To 2) As Double
    ThisDrawing.ModelSpace.AddLine ptStart, ptEnd

     保存修改后的文档
    doc.SaveAs "c:\test.dwg"
End Sub
```

SaveAs 方法的定义为：

```
object.SaveAs FileName, FileType [, SecurityParams]
```

在保存图形的时候，可以指定保存图形的版本和格式，还可以设置图形的安全选项。

(3) 关闭当前图形。使用 Documents 集合或者 Document 对象的 Close 方法可以关闭一个图形，参考下面的示例代码：

```
Public Sub CloseDocument( )
    ThisDrawing.Application.ActiveDocument.Close False
End Sub
```

在关闭图形文档的时候，可以指定是否保存图形。如果要求保存图形，还可以指定图形保存的名称和位置。

(4) 打开已有的图形。使用 Documents 集合的 Open 方法可以打开一个已有的图形，参考下面的示例代码：

```
Public Sub OpenDocument( )
    If Len(Dir("c:\test.dwg")) <> 0 Then
        ThisDrawing.Application.Documents.Open "c:\test.dwg"
    Else
        MsgBox "指定的文件不存在!"
    End If
End Sub
```

Dir 函数会返回与指定文件名称匹配的第一个文件名，如果返回值为零长度字符串，表示指定的文件不存在。Len 函数能够计算字符串内字符的数量。

(5) 设置当前文档。使用 Application 对象的 ActiveDocument 属性来设置当前文档，参考下面的示例代码：

```
Public Sub ActiveDocument( )
    Dim oldDoc As AcadDocument
    Set oldDoc = ThisDrawing.Application.ActiveDocument
```

```
    '打开一个新文档
    If Len(Dir("c:\test.dwg")) <> 0 Then
        ThisDrawing.Application.Documents.Open "c:\test.dwg"
    Else
        MsgBox "指定的文件不存在!"
    End If

    '设置当前文档
    ThisDrawing.Application.ActiveDocument = oldDoc
End Sub
```

(6) 使用带有预览图像的“选择文件”对话框。VBA 没有直接的方法来访问“选择文件”对话框，但是可以通过 AutoLISP 中提供的方法来实现，AutoLISP 中有一个函数 getfiled，该函数能够访问可以预览 DWG 图形的标准文件对话框。创建 OpenDialog 宏来实现该功能：

```
Public Sub OpenDialog( )
    Dim strFileName As String

    '确保 USERS1 系统变量为空
    ThisDrawing.SetVariable "USERS1", ""

    '显示“选择文件”对话框，并且获得用户选择的结果
    ThisDrawing.SendCommand "(setvar " & """USERS1""" & "(getfiled " & """选择图形文件""" & ""
        "c:/""" & """dwg""" & "8)) "
    strFileName = ThisDrawing.GetVariable("USERS1")

    '显示选择结果
    If Len(strFileName) = 0 Then
        MsgBox "未选择任何图形文件!", vbInformation, "选择结果"
    Else
        MsgBox "选择的文件是：" & strFileName, vbInformation, "选择结果"
    End If
End Sub
```

SetVariable 方法能够设置 AutoCAD 系统变量的值，USERS1 系统变量用于存储和检索文字字符串数据，为开发者暂时保存字符串提供了一种很好的方法。如果在使用 USERS1 系统变量之前不将其值清空，那么当用户在“选择文件”对话框中单击“取消”按钮时，就有可能会导致 strFileName 变量获得上一次使用该系统变量时保存的字符串。因此，应在使用 USERS1 系统变量之前或之后及时清空它。

SendCommand 方法用于向 AutoCAD 命令行发送命令字符串，这就提供了一种和 AutoLISP 交互的手段，实际上，VBA 中几乎所有的调用 AutoLISP 的处理都是通过 SendCommand 方法来实现的。在这里，SendCommand 方法的参数是一个 LISP 字符串，其中使用了两个 AutoLISP 函数：

- setvar：AutoLISP 中用于设置 AutoCAD 系统变量的函数，其使用格式为“(setvar

0839459

"filletrad" 1)"。

- getfiled：显示 AutoCAD 标准对话框，提示用户选择一个文件。关于该函数的格式和使用方法，可以参考《AutoCAD 2005 帮助：开发人员文档》中的相关内容。

GetVariable 方法能够获得 AutoCAD 系统变量的值，在此能够获得 SendCommand 方法中保存在 USERS1 系统变量中的字符串。

2.1.4　实例效果

(1) 在 VBA 集成开发环境中，将光标定位在 OpenDialog 宏内部，按 F5 键执行该宏，系统会弹出如图 2.1 所示的对话框。

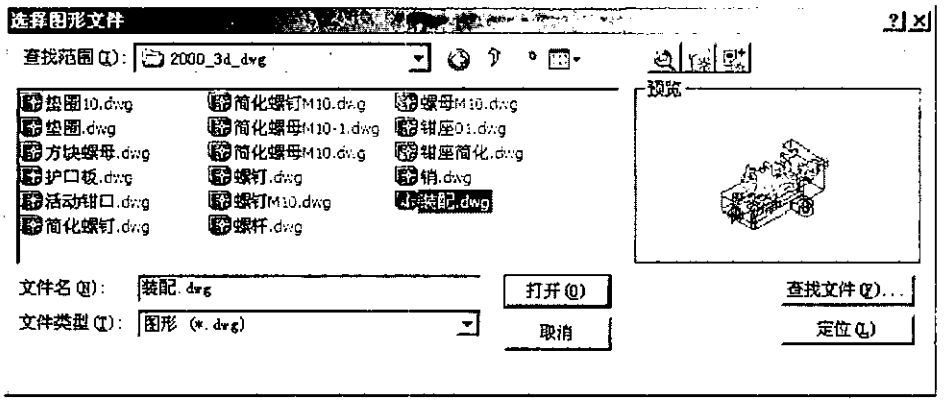


图 2.1　"选择图形文件"对话框

(2) 选择一个图形文件，单击"打开"按钮，系统会弹出如图 2.2 所示的结果。

(3) 如果在"选择图形文件"对话框中单击"取消"按钮，系统会弹出如图 2.3 所示的消息框。

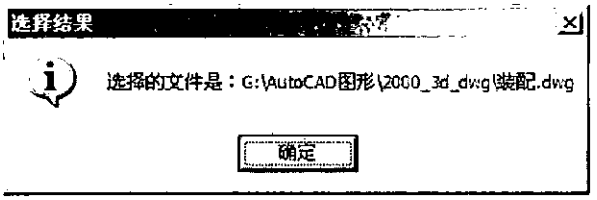


图 2.2　显示选择的文件名称

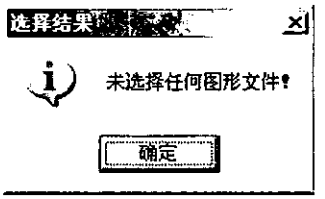


图 2.3　提示未选择图形文件

本节的其他实例都是基本实例，其演示效果不再详细介绍。

2.1.5　小结

利用本节所介绍的功能，读者可以自行创建一个工程管理器，在其中可以用一个树状视图来管理当前工程所用到的所有文件，将图形文件按照规则分类，并实现文档的打开、关闭、保存等操作。

在开发过程中涉及到的树状视图的使用可以参考 VB 的相关图书，或者 MSDN 中 ActiveX 控件参考的部分。

2.2 缩放视图

2.2.1 实例说明

在绘图过程中经常用到缩放、平移等改变视图的操作，在编程过程中同样可能会使用。本节实例将演示全部缩放、范围缩放、窗口缩放、返回上一个缩放窗口、比例缩放和中心缩放。

2.2.2 编程思路

在 Application 对象中提供了一系列用于缩放视图的方法，包括 ZoomAll、ZoomExtents、ZoomWindow、ZoomPickWindow、ZoomPrevious、ZoomScaled 和 ZoomCenter。

执行某种缩放，既可以按照 ActiveX 对象模型的层次结构来访问：

```
ThisDrawing.Application.ZoomPrevious
```

也可以直接输入相应的方法：

```
ZoomPrevious
```

2.2.3 步骤

(1) 在 VBA 集成开发环境中，创建一个新工程，打开 ThisDrawing 窗口。添加一个宏，用于实现全部缩放的功能：

```
Public Sub ZoomAll( )
    ThisDrawing.Application.ZoomAll
End Sub
```

(2) 使用 ZoomExtents 方法，实现范围缩放的功能：

```
Public Sub ZoomExtents( )
    ThisDrawing.Application.ZoomExtents
End Sub
```

“全部缩放”和“范围缩放”是两个不同的概念。在平面视图中，“全部缩放”将图形缩放到图形界限或当前范围两者中较大的区域中，而“范围缩放”则会缩放以显示图形范围，并使所有对象最大显示。在三维视图中，两者的作用完全相同。

(3) 使用 ZoomWindow 方法，能够通过指定区域的角点，将该区域作为图形窗口显示的范围，参考下面的代码：

```
Public Sub ZoomWindow( )
    '提示用户拾取两个角点
    Dim ptLeft(0 To 2) As Double
    Dim ptRight(0 To 2) As Double
```

```
    ptLeft(0) = 0: ptLeft(1) = 0: ptLeft(2) = 0
    ptRight(0) = 100: ptRight(1) = 100: ptRight(2) = 0

    ThisDrawing.Application.ZoomWindow ptLeft, ptRight
End Sub
```

(4) ZoomWindow 方法使用户可以通过编程的方式定义代表“缩放”窗口的两个角点；ZoomPickWindow 方法则要求用户拾取两个点，这两个拾取点将形成“缩放”窗口。关于 ZoomPickWindow 方法的使用可以参考下面的代码：

```
Public Sub ZoomPickWindow( )
    ThisDrawing.Application.ZoomPickWindow
End Sub
```

(5) 要返回上一个缩放窗口，可以使用 ZoomPrevious 方法，其相关代码为：

```
Public Sub ZoomPrevious( )
    ThisDrawing.Application.ZoomPrevious
End Sub
```

(6) 在 AutoCAD 中，用户可以输入三种类型的比例因子来实现不同的比例缩放。输入的值后面跟着 x，根据当前视图指定比例；输入值后跟 xp，指定相对于图纸空间单位的比例；直接输入数值，指定相对于图形界限的比例(此选项使用较少)。

这里使用 ZoomScaled 方法，并且判断用户输入的比例因子，模拟 AutoCAD 应用中的比例缩放，其相关代码为：

```
Public Sub ZoomScale( )
    Dim strInput As String
    Dim scaleZoom As Double
Retry:
    strInput = ThisDrawing.Utility.GetString(False, "输入缩放比例: ")

    Dim zoomType As Integer        '缩放比例的类型
    If StrComp(Right$(strInput, 1), "x", vbTextCompare) <> 0 And StrComp(Right
        $(strInput, 2), "xp", vbTextCompare) <> 0 Then
        zoomType = 0
        scaleZoom = CDbl(strInput)
    ElseIf StrComp(Right$(strInput, 1), "x", vbTextCompare) = 0 Then
        zoomType = 1
        scaleZoom = CDbl(Left$(strInput, Len(strInput) - 1))
    ElseIf StrComp(Right$(strInput, 2), "xp", vbTextCompare) = 0 Then
        zoomType = 2
        scaleZoom = CDbl(Left$(strInput, Len(strInput) - 2))
    End If

    ThisDrawing.Application.ZoomScaled scaleZoom, zoomType
End Sub
```

上面的代码中，使用 GetString 方法来接收用户输入的比例因子，使用 Right $ 函数来截取用户输入比例因子的后几个字符，与 xp 和 x 比较来判断用户所使用的比例因子类型。判断出比例因子的类型之后，要将标识字符(x 或 xp)从比例因子中去除，获得比例因子的数值，并使用 CDbl 函数将其转化成实数。

(7) 使用 ZoomCenter 方法来实现中心缩放，其相关代码为：

```
Public Sub ZoomCenter( )
    Dim ptCenter(0 To 2) As Double
    Dim magnification As Double          图形窗口的高度(用图形单位来表示)

    ptCenter(0) = 100
    ptCenter(1) = 100
    ptCenter(2) = 0
    magnification = 20

    ThisDrawing.Application.ZoomCenter ptCenter, magnification
End Sub
```

2.2.4 实例效果

本节所实现的功能与 AutoCAD 中相应的缩放操作一致，仅为了演示缩放视图的方法，程序运行的具体效果读者可自行验证。

2.2.5 小结

在 AutoCAD 中，用户可以通过缩放或者平移来改变视图，但是在编程时很少使用平移，在需要的场合也可以用缩放操作来完成，因此本节并没有实现平移的操作。

2.3 命名视图

2.3.1 实例说明

在 AutoCAD 操作的过程中，需要经常改变视图。某些特殊的视图可能会经常使用，可以将其保存起来，并在需要的时候直接恢复该视图。

本节实例将演示视图的保存和恢复，在保存视图时，由于 ActiveX 对象模型中并没有提供直接的方法和属性，必须根据 AutoCAD 的系统变量值来获取需要的参数值。

2.3.2 编程思路

本节要实现的目标有两个，保存当前图形窗口的视图和应用某个已经保存的视图。要恢复某个已经保存的视图，可以直接使用 ViewPort 对象的 SetView 方法。对于保存当前图形窗口的视图，系统没有提供直接的方法来实现，必须通过获取系统变量的值，从而获得

当前视图的所有参数，根据这些参数创建一个视图对象，将其作为当前视图的副本进行保存。

获得系统变量的方法是 GetVariable，其定义为：

```
...tVal = object.GetVariable(Name)
```

其中，Name 为所要获得值的系统变量名称，RetVal 中包含了系统变量的值。

设置系统变量的方法是 SetVariable，其定义为：

```
object.SetVariable Name, Value
```

其中，Name 为所要设置值的系统变量名称，Value 中包含所要设置的值。

2.3.3 步骤

(1) 在 VBA 集成开发环境中，创建一个新工程，打开 ThisDrawing 窗口。添加用于保存当前视图的代码：

```
Public Sub SaveCurrentView( )
    Dim view As AcadView
    Set view = ThisDrawing.Views.Add("CurView")
    Dim retVar As Variant

    '获得视图的中心点
    retVar = ThisDrawing.GetVariable("VIEWCTR")
    Dim ptCenter(0 To 1) As Double
    Dim center As Variant
    center = ThisDrawing.Utility.TranslateCoordinates(retVar, acUCS, acDisplayDCS, False)
    ptCenter(0) = center(0): ptCenter(1) = center(1)
    view.center = ptCenter

    '当前视口中目标点的位置(以 UCS 表示)
    retVar = ThisDrawing.GetVariable("TARGET")
    Dim ptTarget As Variant
    ptTarget = ThisDrawing.Utility.TranslateCoordinates(retVar, acUCS, acWorld, False)
    view.Target = ptTarget

    '当前视口的观察方向
    retVar = ThisDrawing.GetVariable("VIEWDIR")
    Dim vecDirection As Variant
    vecDirection = ThisDrawing.Utility.TranslateCoordinates(retVar, acUCS, acWorld, True)
    view.Direction = vecDirection

    '当前视口的视图高度(图形单位)
    retVar = ThisDrawing.GetVariable("VIEWSIZE")
    view.height = retVar
```

```
    '当前视口的视图宽度
    Dim heightWidth As Variant      '高宽比
    heightWidth = ThisDrawing.GetVariable("SCREENSIZE")
    view.Width = heightWidth(0)/heightWidth(1) * retVar
End Sub
```

关于代码中所用的系统变量的具体含义，可以参考 AutoCAD 帮助系统中的系统变量说明，如图 2.4 所示。

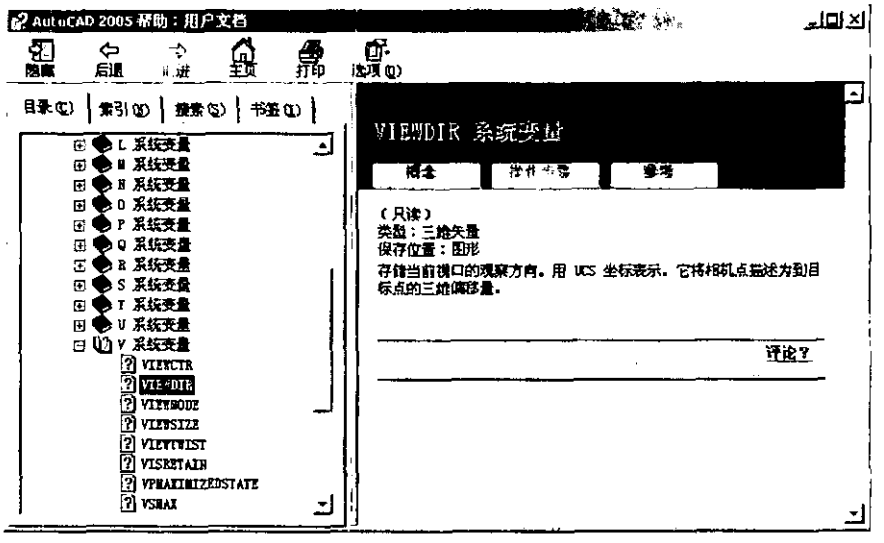


图 2.4 查看系统变量说明

(2) 将已经保存的某个视图作为当前视图，其相关代码为：

```
Public Sub SetCurrentView( )
    '获得当前活动视口
    Dim viewportObj As AcadViewport
    Set viewportObj = ThisDrawing.ActiveViewport

    '在活动视口中设置视图
    viewportObj.SetView ThisDrawing.Views.Item("CurView")
    ThisDrawing.ActiveViewport = viewportObj

    ThisDrawing.Regen acActiveViewport
End Sub
```

2.3.4 实例效果

在 AutoCAD 2005 中打开一幅图形，对其进行适当的缩放，运行 SaveCurrentView 宏，保存当前的视图。继续对其进行缩放、平移、三维旋转等改变视图的操作，运行 SetCurrentView 宏恢复前面保存的视图，得到的视图与改变视图之前的结果相同。

2.3.5　小结

在 AutoCAD 二次开发中，系统变量起着重要的作用，这里给出几个经常在开发中使用的系统变量，以供参考。

- ACADVER：存储 AutoCAD 的版本号。
- ATTDIA：控制 INSERT 命令是否使用对话框用于属性值的输入。
- FILEDIA：控制“文件定位”对话框和“创建新图形”对话框的显示。
- FILLMODE：指定图案填充（包括实体填充和渐变填充）、二维实体和宽多段线是否被填充。
- MENUNAME：存储菜单文件名，包括文件名路径。
- PDMODE：控制如何显示点对象。
- PDSIZE：设置点对象的显示大小。
- TILEMODE：将模型选项卡或上一个布局选项卡置为当前。
- UCSNAME：存储当前空间当前视口的当前坐标系名称。如果当前 UCS 尚未命名，则返回一个空字符串。

2.4　和用户交互

2.4.1　实例说明

ActiveX 对象模型中提供了多种提示用户输入的方法，包括 GetString、GetPoint、GetInteger、GetKeyword 和 GetReal 等。这些方法本身的使用很简单，参照帮助系统提供的示例代码就可以学会，但是本节所要讨论的主要问题是 GetPoint 方法和关键字的结合应用，例如在 AutoCAD 中执行 PLINE 命令时，能够得到如下的命令提示：

```
命令：_pline
指定起点：
当前线宽为 0.0000
指定下一个点或 [圆弧(A)/半宽(H)/长度(L)/放弃(U)/宽度(W)]：
指定下一点或 [圆弧(A)/闭合(C)/半宽(H)/长度(L)/放弃(U)/宽度(W)]：
```

其中，“指定下一个点或 [圆弧(A)/半宽(H)/长度(L)/放弃(U)/宽度(W)]：”就将提示用户输入点和关键字结合在一起。

本节的实例将要模拟 AutoCAD 中动态创建多段线的效果，并且允许用户通过命令行的关键字来改变多段线的某些特性。

2.4.2　编程思路

1. 动态创建多段线

动态创建多段线，最基本的要求是用户在图形窗口中按顺序拾取多个顶点，每次拾取一点都会将其添加到多段线的末尾，最后按 Enter 键或者 Esc 键完成多段线的创建。

根据这个思路，可以编写下面的代码来完成动态创建多段线：

```
Public Sub CreatePolylineBasic( )
    On Error Resume Next

    Dim index As Integer                                '当前输入点的次数
    index = 2

    '提示用户输入第一点
    Dim pt1 As Variant
    pt1 = ThisDrawing.Utility.GetPoint(, "输入第一点：")
    If Err Then                                         '处理 Esc 键或者 Enter 键的事件
        Err.Clear
        Exit Sub
    End If

    Dim ptPrevious As Variant, ptCurrent As Variant     '拾取点过程中，存储上一点和当前点的变量
    ptPrevious = pt1

NEXTPOINT:
    ptCurrent = ThisDrawing.Utility.GetPoint(ptPrevious, "输入下一点：")

    If Err Then                                         '处理 Esc 键或者 Enter 键的事件
        Err.Clear
        Exit Sub
    End If

    Dim objPline As AcadLWPolyline
    If index = 2 Then
        '创建多段线
        Dim points(0 To 3) As Double
        points(0) = ptPrevious(0)
        points(1) = ptPrevious(1)
        points(2) = ptCurrent(0)
        points(3) = ptCurrent(1)
        Set objPline = ThisDrawing.ModelSpace.AddLightWeightPolyline(points)
    ElseIf index > 2 Then
        Dim ptVert(0 To 1) As Double
        ptVert(0) = ptCurrent(0)
        ptVert(1) = ptCurrent(1)
        objPline.AddVertex index - 1, ptVert
    End If

    index = index + 1
    ptPrevious = ptCurrent
```

```
    GoTo NEXTPOINT                                    提示用户继续拾取顶点
End Sub
```

上面的代码有一点不太好理解，因为 GetPoint 方法无法返回表示执行成功或失败的结果代码，只能根据在执行过程中是否发生错误来判断。

其算法可以描述为：

(1) 拾取第一点。

(2) 拾取第二点，创建多段线。

(3) 如果用户没有按 Enter 键或者 Esc 键，拾取下一点，并将拾取的点添加到多段线的末尾。

(4) 如果用户按 Enter 键或者 Esc 键，退出程序执行，完成多段线的创建，否则反复执行步骤(3)。

2. GetPoint 中使用关键字

要在 GetPoint 中使用关键字，必须在使用 GetPoint 方法之前定义关键字，并且使用 InitializeUserInput 方法来设置关键字，并紧跟这一句调用 GetPoint 方法。下面的代码演示了在 GetPoint 中使用关键字的方法。

```
Public Sub GetPointWithKeyword( )
    On Error Resume Next

    '定义有效的关键字
    Dim keywordList As String
    keywordList = "Keyword1 Keyword2"

    '调用 InitializeUserInput 方法来设置关键字
    ThisDrawing.Utility.InitializeUserInput 128, keywordList

    '获得用户的输入
    Dim returnPnt As Variant
    returnPnt = ThisDrawing.Utility.GetPoint(, "输入一个点(Keyword1, Keyword2): ")
    If Err Then
        If StrComp(Err.Description, "用户输入的是关键字", 1) = 0 Then
        '如果一个用户输入的是一个关键字
            Dim inputString As String
            Err.Clear
            inputString = ThisDrawing.Utility.GetInput
            MsgBox "你输入的关键字: " & inputString
        Else
            MsgBox "选择点时发生错误: " & Err.Description
            Err.Clear
        End If
    Else
        '显示拾取点的坐标
        MsgBox "你选择的点的世界坐标为: " & returnPnt(0) & ", " & returnPnt(1) & ", " &
          returnPnt(2), , "GetPoint 中使用关键字"
```

```
    End If
End Sub
```

要获得 GetPoint 方法执行过程中用户输入的关键字，同样只能在错误处理中进行，通过 GetInput 方法获得输入的内容。由于必须在错误发生时捕获错误，并从错误代码中判断用户输入了关键字，因此代码开始处的 On Error Resume Next 语句不可缺少。

2.4.3 步骤

(1) 在 VBA 集成开发环境中，创建一个新工程，保存在适当的位置，打开 ThisDrawing 模块的窗口。创建一个函数 GetWidth，用于获取用户输入的线宽，其实现代码为：

```
Public Function GetWidth( ) As Double
    On Error Resume Next

    Dim width As Double
    width = ThisDrawing.Utility.GetReal("输入线宽：")

    If Err Then
        width = -1
    End If

    GetWidth = width
End Function
```

系统提示“输入线宽：”时，如果用户没有正确输入线宽，会引发一个错误，错误的结果用返回值－1 来标识。很明显，对于线宽来说，－1 是不可能的结果。

(2) 创建一个函数 GetColorIndex，用于提示用户输入颜色索引值，其实现代码为：

```
Public Function GetColorIndex( ) As Integer
    On Error Resume Next

    Dim colorIndex As Integer
    colorIndex = ThisDrawing.Utility.GetInteger("输入颜色索引值：")

    If Err Then
        colorIndex = -1
    End If

    GetColorIndex = colorIndex
End Function
```

这个函数的结构与 GetWidth 完全相同。还可以给这个函数加上更完善的错误处理机制，因为颜色索引值的有效范围是 0～256。

(3) 在 ThisDrawing 模块的开始处添加下面的声明：

```
Private Const VK_ESCAPE = &H1B                          '代表 Esc 键
Private Declare Function GetAsyncKeyState Lib "user32" (ByVal vKey As Long) As Integer
```

GetAsyncKeyState 是一个 Windows API 函数，用于判断函数调用时指定虚拟键的状态。其返回值是 Long 类型，自对 GetAsyncKeyState 函数的上一次调用以来，如指定的虚拟键被按过，则位 0 设为 1；否则设为 0。

(4) 创建一个函数 CheckKey，用于判断某一个键是否在调用前被按下，其实现代码为：

```
Private Function CheckKey(lngKey As Long) As Boolean
    If GetAsyncKeyState(lngKey) Then
        CheckKey = True
    Else
        CheckKey = False
    End If
End Function
```

(5) 编写模拟绘制多段线的代码，将其放置在 CreatePolyline 宏中，其实现代码为：

```
Public Sub CreatePolyline( )
    On Error Resume Next

    Dim colorIndex As Integer                    '多段线的颜色索引号
    Dim width As Double                          '多段线的线宽
    colorIndex = 0
    width = 0

    Dim index As Integer                         '当前输入点的次数
    index = 2

    '提示用户输入第一点
    Dim pt1 As Variant
    pt1 = ThisDrawing.Utility.GetPoint(, "输入第一点：")
    If Err Then
        Err.Clear
        Exit Sub
    End If

    Dim ptPrevious As Variant, ptCurrent As Variant  '拾取点过程中，存储上一点和当前点的变量
    ptPrevious = pt1

    '定义有效的关键字
    Dim strKeyWords As String
    strKeyWords = "W C O"

NEXTPOINT:
    '设置关键字
    ThisDrawing.Utility.InitializeUserInput 128, strKeyWords
    ptCurrent = ThisDrawing.Utility.GetPoint(ptPrevious, "输入下一点 [宽度(W)/颜色(C)]
        <完成(O)>：")
```

```
If Err Then                                          '在错误处理中判断用户输入的关键字
    If StrComp(Err.Description, "用户输入的是关键字", 1) = 0 Then
        Dim strInput As String
        strInput = ThisDrawing.Utility.GetInput
        Err.Clear

        '根据输入的关键字进行相应的处理
        If StrComp(strInput, "W", vbTextCompare) = 0 Then
            '获得用户输入的宽度值,然后继续提示用户拾取顶点
            width = GetWidth
            GoTo NEXTPOINT
        ElseIf StrComp(strInput, "C", vbTextCompare) = 0 Then
            '获得用户输入的颜色索引值,然后继续提示用户拾取顶点
            colorIndex = GetColorIndex
            GoTo NEXTPOINT
        ElseIf StrComp(strInput, "O", vbTextCompare) = 0 Or Len(strInput) = 0 Then
            '完成多段线的创建
            Exit Sub
        End If
    Else
        Err.Clear
        '判断用户是否按了 Esc 键
        If CheckKey(VK_ESCAPE) = True Then
            Exit Sub
        End If
    End If
End If

Dim objPline As AcadLWPolyline
If index = 2 Then
    '创建多段线
    Dim points(0 To 3) As Double
    points(0) = ptPrevious(0)
    points(1) = ptPrevious(1)
    points(2) = ptCurrent(0)
    points(3) = ptCurrent(1)
    Set objPline = ThisDrawing.ModelSpace.AddLightWeightPolyline(points)
ElseIf index > 2 Then
    Dim ptVert(0 To 1) As Double
    ptVert(0) = ptCurrent(0)
    ptVert(1) = ptCurrent(1)
    objPline.AddVertex index - 1, ptVert
End If

'修改多段线的线宽和颜色
```

```
    If width <> -1 Then
        objPline.ConstantWidth = width
    End If
    If colorIndex <> -1 Then
        Dim color As New AcadAcCmColor
        color.colorIndex = colorIndex
        objPline.TrueColor = color
    End If

    index = index + 1
    ptPrevious = ptCurrent

    GoTo NEXTPOINT                                    '提示用户继续拾取顶点
End Sub
```

前面已经分别对动态创建多段线和在 GetPoint 中使用关键字进行了详细介绍，这里的代码基本上就是将两个程序综合在一起，因此不再详细分析。

2.4.4　实例效果

在 VBA 集成开发环境中，将光标定位在 CreatePolyline 宏内部，按 F5 键运行该宏，按照命令提示进行操作：

```
命令：输入第一点：
命令：输入下一点 [宽度(W)/颜色(C)]<完成(O)>：
命令：输入下一点 [宽度(W)/颜色(C)]<完成(O)>：w  【输入 W 要求改变线宽】

命令：
命令：输入线宽：2                                  【输入 2 作为线宽值】

命令：输入下一点 [宽度(W)/颜色(C)]<完成(O)>：c  【输入 C 要求改变颜色】

命令：
命令：输入颜色索引值：1                            【创建红色的多段线】

命令：输入下一点 [宽度(W)/颜色(C)]<完成(O)>：
命令：输入下一点 [宽度(W)/颜色(C)]<完成(O)>：
命令：输入下一点 [宽度(W)/颜色(C)]<完成(O)>：o  【输入 O 完成创建】
```

完成操作后，得到一条线宽为 2 的红色多段线。

2.4.5　小结

1. 本节所涉及的知识点

本节内容涉及到较多的知识点，需要读者熟练掌握，因为在编程过程中这些知识可能会经常被用到。

- 判断用户是否按 Esc 键。

- 交互(动态)创建多段线。
- 在 GetPoint 方法中使用关键字(在 GetDistance、GetAngle 等方法中同样可以使用)。
- 利用 GoTo 语句实现 While 语句的功能。

2. 程序进一步改进的建议

本节的实例中还有一个小问题没有解决,也可以看作是一个 Bug,如果在输入所有的顶点之后调整线宽和颜色,然后完成多段线的创建,那么对多段线线宽和颜色的调整将不起作用!至于这个问题如何解决,就请开动脑筋了。

第 3 章　创建和编辑二维图形对象

创建和编辑二维图形对象包含了很多的内容，本章对选择集的使用进行了完全剖析，为 SendCommand 提供了更广泛的应用，分析了直线、圆、圆弧和样条曲线向多段线的转化，提供了一些常用的多段线操作，提出制图法解决问题的新思路，编写了根据起点、终点和弧长创建圆弧的函数，最后又给出绘制放样展开图的方法。

3.1　根据起点、终点和弧长创建圆弧

3.1.1　实例说明

AutoCAD 中提供了 11 种创建圆弧的方法，但是在 ActiveX 模型中仅提供了一种方法 AddArc，能够根据圆心、半径、起始角度和终止角度来创建对应的圆弧。实际应用中，可能需要根据起点、终点和弧长来创建圆弧，这种方式在 AutoCAD 中也没有提供。本节利用牛顿切线法来解决这个问题。

3.1.2　编程思路

假定圆弧起点为 A，终点为 B，弧长为 S，如图 3.1 所示。要画出圆弧，就要确定圆心 O、半径 R，圆弧初始角 $\angle AOE$ 和圆弧终止角 $\angle BOE$。

设圆弧的圆心角 $\angle AOB=x$，因为 A、B 两点已定，故可求出 AB 两点间距离 L，并且有

$$\begin{cases} S = R \cdot x \\ L = 2R \cdot \sin\dfrac{x}{2} \end{cases}$$

联立两个方程消去 R，得到 $2S \cdot \sin\dfrac{x}{2} - L \cdot x = 0$。

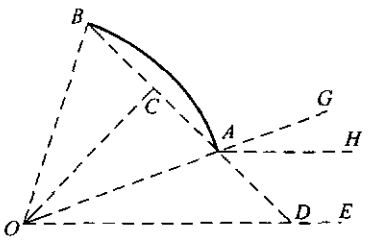


图 3.1　分析圆弧

在上式中，只有 x 是未知数，如果能求出 x，问题就解决了。遗憾的是，一般的方法无法解出这个方程。这时，牛顿切线法就可大显身手了。

对方程 $f(x)=0$，先给定一初值 x_0 作为方程的近似解，经过若干次迭代后，得到方程很高精度的近似解。牛顿切线法迭代公式为：

$$x_{i+1} = x_i - \frac{f(x)}{f'(x)}$$

其中，$f'(x_i)$ 是 $f(x_i)$ 的导数，当 $|x_{i+1} - x_i| \leqslant \epsilon$（$\epsilon$ 可取很小的值，如 0.000 000 01）时，x_{i+1} 就作为方程的近似解。

在这里，

$$f(x) = 2S \cdot \sin\frac{x}{2} - L \cdot x = 0$$

$$f'(x) = S \cdot \cos\frac{x}{2} - L$$

注意，在编程时要设一个初值。初值必须取在函数相应的单调区间，如图 3.2 所示，初值点 C 必须取在 A、B 两点之间。

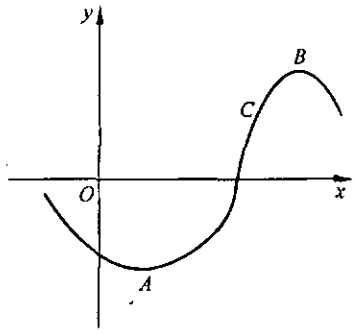


图 3.2 选择初值

求出圆心角 x 后，圆心 O、半径 R，圆弧初始角$\angle AOE$和圆弧终止角$\angle BOE$就很容易求出。

3.1.3 步骤

(1) 在 VBA 集成开发环境中，创建一个新工程，保存在适当的位置。在“工程资源管理器”窗口中双击 ThisDrawing 图标，打开该模块的代码窗口，向其中添加一个函数 Distance，用于计算两点之间的距离。

```
Private Function Distance(ByVal ptSt As Variant, ByVal ptEn As Variant) As Double
    Dim x As Double
    Dim y As Double
    Dim z As Double

    x = ptSt(0) - ptEn(0)
    y = ptSt(1) - ptEn(1)
    z = ptSt(2) - ptEn(2)

    Distance = Sqr((Sqr((x ^ 2) + (y ^ 2)) ^ 2) + (z ^ 2))
End Function
```

(2) 创建宏 CreateArc，根据用户输入的起点、终点和圆心创建圆弧，其实现代码如下所示。

```
Public Sub CreateArc( )
    Dim objArc As AcadArc
    Dim arcLength As Double, chordLength As Double, radius As Double   '弧长、弦长和半径
    Dim x As Double, xx As Double                                      '前一次迭代值和后一次迭代值
    Dim fx As Double, flx As Double                                    'f(x)和 f'(x)
    Dim obliquityStEn As Double                                        '起点到终点的矢量与 X 轴正半轴角度
    Dim obliquityStCen As Double                                       '起点到圆心的矢量与 X 轴正半轴角度
    Dim AngStart As Double, AngEnd As Double                           '圆弧的起始角度和终止角度
    Dim ptStart As Variant, ptEnd As Variant, ptCen As Variant         '圆弧的起点、终点和圆心
    Dim keyWord As String                                              '用户输入的关键字
    Dim pi As Double                                                   'π 的值

    pi = 4 * Atn(1)

    '提示用户输入起点、终点和弧长
    ptStart = ThisDrawing.Utility.GetPoint(, vbCrLf & "请输入圆弧起点：")
    ptEnd = ThisDrawing.Utility.GetPoint(ptStart, vbCrLf & "请输入圆弧终点：")
    arcLength = ThisDrawing.Utility.GetDistance(ptStart, vbCrLf & "请输入圆弧弧长：")
```

```
    ThisDrawing.Utility.InitializeUserInput 0, "S N"
    keyWord = ThisDrawing.Utility.GetKeyword(vbCrLf & "圆弧的方向 [逆时针(N)/顺时针/(S)]
        <N>: ")
    If keyWord = "" Then keyWord = "N"

    '如果是顺时针绘制,将起点和终点交换
    If StrComp(keyWord, "S") = 0 Then
        Dim ptTemp As Variant
        ptTemp = ptStart
        ptStart = ptEnd
        ptEnd = ptTemp
    End If

    chordLength = Distance(ptStart, ptEnd)
    obliquityStEn = ThisDrawing.Utility.AngleFromXAxis(ptStart, ptEnd)

    If arcLength <= chordLength Then
        MsgBox "您要画的圆弧并不存在,请再执行一次程序!"
        End
    End If

    '利用"牛顿切线法"计算圆弧的圆心角(取迭代结果 xx)
    x = 2 * pi      '给定初值
    xx = x
    Do
        x = xx
        fx = 2 * arcLength * Sin(x/2) - chordLength * x
        flx = arcLength * Cos(x/2) - chordLength
        xx = x - fx/flx
    Loop While Abs(xx - x) > 0.0000000001

    '计算圆弧的其他参数
    radius = arcLength/xx
    obliquityStCen = obliquityStEn - xx * 0.5 + pi/2
    ptCen = ThisDrawing.Utility.PolarPoint(ptStart, obliquityStCen, radius)
    AngStart = obliquityStCen + pi
    AngEnd = AngStart + xx

    Set objArc = ThisDrawing.ModelSpace.AddArc(ptCen, radius, AngStart, AngEnd)
End Sub
```

计算 π 的值使用了一个小技巧,由于角度为 π/4 的角的正切值是 1,那么 1 的反正切值就是 π/4,由此不难写出获得 π 精确值的语句:

```
pi = 4 * Atn(1)
```

使用牛顿切线法,程序中循环次数很少。可以修改代码,增加一个计数器变量,以观察

循环次数。对于有的程序可能存在无解的情况，可通过条件转移语句，当循环次数超过一定数值后，跳出循环，并向用户反馈信息。

另外，程序中又增加了圆弧方向的选择，要注意 InitializeUserInput 方法和 GetKeyword 方法的运用。

3.1.4 实例效果

在 VBA 集成开发环境中，将光标定位在 CreateArc 宏内部，然后按 F5 键，运行该宏，按照命令提示进行操作，即可创建符合要求的圆弧。

3.1.5 小结

通过本节的实例简单介绍了牛顿切线法的应用，这种方法是求解方程的有力工具，毕竟很多方程(高次方程、带有三角函数的方程等)用传统的方法是难以求解的。

3.2 利用制图法通过三点创建圆

3.2.1 实例说明

在没有 CAD 前，一般图样都是通过手工制图来完成的，在手工制图中，需要很丰富的制图技术才能提高制图的速度。自从有了计算机辅助设计，特别是有了 AutoCAD 后，许多计算都可以由计算机来完成，但是掌握丰富的制图技巧一样可提高制图的速度。

在制图过程中，辅助线是必不可少的，例如，为了绘制一个边长已知的等边三角形，必须先画一条线，然后以线的两个端点为圆心，边线的长度为半径，从两个顶点处分别绘制两个圆，这样，圆的交点就是三角形的第三个顶点了，通过这三个顶点形成的三角形就是等边三角形。该示例中，两个圆就是辅助线，它在完成求交点后会被删除。

过三点画圆，在 AutoCAD 界面下本身就提供了这样的功能，但本节要介绍的是通过程序并用制图法来画。

3.2.2 编程思路

过三点画圆，最关键是取得圆心，而圆心到三点的距离是相同的。首先绘制两点形成的线段的中垂线，这样，中垂线上任意一点到这两个端点的距离是相同的。同样，再绘制另外两点形成的线段的中垂线，两条中垂线相交的交点到三个点的距离是相同的，也就是说，两条中垂线的交点就是所要求的圆心。

圆的半径就是中垂线交点到三点中任意一点的距离。

圆心和半径求出后，自然可以绘制出圆。

3.2.3 步骤

(1) 在 AutoCAD 2005 中，打开 VBA 管理器，创建一个新工程，保存在适当的位置。进入 VBA 集成开发环境，双击“工程资源管理器”窗口中的 ThisDrawing 图标，打开代码窗

口，创建一个函数 BisectorPLine，通过两个点绘制中垂线，其实现代码为：

```
Function BisectorPLine(Point1 As Variant, Point2 As Variant) As AcadPolyline
    Dim Dist As Double
    Dim Circle1 As AcadCircle
    Dim Circle2 As AcadCircle
    Dim Pnts As Variant

    '求两点间的距离
    Dist = Distance(Point1, Point2)

    '按点为圆心，两点距离为半径画圆
    Set Circle1 = ThisDrawing.ModelSpace.AddCircle(Point1, Dist)
    Set Circle2 = ThisDrawing.ModelSpace.AddCircle(Point2, Dist)

    '获取两个圆的交点，交点一定有两个
    Pnts = Circle1.IntersectWith(Circle2, acExtendNone)

    '删除这两个辅助的圆
    Circle1.Delete: Circle2.Delete

    '用交点绘制多段线
    Set BisectorPLine = ThisDrawing.ModelSpace.AddPolyline(Pnts)
End Function
```

其中，Distance 是一个自定义函数，用于计算两点之间的距离，其实现代码为：

```
Function Distance(Point1 As Variant, Point2 As Variant) As Double
    Dim Line As AcadLine

    '通过两个点绘制直线
    Set Line = ThisDrawing.ModelSpace.AddLine(Point1, Point2)

    '获取线的长度，即为两点间的距离
    Distance = Line.Length
    Line.Delete
End Function
```

(2) 添加一个函数 ThreePointCircle，用于根据已知的三点创建一个圆，其实现代码为：

```
Function ThreePointCircle(Pnt1 As Variant, Pnt2 As Variant, Pnt3 As Variant) As AcadCircle
    Dim Dist As Double
    Dim Pnts As Variant
    Dim Line1 As AcadPolyline
    Dim Line2 As AcadPolyline

    '分两次求两点的中垂线
    Set Line1 = BisectorPLine(Pnt1, Pnt2)
```

```
    Set Line2 = BisectorPLine(Pnt1, Pnt3)

    '求两条中垂线的交点,该交点即为所要求的圆的圆心
    Pnts = Line1.IntersectWith(Line2, acExtendBoth)
    '删除辅助用的两条中垂线
    Line1.Delete: Line2.Delete

    '求圆心到其中一点的距离,即为圆的半径
    Dist = Distance(Pnts, Pnt1)

    '画圆
    Set ThreePointCircle = ThisDrawing.ModelSpace.AddCircle(Pnts, Dist)
End Function
```

(3) 添加一个宏 DrawCircle,用于提示用户拾取三个点,并根据这三点在图形窗口中添加一个通过三点的圆,其实现代码为:

```
Public Sub DrawCircle( )
    Dim P1 As Variant
    Dim P2 As Variant
    Dim P3 As Variant

    '提示用户在图形中选择三个点
    P1 = ThisDrawing.Utility.GetPoint(, vbCr & "第一点: ")
    P2 = ThisDrawing.Utility.GetPoint(, vbCr & "第二点: ")
    P3 = ThisDrawing.Utility.GetPoint(, vbCr & "第三点: ")
    Dim C As AcadCircle

    '用三点画圆
    Set C = ThreePointCircle(P1, P2, P3)
End Sub
```

3.2.4 实例效果

在 VBA 集成开发环境中,执行 DrawCircle 宏,在图形窗口中以此拾取三个点,系统会自动创建一个通过这三点的圆。

3.2.5 小结

本节提供给了一个思路,在 VBA 编程时,可以通过数学的方法求解,也可以通过 AutoCAD 图形的方法求解。虽然说数学的方法会快些,但制图法却可以让整个程序的思路更加清晰。编程者可以利用机械制图的知识进行编程。

本节充分地利用了 IntersectWith 方法,也正是该方法使通过图形制图法进行求解的思路得以实现。IntersectWith 方法是求两个图元对象的交点,它对任何图元对象都有效,参数中可以选择图元是否通过延长线求交点。

3.3　获得点到多段线的最短距离

3.3.1　实例说明

在某些情况下，需要计算任意一点到多段线的最短距离，但是 ActiveX 模型中没有提供相应的方法，只能使用数学的方法进行计算和分析。

本节的实例能够获得 WCS 的 XOY 平面上任意点到轻量多段线的最短距离，由于多段线包含直线段和圆弧段，因此，本节的实例同样包含了计算点到直线和圆弧最短距离的方法。

3.3.2　编程思路

计算点到轻量多段线的最短距离，其基本思想是分别计算点到轻量多段线各段的最短距离，然后从中找出一个最小值，将其作为点到整个多段线的最短距离。

1. 计算点到直线的距离

点和直线的位置关系可以分为两种情况：点到直线的垂足在直线上，点到直线的垂足在直线的延长线上。这两种情况如图 3.3 所示。

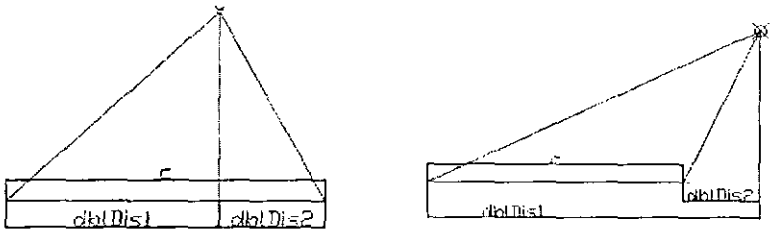


图 3.3　点和直线位置关系的两种情况

其中，c 是直线的长度，dblDis1 是直线的一个端点到垂足的距离，dblDis2 是直线的另一端点到垂足的距离。如果 dblDis1 和 dblDis2 的任何一个大于 c，就表示点到直线的垂足在直线的延长线上。

点到直线的垂足在直线上时，点到直线的最短距离就是点到直线的垂线段长度；点到直线的垂足在直线的延长线上时，点到直线的最短距离是点到直线两端点距离中较小的值。

2. 计算点到圆弧的距离

点和圆弧的位置关系同样可以分为两种情况：点在圆弧的扇形区域内(pt1 点)，点在圆弧的扇形区域外(pt2 点)。这两种情况如图 3.4 所示。

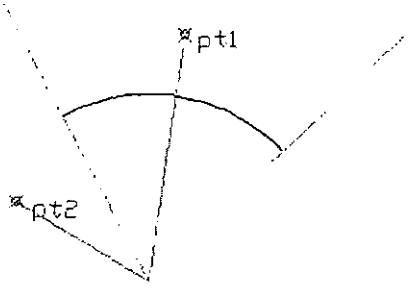


图 3.4　点和圆弧的两种位置关系

假设圆弧的起始角度为 angleStart，终止角度为

angleEnd，圆弧圆心到点的矢量与 X 轴正半轴所成的角度为 angle，那么如果(angle - angleStart) * (angle - angleEnd) * (angleEnd - angleStart) > 0，则点不在圆弧的扇形区域内；否则，点在圆弧的扇形区域内。

点在圆弧的扇形区域内时，点到圆弧的最近距离是点到圆心的距离和半径之差的绝对值；点在圆弧的扇形区域外时，点到圆弧的最近距离是点到圆弧两端点距离中较小的值。

3.3.3 步骤

(1) 在 AutoCAD 2005 中，打开 VBA 管理器，创建一个新工程，保存在适当的位置。进入 VBA 集成开发环境，双击“工程资源管理器”窗口中的 ThisDrawing 图标，打开代码窗口，在其中添加一个常量的声明：

```
Const ZERO As Double = 0.00000001
```

这个常量在程序中表示 0。

(2) 添加一个函数 DisPt2Line，用于计算点到直线的最短距离，其实现代码为：

```
Private Function DisPt2Line(ByVal pt As Variant, ByVal ptStart As Variant,
                            ByVal ptEnd As Variant) As Double
    '计算三点所组成的三角形的面积 s = sqr(p * (p - a) * (p - b) * (p - c)),p = 0.5 * (a + b + c)
    Dim area As Double
    Dim p As Double                                  '周长的一半
    Dim a As Double, b As Double, c As Double        '各条边的边长
    a = Distance(pt, ptStart)
    b = Distance(pt, ptEnd)
    c = Distance(ptStart, ptEnd)
    p = (a + b + c)/2
    area = Sqr(p * (p - a) * (p - b) * (p - c))

    '计算点到直线的垂直距离
    Dim dblDistance As Double
    dblDistance = 2 * area/c

    '计算垂足到直线两端点的距离
    Dim dblDis1 As Double, dblDis2 As Double
    dblDis1 = Sqr(a ^ 2 - dblDistance ^ 2)
    dblDis2 = Sqr(b ^ 2 - dblDistance ^ 2)

    '根据点是否在直线两端点之间，返回点到直线的最短距离
    If dblDis1 > c Or dblDis2 > c Then
        If a > b Then
            DisPt2Line = b
        Else
            DisPt2Line = a
        End If
    Else
```

```
        DisPt2Line = dblDistance
    End If
End Function
```

在计算点到直线的垂线段长度时，首先使用海伦公式根据三角形的三边长计算三角形面积，然后将直线两个端点之间的线段作为底边长，计算出点到直线的垂线段长度。

其中，Distance 是一个自定义函数，用于计算 XOY 平面上两点之间的距离：

```
Private Function Distance(ByVal pt1 As Variant, ByVal pt2 As Variant) As Double
    Distance = Sqr((pt1(0) - pt2(0)) ^ 2 + (pt1(1) - pt2(1)) ^ 2)
End Function
```

前面已经介绍过，本节的实例仅适用于在 WCS 的 XOY 平面上计算点到轻量多段线的最短距离，由于轻量多段线返回的坐标值是二维的，因此这里就忽略了对第三维(Z 轴)坐标的计算。

(3) 添加一个函数 DisPt2Arc，用于计算点到圆弧的距离，其实现代码为：

```
Private Function DisPt2Arc(ByVal pt As Variant, ByVal objArc As AcadArc) As Double
    '假设点在圆弧的扇形区域内，返回点到圆弧的距离
    DisPt2Arc = Abs(Distance(pt, objArc.Center) - objArc.radius)

    '计算点到圆弧两个端点的距离
    Dim dblDis1 As Double, dblDis2 As Double
    dblDis1 = Distance(pt, objArc.StartPoint)
    dblDis2 = Distance(pt, objArc.EndPoint)

    '如果点不在圆弧的扇形区域内，则到两端点的距离包含了一个最小距离
    Dim angle As Double                          '圆心到点的矢量的角度
    angle = ThisDrawing.Utility.AngleFromXAxis(objArc.Center, pt)
    Dim angleStart As Double, angleEnd As Double
    angleStart = objArc.startAngle
    angleEnd = objArc.endAngle
    If (angle - angleStart) * (angle - angleEnd) * (angleEnd - angleStart) > ZERO Then
        If dblDis1 > dblDis2 Then
            DisPt2Arc = dblDis2
        Else
            DisPt2Arc = dblDis1
        End If
    End If
End Function
```

(4) 添加一个函数 DistancePt2Poly，用于计算点到多段线的最近距离，其实现代码为：

```
Private Function DistancePt2Poly(ByVal pt As Variant, ByVal objPoly As AcadLWPolyline)
    As Double
    Dim intVertCount As Integer                  '多段线的顶点数量
    Dim varCoords As Variant                     '保存所有顶点的坐标数组
    varCoords = objPoly.Coordinates
```

```
intVertCount = (UBound(varCoords) + 1)/2

'遍历所有顶点,判断每一段曲线与点之间的距离
Dim i As Integer
Dim ptCurrent As Variant, ptNext As Variant      '当前顶点和下一个顶点
Dim minDistance As Double                        '最短距离
For i = 0 To intVertCount - 1
    '获得当前顶点和下一个顶点的坐标
    If i < intVertCount - 1 Then
        ptCurrent = objPoly.Coordinate(i)
        ptNext = objPoly.Coordinate(i + 1)
    ElseIf objPoly.Closed Then
        ptCurrent = objPoly.Coordinate(i)
        ptNext = objPoly.Coordinate(0)
    Else
        Exit For
    End If

    '分情况计算距离
    If objPoly.GetBulge(i) = 0 Then      '如果该段是直线
        If i = 0 Then
        minDistance = DisPt2Line(pt, ptCurrent, ptNext)
    Else
        If DisPt2Line(pt, ptCurrent, ptNext) < minDistance Then
            minDistance = DisPt2Line(pt, ptCurrent, ptNext)
        End If
    End If
Else
    Dim aimPoly As AcadLWPolyline                '辅助多段线
    Dim ptVerts(0 To 3) As Double                '辅助多段线的顶点
    Dim varEnts As Variant                       '分解辅助多段线后得到的实体
    Dim aimArc As AcadArc                        '辅助圆弧
    ptVerts(0) = ptCurrent(0)
    ptVerts(1) = ptCurrent(1)
    ptVerts(2) = ptNext(0)
    ptVerts(3) = ptNext(1)
    Set aimPoly = ThisDrawing.ModelSpace.AddLightWeightPolyline(ptVerts)
    aimPoly.SetBulge 0, objPoly.GetBulge(i)

    '分解辅助多段线,得到一个圆弧
    varEnts = aimPoly.Explode
    If TypeOf varEnts(0) Is AcadArc Then
        Set aimArc = varEnts(0)
    End If
```

```
        '计算点到圆弧段的距离
        If i = 0 Then
            minDistance = DisPt2Arc(pt, aimArc)
        Else
            If DisPt2Arc(pt, aimArc) < minDistance Then
                minDistance = DisPt2Arc(pt, aimArc)
            End If
        End If

        '删除辅助圆弧和多段线
        aimPoly.Delete
        aimArc.Delete
        End If
    Next i

    DistancePt2Poly = minDistance
End Function
```

上面的代码分别计算点到多段线每一段的最短距离，然后取其中的最小值作为点到多段线的最短距离。

在求点到曲线段的最短距离时，首先根据多段线该段曲线的参数（起、终点坐标和凸度）创建一条与该段曲线完全重合的多段线，然后对其进行分解，得到一个与多段线该段曲线重合的圆弧，以此圆弧作为参数调用 DisPt2Arc 函数，获得点到该段曲线的最近距离。

（5）添加一个宏 GetDistancePt2Poly，用于测试本节创建的 DistancePt2Poly 函数，其实现代码为：

```
Public Sub GetDistancePt2Poly( )
    Dim pt As Variant
    pt = ThisDrawing.Utility.GetPoint(, "拾取一点：")

    Dim objPoly As AcadLWPolyline
    Dim ptPick As Variant
    ThisDrawing.Utility.GetEntity objPoly, ptPick, "选择多段线："

    MsgBox "点到多段线的最近距离是：" & DistancePt2Poly(pt, objPoly)
End Sub
```

3.3.4　实例效果

在 AutoCAD 2005 中，创建一条多段线。在 VBA 集成开发环境中，执行 DistancePt2Poly 宏，在图形中选择一点，然后拾取创建的多段线，就能得到点到多段线的最短距离，如图 3.5 所示。

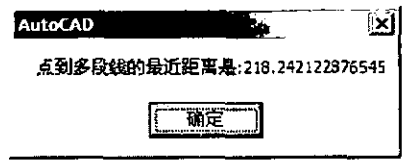


图 3.5　获得点到多段线的最短距离

3.3.5 小结

学习本节内容之后，为了巩固学习成果，可以对本节的程序进行改进，在获得点到轻量多段线最近距离的同时，能够获得多段线上距已知点最近的点的坐标。

3.4 选 择 集

3.4.1 实例说明

在 VBA 开发过程中，经常需要用户和 AutoCAD 之间进行交互操作，除了前面介绍的 Utility 对象的 Get 系列方法之外，选择集也是 AutoCAD 和用户操作的重要手段。与 GetEntity 方法不同，选择集允许用户同时选择多个图形对象，并提供了丰富的手段来选择符合特定条件的实体。

本节实例详细介绍了选择集的使用，可以分为以下几个方面。

- 选择集的创建和删除。
- 选择集中对象的增加和删除。
- 对象选择的方法。
- 选择集过滤器的使用。

3.4.2 编程思路

1. 选择集的创建和删除

要创建一个选择集，可以使用 SelectionSets 集合的 Add 方法。在创建选择集时，如果已存在同名的选择集，AutoCAD 将返回一条错误信息。因此，应及时删除不再需要的选择集。

要删除一个选择集，可以使用 SelectionSet 对象的 Delete 方法。

提示：有一个十分常见的问题，那就是“为什么我的程序第一次运行时正确，第二次就出问题了？后来一直不能运行。”这个问题绝大部分的原因都是未能正确使用选择集。不要以为在宏的开始处创建选择集，结束处用 Delete 方法删除对应的选择集就可以万事无忧。一次写出完全正确的程序基本是不可能的，如果在中途出现错误而退出程序，那么图形中就留下了一个选择集，下一次运行程序的时候必然导致图形中存在同名的选择集，返回错误信息。

安全创建选择集的方法，参见 3.4.3 节的内容。

2. 选择集中对象的增加和删除

这里所说的对象的增加和删除，仅指在已经获得对象引用的情况下，使用 AddItems 和 RemoveItems 方法进行对象的增加和删除。

AddItems 方法定义为：

```
object.AddItems Items
```

RemoveItems 方法定义为：

```
object.RemoveItems Objects
```

这两个方法中，Items 和 Objects 的定义均为所要操作的实体的数组。

此外，Clear 和 Erase 方法均能删除选择集中的所有元素。Clear 方法仅仅将元素从选择集中删除，被删除的元素仍然存在于图形中；Erase 方法在将元素从选择集中删除的同时，也将其从图形中删除。

3. 对象选择的方法

所谓对象选择的方法，就是以某种方式从图形窗口中获得满足某些条件的图形对象。

Select 方法选择对象并将其放到活动的选择集中。用户可以选择所有对象、位于矩形区域内或与其相交的对象、位于多边形区域内或与其相交的对象、与选择栏相交的所有对象、最近创建的对象、上一个选择集中的对象、窗口内的对象，以及多边形窗口内的对象。

SelectAtPoint 方法选择穿过给定点的一个对象并将其放到活动的选择集中。由于该方法仅能选择一个过给定点的对象，那么其意义实在不大，后面将用其他函数来代替它。

SelectByPolygon 方法选择位于选择栏内的对象，并将其添加到活动的选择集中。

SelectOnScreen 方法提示用户在屏幕上拾取的对象，并将其添加到活动的选择集中。

4. 使用选择集过滤器

在使用各种选择对象的方法时，可以使用过滤器来限定选择的对象。例如，可以指定仅选择图层 0 上的直线对象，也可以指定仅选择蓝色的半径大于 30 的圆等。

下面是使用选择集过滤器的一个典型实例：

```
Dim fType(0 To 1) As Integer
Dim fData(0 To 1) As Variant
fType(0) = 0: fData(0) = "Line"
fType(1) = 8: fData(1) = "0"
SSet.Select acSelectionSetAll, , , fType, fData
```

过滤器列表由成对的参数组成。第一个参数标识过滤器的类型（例如对象），第二个参数指定要过滤的值（例如圆）。过滤器类型是指定使用哪种过滤器的 DXF 组码，下面列出了一些最常用的过滤器类型。

- 0：对象类型（字符串），例如 Line、Arc 等。
- 1：图元的主文字值（字符串），例如文字、属性的字符串内容。
- 2：对象名（字符串），例如属性标记、块名等。
- 8：图层名（字符串）。
- 10：主要点（直线或文字图元的起点、圆的圆心等），用三个实数的数组来表示三维点。
- 60：对象可见性（整数），0 表示可见，1 表示不可见。

- 62：颜色编号（整数），可取 0～256 的整数值，其中 0 表示 ByBlock，256 表示 ByLayer。
- 67：模型/图纸空间标识符（整数），0（默认）表示在模型空间，1 表示在图纸空间。
- 1000：扩展数据中的 ASCII 字符串，最多可以包含 255 个字节。
- 1001：扩展数据的注册应用程序名，最多可以包含 31 个字节的 ASCII 字符串。
- 1003：扩展数据图层名。

关于 DXF 组码的完整列表，可以参考“DXF 参考”中的内容，如图 3.6 所示。

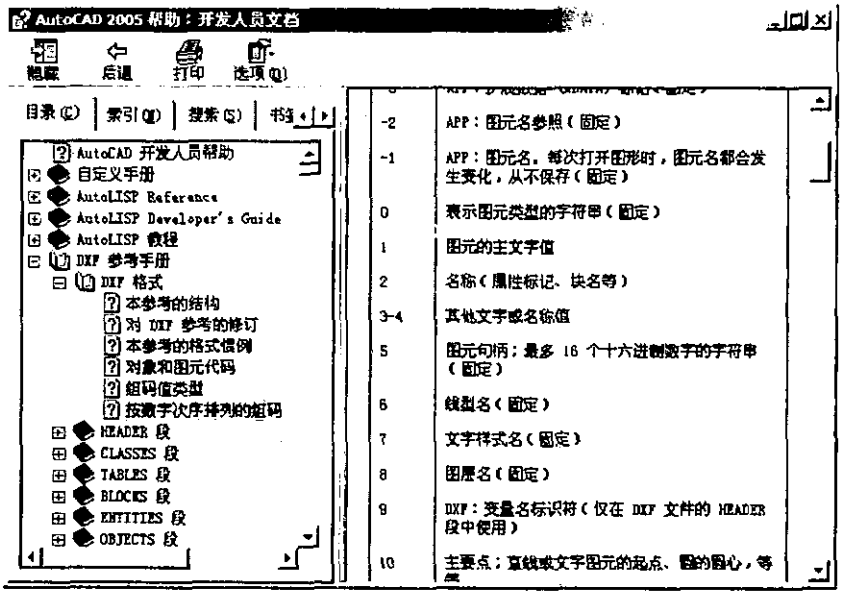


图 3.6 完整的 DXF 组码参考

不得不指出，创建选择集过滤器的语句是繁琐的，特别是在同时指定多个过滤条件的情况下。在后面的步骤中，将使用其他函数来简化创建选择集过滤器的步骤，同时会介绍多种复杂条件的选择集。

需要注意的一点是，在 Select、SelectAtPoint、SelectByPolygon 和 SelectOnScreen 方法中均可以使用选择集过滤器。

3.4.3 步骤

1. 选择集的创建和删除

在 AutoCAD 2005 中，打开 VBA 管理器，创建一个新工程，保存在适当的位置上。进入 VBA 集成开发环境，在“工程资源管理器”窗口中双击 ThisDrawing 图标，进入代码窗口，在其中添加安全创建选择集的宏：

```
Public Sub CreateSelectionSet( )
    '安全创建选择集
    On Error Resume Next
```

```
    Dim SSet As AcadSelectionSet
    If Not IsNull(ThisDrawing.SelectionSets.Item("Example")) Then
        Set SSet = ThisDrawing.SelectionSets.Item("Example")
        SSet.Delete
    End If
    Set SSet = ThisDrawing.SelectionSets.Add("Example")

    '删除选择集
    SSet.Delete
End Sub
```

首先用 IsNull 函数判断指定名称的选择集是否存在，如果存在就先将其删除，然后再创建选择集，这就有效地避免了图形中存在同名的选择集。

2. 选择集中对象的增加和删除

(1) 在 VBA 集成开发环境中，选择“插入”|“模块”菜单项，向工程中添加一个标准模块，在“属性”窗口中将其名称修改为 mdlBasic。在标准模块中，添加一个宏 AddEntToSSet，用于将一个实体添加到选择集中，其实现代码为：

```
Public Sub AddEntToSSet(ByVal ent As AcadEntity, ByRef SSet As AcadSelectionSet)
    Dim objCollection(0) As AcadEntity
    Set objCollection(0) = ent

    SSet.AddItems objCollection
End Sub
```

(2) 添加一个宏 AddEntsToSSet，用于将多个实体添加到选择集中，其实现代码为：

```
Public Sub AddEntsToSSet(ByVal ents As Variant, ByRef SSet As AcadSelectionSet)
    If VarType(ents) <> vbArray + vbObject Then
        MsgBox "ents 参数无效!"
        Exit Sub
    End If

    Dim objCollection( ) As AcadEntity
    ReDim objCollection(UBound(ents))

    Dim i As Integer
    For i = 0 To UBound(ents)
        Set objCollection(i) = ents(i)
    Next i

    SSet.AddItems objCollection
End Sub
```

该函数的输入参数是一个 Variant 类型的变量，但实际上保存了一个包含多个实体的数组，因此，VarType(ents)的返回值应该是 vbArray＋vbObject，否则就表示参数无效。

ObjCollection 数组的元素个数应该与实体数组一致，但是无法在编写程序时知道实体数组的维数，只能通过动态数组来实现。

(3) RemoveEntFromSSet 宏用于从选择集中移除指定的实体，其实现代码为：

```
Public Sub RemoveEntFromSSet(ByVal ent As AcadEntity, ByRef SSet As AcadSelectionSet)
    Dim objCollection(0) As AcadEntity
    Set objCollection(0) = ent

    SSet.RemoveItens objCollection
End Sub
```

(4) RemoveEntsFromSSet 宏用于从选择集中移除多个实体，其实现代码为：

```
Public Sub RemoveEntsFromSSet(ByVal ents As Variant, ByRef SSet As AcadSelectionSet)
    If VarType(ents) <> vbArray + vbObject Then
        MsgBox "ents 参数无效!"
        Exit Sub
    End If

    Dim objCollection( ) As AcadEntity
    ReDim objCollection(UBound(ents))

    Dim i As Integer
    For i = 0 To UBound(ents)
      Set objCollection(i) = ents(i)
    Next i

    SSet.RemoveItems objCollection
End Sub
```

(5) ExportSSet 宏用于将选择集导出为对象数组，以便用于其他的函数，其实现代码为：

```
    Public Function ExportSSet(ByVal SSet As AcadSelectionSet) As Variant
    Dim objCollection( ) As AcadEntity
    ReDim objCollection(SSet.count - 1)

    Dim i As Integer
    For i = 0 To SSet.count - 1
        Set objCollection(i) = SSet.Item(i)
    Next i

    ExportSSet = objCollection
End Function
```

3. 对象选择的方法

(1) 在 VBA 集成开发环境中，向工程中添加一个新的模块，修改其名称为 mdlSelectEnts。在其中添加一个函数 SelectAtPoint，用于选择通过某点的所有实体，其实现代码为：

```
Public Sub SelectAtPoint(ByRef SSet As AcadSelectionSet, ByVal pt As Variant)
    '构造一个以 pt 为中心的小矩形作为选择范围
    Dim pt1 As Variant, pt2 As Variant
    Dim objUtility As Object
    Set objUtility = ThisDrawing.Utility              '必须使用后期绑定
    objUtility.CreateTypedArray pt1, vbDouble, pt(0) - 0.0001, pt(1) - 0.0001, pt(2)
    objUtility.CreateTypedArray pt2, vbDouble, pt(0) + 0.0001, pt(1) + 0.0001, pt(2)

    SSet.Select acSelectionSetCrossing, pt1, pt2
End Sub
```

上面的函数中，使用 Utility 对象的 CreateTypedArray 来简化创建点坐标，其作用与下面的代码完全一致。

```
Dim pt1(0 To 2) As Double
pt1(0) = pt(0) - 0.0001: pt1(1) = pt(1) - 0.0001: pt1(2) = pt(2)
```

需要注意的一点是，使用 CreateTypedArray 方法时，必须对 Utility 进行后期绑定，否则就会导致一个错误。

Select 方法提供了五种选择模式，其含义分别为：

- acSelectionSetWindow：窗口选择模式，以 pt1 和 pt2 作为窗口的角点，会选中全部位于窗口内部的实体。
- acSelectionSetCrossing：交叉选择模式，以 pt1 和 pt2 作为窗口的角点，会选中全部位于窗口内部的实体，以及与窗口矩形边界相交的实体。
- acSelectionSetPrevious：选中上一个选择集中包含的实体。
- acSelectionSetLast：选中图形窗口中创建的最后一个可见实体。
- acSelectionSetAll：选中图形窗口中所有的实体。

前面已经介绍，系统自身提供的 SelectAtPoint 方法仅能返回通过某点的一个实体，那么就可以用上面的函数来代替它。

(2) 添加一个宏 SelectByPoly，将指定的闭合多段线作为边界进行选择，其实现代码为：

```
Public Sub SelectByPoly(ByRef SSet As AcadSelectionSet, ByVal objPline As AcadLWPolyline,
    ByVal mode As AcSelect)
    If objPline.Closed = False Then
        MsgBox "作为边界的多段线不闭合!"
        Exit Sub
    End If

    '将轻量多段线的坐标输入到点数组中
    Dim pointArrs( ) As Double
    ReDim pointArrs((UBound(objPline.Coordinates) + 1) * 3/2 - 1)

    Dim i As Integer
    For i = 0 To ((UBound(objPline.Coordinates) + 1)/2 - 1)
        pointArrs(3 * i) = objPline.Coordinates(2 * i)
        pointArrs(3 * i + 1) = objPline.Coordinates(2 * i + 1)
```

```
        pointArrs(3 * i + 2) = 0
    Next i

    SSet.SelectByPolygon mode, pointArrs
End Sub
```

由于 SelectByPolygon 方法接收的是一个三维坐标的数组，而作为参数输入的 objPline 是一条轻量多段线，每个顶点仅保存两个坐标值，因此需要使用 AcadLWPolyline 对象的 Coordinates 属性，将其按照一定的规则转入到坐标数组中。

4. 使用选择集过滤器

(1) 在 VBA 集成开发环境中，向工程中添加一个新的模块，重命名为 mdlFilter，向其中添加一个函数 CreateSSetFilter，用于简化选择集过滤器的构建，其实现代码为：

```
Public Sub CreateSSetFilter(ByRef filterType As Variant, ByRef filterData As Variant,
    ParamArray filter( ))
    If UBound(filter) Mod 2 = 0 Then
        MsgBox "filter 参数无效!"
        Exit Sub
    End If

    Dim fType( ) As Integer '过滤器规则
    Dim fData( ) As Variant '过滤器参数
    Dim count As Integer
    count = (UBound(filter) + 1)/2
    ReDim fType(count - 1)
    ReDim fData(count - 1)

    Dim i As Integer
    For i = 0 To count - 1
        fType(i) = filter(2 * i)
        fData(i) = filter(2 * i + 1)
    Next i

    filterType = fType
    filterData = fData
End Sub
```

(2) 创建一个宏 FilterSSet，对 CreateSSetFilter 函数进行测试，创建一个基本的选择集过滤器，其实现代码为：

```
Public Sub FilterSSet( )
    '安全创建选择集
    On Error Resume Next
    Dim SSet As AcadSelectionSet
    If Not IsNull(ThisDrawing.SelectionSets.Item("Example")) Then
        Set SSet = ThisDrawing.SelectionSets.Item("Example")
```

```
    SSet.Delete
End If
Set SSet = ThisDrawing.SelectionSets.Add("Example")

'用 CreateSSetFilter 函数改进的过滤器
Dim fType As Variant, fData As Variant '选择集过滤器
Call CreateSSetFilter(fType, fData, 0, "Line", 8, "0")
SSet.Select acSelectionSetAll, , , fType, fData
MsgBox SSet.count

'删除选择集
SSet.Delete
End Sub
```

使用 CreateSSetFilter 函数创建的过滤器，将会限定选择条件是图层 0 上的直线，其效果与下面的标准过滤器相同：

```
Dim fType(0 To 1) As Integer
Dim fData(0 To 1) As Variant
fType(0) = 0: fData(0) = "Line"
fType(1) = 8: fData(1) = "0"
SSet.Select acSelectionSetAll, , , fType, fData
```

(3) 在过滤器中使用通配符。在 FilterSSet 宏中，创建一个带有通配符的过滤器，其相关代码为：

```
Dim fType As Variant, fData As Variant                    '选择集过滤器
Call CreateSSetFilter(fType, fData, 0, "Text", 8, "0,图层 1", 1, "*mjtd*")
SSet.Select acSelectionSetAll, , , fType, fData
```

上面的代码构建的过滤器，能够获得位于图层 0 或者图层 1 中、字符串内容包含 mjtd 的所有文字对象。其中，"0,图层 1"中的逗号(,)是一个通配符，用于分隔两个模式，表示两种情况均可；*mjtd* 中的星号(*)是一个通配符，用于匹配任意的字符序列，表示任何包含 mjtd 的字符串均可。

关于 AutoCAD 能够识别的通配符及其含义，如图 3.7 所示。

# （磅值符号）	匹配任意一个数字
@ （at）	匹配任意一个字母
. （句号）	匹配任意一个非字母数字的字符
* （星号）	匹配任意的字符序列（包括空字符串），它可以用在任何搜索模式中：包括开头、中间和结尾处
? （问号）	匹配任意一个字符
~ （波浪号）	如果它是模式中的的第一个字符，则匹配除此模式以外的任意内容
[...]	匹配方括号中的任意一个字符
[~...]	匹配不在方括号中的任意一个字符
- （连字符）	用在方括号中，指定一个字符的取值范围
, （逗号）	分隔两个模式
` （单引号）	避开特殊的字符（直接读取下一个字符）

图 3.7　帮助系统中对于通配符的说明

(4) 在过滤器中使用逻辑运算符。在宏 FilterSSet 中,创建包含逻辑运算符的过滤器,其相关代码为:

```
Dim fType As Variant, fData As Variant                    '选择集过滤器
Call CreateSSetFilter(fType, fData, -4, "<or", 0, "Text", 0, "MText", -4, "or>")
SSet.Select acSelectionSetAll, , , fType, fData
```

选择集过滤器所用的逻辑或运算符

开始运算符	包含的内容	结束运算符
"<AND"	一个或多个运算对象	"AND>"
"<OR"	一个或多个运算对象	"OR>"
"<XOR"	两个运算对象	"XOR>"
"<NOT"	一个运算对象	"NOT>"

图 3.8 帮助系统中对于逻辑运算符的描述

上面构建的过滤器,能够选择图形中所有的文字或多行文字。过滤器列表中的逻辑运算符用 DXF 组码-4 来指示。逻辑运算符是字符串,但必须成对出现,运算符以小于号开始(<),以大于号结束(>)。图 3.8 列出了可以在选择集过滤中使用的逻辑运算符。

(5) 在过滤器中使用关系运算符。在宏 FilterSSet 中,创建包含关系运算符的过滤器,其相关代码为:

```
Dim fType As Variant, fData As Variant                    '选择集过滤器
Call CreateSSetFilter(fType, fData, 0, "Circle", -4, ">=", 40, "30")
SSet.Select acSelectionSetAll, , , fType, fData
MsgBox SSet.count
```

上面构建的过滤器,能够选择图形中半径大于或等于 30 的所有圆,其中的组码 40 用于指定圆的半径。过滤器列表中的关系运算符以 DXF 组码-4 来指示,用字符串来表示关系运算符。图 3.9 列出了 AutoCAD 中可以使用的关系运算符。

(6) 结合使用通配符和关系运算符,创建更为复杂的过滤器。在宏 FilterSSet 中,创建包含关系运算符和通配符的过滤器,其相关代码为:

```
Dim pt1(0 To 2) As Double, pt2(0 To 2) As Double
pt1(0) = 0: pt1(1) = 0: pt1(2) = 0
pt2(0) = 100: pt2(1) = 100: pt2(2) = 0
Dim fType As Variant, fData As Variant
                                          '选择集过滤器
Call CreateSSetFilter(fType, fData, 0, "Circle",
     -4, ">,>,*", 10, pt1, -4, "<,<,*",
     10, pt2)
SSet.Select acSelectionSetAll, , , fType, fData
MsgBox SSet.count
```

上面构建的过滤器,能够选择图形中圆心在 pt1 和 pt2 两点构成的矩形内的圆,其中的组码 10 用于指定圆的圆心。

(7) 使用过滤器过滤扩展数据。在宏 FilterSSet 中,创建过滤扩展数据的过滤器,其相关代码为:

运算符	说明
"*"	任何情况(总为真)
"="	等于
"!="	不等于
"/="	不等于
"<>"	不等于
"<"	小于
"<="	小于或等于
">"	大于
">="	大于或等于
"&"	按位与(AND,仅限于整数组)
"&="	按位屏蔽相等(仅限于整数组)

图 3.9 帮助系统中对于关系运算符的描述

```
Dim fType As Variant, fData As Variant                    '选择集过滤器
Call CreateSSetFilter(fType, fData, 1001, "XData")
SSet.Select acSelectionSetAll, , , fType, fData
MsgBox SSet.count
```

上面构建的过滤器，能够选择图形中所有包含 Xdata 应用程序扩展数据的图元。但是，除轻量多段线之外，其他的实体都不支持扩展数据内容的过滤，也就是说，如果使用下面的代码：

```
Dim fType As Variant, fData As Variant        '选择集过滤器
Call CreateSSetFilter(fType, fData, 1000, "Road")
SSet.Select acSelectionSetAll, , , fType, fData
MsgBox SSet.count
```

上面的过滤器仅能选择到图形中所有包含字符串 Road 扩展数据的轻量多段线，其他类型的实体不起作用。

关于在创建选择集过滤器时，如何获得所需要的组码将在本节的小结中进行讲解。

3.4.4　实例效果

(1) 在 mdlBasic 模块中，添加一个 CreateSelectionSet 宏，对编写的相关函数进行测试，其实现代码为：

```
Public Sub CreateSelectionSet( )
    '安全创建选择集
    On Error Resume Next
    Dim SSet As AcadSelectionSet
    If Not IsNull(ThisDrawing.SelectionSets.Item("Example")) Then
        Set SSet = ThisDrawing.SelectionSets.Item("Example")
        SSet.Delete
    End If
    Set SSet = ThisDrawing.SelectionSets.Add("Example")

    '创建测试所用的实体
    Dim pt(0 To 2) As Double
    pt(0) = 0: pt(1) = 0: pt(2) = 0
    Dim objCircle1 As AcadCircle, objCircle2 As AcadCircle, objCircle3 As AcadCircle
    Set objCircle1 = ThisDrawing.ModelSpace.AddCircle(pt, 10)
    Set objCircle2 = ThisDrawing.ModelSpace.AddCircle(pt, 20)
    Set objCircle3 = ThisDrawing.ModelSpace.AddCircle(pt, 30)

    Dim objCollection(0 To 2) As AcadEntity
    Set objCollection(0) = objCircle1
    Set objCollection(1) = objCircle2
    Set objCollection(2) = objCircle3

    '测试编写的函数
```

```
    Call AddEntsToSSet(objCollection, SSet)
    MsgBox SSet.count

    '删除选择集
    SSet.Delete
End Sub
```

(2) 在 mdlSelectEnts 模块中,可以使用下面的宏对编写的函数进行测试(每次仅测试一个函数),其实现代码为:

```
Sub test( )
    '安全创建选择集
    On Error Resume Next
    Dim SSet As AcadSelectionSet
    If Not IsNull(ThisDrawing.SelectionSets.Item("Example")) Then
        Set SSet = ThisDrawing.SelectionSets.Item("Example")
        SSet.Delete
    End If
    Set SSet = ThisDrawing.SelectionSets.Add("Example")

    '测试多段线作为选择边界
    Dim objSelect As Object
    Dim ptPick As Variant
    ThisDrawing.Utility.GetEntity objSelect, ptPick

    Call SelectByPoly(SSet, objSelect, acSelectionSetWindowPolygon)

    '测试 SelectAtPoint 函数
    ' Dim pt(0 To 2) As Double
    ' pt(0) = 100: pt(1) = 100: pt(2) = 0
    ' Call SelectAtPoint(SSet, pt)

    '直接在图形窗口中选择
    'SSet.SelectOnScreen

    '测试选择集作用的区域
    ' Dim pt1 As Variant, pt2 As Variant
    ' Dim objUtility As Object
    ' Set objUtility = ThisDrawing.Utility          '必须使用后期绑定
    ' objUtility.CreateTypedArray pt1, vbDouble, 0, 0, 0
    ' objUtility.CreateTypedArray pt2, vbDouble, 100, 100, 0
    ' SSet.Select acSelectionSetCrossing, pt1, pt2

    MsgBox SSet.count

    '删除选择集
```

```
    SSet.Delete
End Sub
```

(3) 测试 Select 对象发现了一些问题。为了测试的方便，创建了如图 3.10 所示的图形。其中，矩形的两个角点分别为(0,0)和(100,100)。

在 Test 宏中，使用以下代码进行测试，其他的测试代码均将其注释掉(选择所要注释的代码，单击“编辑”工具栏中的“设置注释块”按钮)：

```
Dim pt1 As Variant, pt2 As Variant
Dim objUtility As Object
Set objUtility = ThisDrawing.Utility          '必须使用后期绑定
objUtility.CreateTypedArray pt1, vbDouble, 0, 0, 0
objUtility.CreateTypedArray pt2, vbDouble, 100, 100, 0
SSet.Select acSelectionSetCrossing, pt1, pt2
```

确保整个矩形都位于图形窗口内部，执行该宏，得到的结果如图 3.11 所示。这个结果容易理解，包含矩形在内，符合条件的对象是 5 个。

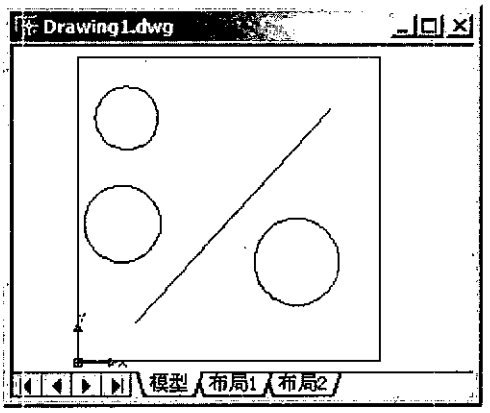


图 3.10　测试所用图形

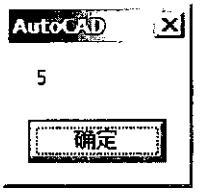


图 3.11　执行结果(一)

如果平移图形，使一些实体位于图形窗口外部，如图 3.12 所示。再次执行 Test 宏，能够得到如图 3.13 所示的结果。

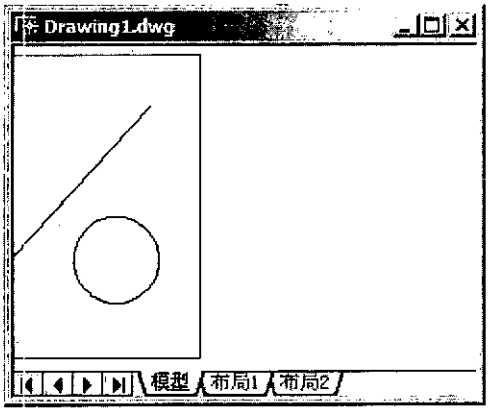


图 3.12　平移图形



图 3.13　执行结果(二)

这里的结果就让人费解了，矩形内部有 4 个对象，但是此处选择集仅获得了 3 个对象。实际上，使用 Select 方法仅能获得位于图形窗口（视口）内部的实体，对于视口外的实体不起作用。从图 3.12 中看，当前视口中确实包含了 3 个符合要求的实体，这个结果的含义也就明朗了。

经测试发现，SelectByPolygon 方法同样存在这个问题。

3.4.5 小结

使用选择集的过程中，最灵活的就是过滤器的构建，通过构建不同类型的过滤器，可以使用选择集获得符合多种条件的实体。但是，过滤器和 DXF 组码是紧紧联系在一起的，不了解组码知识，要灵活创建过滤器似乎非常困难。

实际上，只需要掌握本节列出的常用组码，其他的组码都可以通过下面的宏来获得（当然，也可以直接在命令行输入(entget(car(entsel)))并按下 Enter 键）：

```
Public Sub GetDXFGroup( )
    ThisDrawing.SendCommand "(entget(car(entsel))) "
End Sub
```

前面多次以圆作例子，那么现在就来看看如何使用 GetDXFGroup 宏来获得圆心、半径的组码。在图形窗口中创建一个圆心在（50，50，0）、半径为 30 的圆，然后执行 GetDXFGroup 宏，在图形窗口中选择创建的圆，能够在命令窗口中得到如图 3.14 所示的结果。

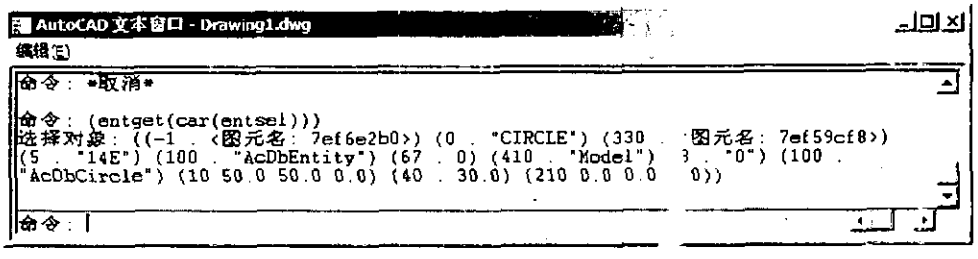


图 3.14 查看圆的 DXF 组码

组码和其对应的参数值成对出现，形如(组码. 参数值)的形式。从图 3.14 中可以获得几个常用的组码，例如对象类型(0. "CIRCLE")的组码 0，所在空间(67. 0)的组码 67，图层(8. "0")的组码 8。由于我们已经知道圆的圆心和半径，自然就很容易找到圆心的组码 10 和半径的组码 40。

3.5 使用 SendCommand 方法

3.5.1 实例说明

如果开发者对 AutoLISP 的开发技术有所了解的话，则对 AutoLISP 中的 Command 函数会有较深的印象。在 AutoLISP 开发中，Command 函数是用得最多的一个函数。

在 VBA 开发时，许多用户都希望能有这样一个与 Command 函数相同功能的方法，以便 VBA 的开发。

在 AutoCAD 的对象模型中，提供了 SendCommand 方法，从字面上应可理解为与 Command 函数一样的功能。但实际使用时却无法达到满意的效果，可以说，SendCommand 方法只是一个摆设，它不能完成复杂的编程工作。

它的缺点如下：

- 通过 SendCommand 方法发送的命令无法让它不在命令行中显示。这对于一个专业的程序来说，杂乱的命令行显示显得不够专业。
- 无法控制程序的执行顺序。SendCommand 方法只是单方面的向命令行发送命令，该命令发送完成后就直接执行后续的程序，而不管发送的命令执行是否完成，这样就有可能会使程序的执行顺序发生变化，而导致程序出错。
- SendCommand 方法发送的命令只能以字符串的方式发送，不能传递点坐标、双元表、图元对象及其他数据格式。这样就无法完成一些复杂的命令过程。

总之，AutoLISP 中的 Command 函数中的那些优点在 SendCommand 方法中都找不到。

但是，由于 AutoCAD 提供的对象模型到目前版本（AutoCAD 2006 版）都还不完善，许多功能都不能通过对象模型来完成编程，所以这里还需要花点精力来挖掘 SendCommand 方法的一些功能。

在《AutoCAD VBA 编程精彩实例》教程一书中，很少使用 SendCommand 方法来解决问题，主要是考虑到此方法本身的一些局限性，例如不能传递参数、导致宏执行顺序的错乱等。目前，经过大家的努力，已经通过和 AutoLISP 的结合解决了这些问题，本节要解决的是下面几个问题：

- 在 SendCommand 方法中使用参数。
- 使用 SendCommand 方法模拟断开和修剪命令。
- 讨论 SendCommand 方法引起的执行顺序混乱问题。

3.5.2　编程思路

在 VBA 中，可以通过 SendCommand 方法来发送命令给 AutoCAD 并对其直接进行操作，就像 LISP 中的 Command 函数一样，但它却没有 Command 函数这么方便。因为 Command 函数可以直接接受 LISP 命令以及 LISP 提供的点坐标形式以及双元表等，所以它能够通过程序对图元进行修剪、截断、延伸等操作，但 SendCommand 方法却不能接受 VBA 提供的点坐标，不能接受 VBA 中的其他方法，例如点坐标形式，而且也不允许使用双元表这样的特殊格式，它只能输入在命令行中能识别的点形式及双元表。

通过对 SendCommand 方法的研究不难发现，它可以接受与命令输入一模一样的语句，也就是说它也能够接受 AutoLISP 的函数，以及相同格式的点坐标。这样，就可以通过 VBA 来制造这样的点坐标，也可以通过 VBA 来生成一些供 SendCommand 方法使用的、组合起来的 AutoLISP 语句来表达的一个双元表。

在本节所介绍的实例中，专门创建了三个函数用来完成参数转换的工作：

- axPoint2lspPoint 是转换 VBA 的点为 SendCommand 用的点格式。

- axEnt2lspEnt 是转换 VBA 的图元为 SendCommand 中用于供选择的图元格式。
- GetDoubleEntTable 是转换 VBA 的图元及点为 SendCommand 中用的双元表格式。

3.5.3 步骤

(1) 在 AutoCAD 2005 中,打开 VBA 管理器,创建一个新工程,保存在适当的位置。进入 VBA 集成开发环境,双击"工程资源管理器"窗口中的 ThisDrawing 图标,打开代码窗口,添加一个函数 axPoint2lspPoint,用于将 VBA 中的点转换成 AutoLISP 中可以使用的点参数,其实现代码为:

```
Public Function axPoint2lspPoint(ByVal Pnt As Variant) As String
    axPoint2lspPoint = Pnt(0) & "," & Pnt(1) & "," & Pnt(2)
End Function
```

(2) 添加一个函数 axEnt2lspEnt,用于将 VBA 中的实体转换成 AutoLISP 中可以使用的参数,其实现代码为:

```
Public Function axEnt2lspEnt(ByVal entObj As AcadEntity) As String
    Dim entHandle As String
    entHandle = entObj.Handle
    axEnt2lspEnt = "(handent " & Chr(34) & entHandle & Chr(34) & ")"
End Function
```

其中,handent 是一个 AutoLISP 中的函数,根据指定的句柄返回实体名称。

(3) 添加一个函数 GetDoubleEntTable,用于将 VBA 中 GetEntity 方法返回的参数转换成可在 AutoLISP 中使用的形式,其实现代码为:

```
Public Function GetDoubleEntTable(ByVal entObj As AcadEntity, ByVal Pnt As Variant) As String
    Dim entHandle As String
    entHandle = entObj.Handle
    GetDoubleEntTable = "(list(handent " & Chr(34) & entHandle & Chr(34) & _
                        ")(list " & Str(Pnt(0)) & Str(Pnt(1)) & Str(Pnt(2)) & "))"
End Function
```

此函数根据 VBA 中 GetEntity 方法的返回值构建双元表,用于 AutoLISP 中。

"双元表",是 AutoLISP 特有的表数据类型之一,是一种针对 AutoCAD 数据库所特有的检索指针,就像是 (EntSel) 函数的返回值那样的数据结构。实际上也可以用 (List) 函数,按这种格式组成自定义的双元表。这种特有的数据结构,是在程序中使用 CAGD 技术的关键之一。

对于一些编辑命令,在指目标时,所指点位与处理的结果有关,就像在 AutoLISP 中应当用双元表来"精确响应"目标的提示,这也是 AutoLISP 特有的、相当有实用价值的技巧。这类命令有 DIM、EXTEND、TRIM、FILLET、CHAMFER、DIVIDE、MEASURE 等。

(4) 添加一个函数 axSSet2lspEnts,用于将 VBA 选择集中的实体转换成可在 AutoLISP 中使用的形式,其实现代码为:

```
Public Function axSSet2lspEnts(ByVal SSet As AcadSelectionSet) As String
    If SSet.Count = 0 Then Exit Function
```

```
    Dim entHandle As String
    Dim strEnts As String
    entHandle = SSet.Item(0).Handle
    strEnts = "(handent " & Chr(34) & entHandle & Chr(34) & ")"

    If SSet.Count > 1 Then
        Dim i As Integer
        For i = 1 To SSet.Count - 1
            entHandle = SSet.Item(i).Handle
            strEnts = strEnts & vbCr & "(handent " & Chr(34) & entHandle & Chr(34) & ")"
        Next i
    End If

    axSSet2lspEnts = strEnts
End Function
```

(5) 创建一个宏 Break,用于模拟 AutoCAD 中的 BREAK 命令,其实现代码为:

```
Public Sub Break( )
    '选择所要打断的实体(兼作第一点)
    Dim Pnt As Variant
    Dim entObj As AcadEntity
    ThisDrawing.Utility.GetEntity entObj, Pnt, vbCrLf & "选择图元: "

    '选择打断的第二点
    Dim Pnt2 As Variant
    Pnt2 = ThisDrawing.Utility.GetPoint(,vbCrLf & "选择点: ")

    '转换双元表
    Dim det As String
    det = GetDoubleEntTable(entObj, Pnt)

    '调用 SendCommand 方法进行操作
    Dim lspPnt As String
    lspPnt = axPoint2lspPoint(Pnt2)
    ThisDrawing.SendCommand "_break" & vbCr & det & vbCr & lspPnt & vbCr
End Sub
```

注意下面的一行代码:

```
ThisDrawing.Utility.GetEntity entObj, Pnt, vbCrLf & "选择图元: "
```

在命令行的提示中加上 vbCrLf 关键字,可以让跟在此关键字后面的内容换行显示,这在 *SendCommand* 和 *Prompt* 方法中均是有效的。

(6) 创建一个宏 Trim,用于模拟 AutoCAD 中的 TRIM 命令,其实现代码为:

```
Public Sub Trim( )
    '安全创建选择集
```

```
    On Error Resume Next
    Dim SSet As AcadSelectionSet
    If Not IsNull(ThisDrawing.SelectionSets.Item("trim")) Then
        Set SSet = ThisDrawing.SelectionSets.Item("trim")
        SSet.Delete
    End If
    Set SSet = ThisDrawing.SelectionSets.Add("trim")

    '选择实体,并将选择集中的数据转换成可在 AutoLISP 中使用的形式
    SSet.SelectOnScreen
    Dim det1 As String
    det1 = axSSet2lspEnts(SSet)

    SSet.Delete

    Dim Pnt2 As Variant
    Dim entObj2 As AcadEntity
    ThisDrawing.Utility.GetEntity entObj2, Pnt2, "选择被剪图元: "
    Dim det2 As String
    det2 = GetDoubleEntTable(entObj2, Pnt2)

    ThisDrawing.SendCommand "_trim" & vbCr & det1 & vbCr & vbCr & det2 & vbCr & vbCr
End Sub
```

3.5.4 实例效果

(1) 在 AutoCAD 中,使用 CIRCLE 命令创建一个圆。在 VBA 集成开发环境中,执行 Break 宏,按照命令提示,首先在图形窗口中选择实体(兼作第一个打断点),然后指定第二个打断点,完成操作后,就能对圆执行打断的操作。

(2) 重新创建一个圆和两条直线,其位置关系如图 3.15 所示。

(3) 在 VBA 集成开发环境中,执行 Trim 宏,首先选择两条直线作为修剪的边界,然后单击圆位于两条直线之间的部分,就能得到如图 3.16 所示的结果。

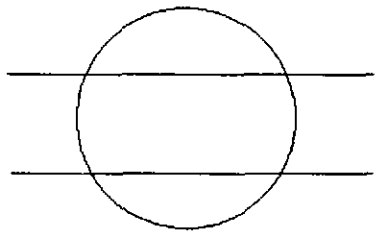


图 3.15 创建测试实体

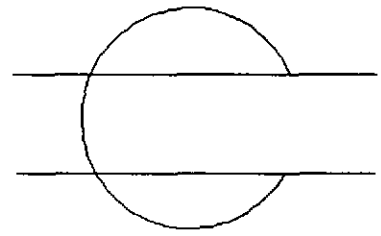


图 3.16 修剪实体的结果

3.5.5 小结

使用 SendCommand 方法简单直接,能在 AutoCAD 命令行中完成的操作,用 SendCommand 方法基本都能完成,但是 SendCommand 方法是将命令传给 AutoCAD,而不

等其执行完毕就直接运行后续的语句，所以经常会导致程序运行次序的混乱。

尽量不要使用 SendCommand 方法，如果用的话，最好也把它放在最后。再不行的话，把一个过程分成两段，第一段把 SendCommand 方法写在最后，接下再执行第二段。而第二段可以在第一段最后直接用程序调用。

在 SendCommand 方法中，可以使用空格或者 vbCr 表示按下 Enter 键，例如 line 和 line & vbCr 是等效的，但是在一些特殊的情况下空格键是无效的，因此建议在所有的情况下使用 vbCr 来表示按下了 Enter 键。

3.6 多 段 线

3.6.1 实例说明

多段线是 AutoCAD 中应用非常广泛的实体，在编程中可能需要对其进行多种操作，但是 VBA 中有很多功能不能直接实现，例如获得多段线的顶点数、合并多段线等。本节的实例探讨了若干常用的多段线操作，实现了下面的几个功能：

- 获得多段线的顶点数。
- 将若干首尾连接的多段线合并成一条多段线。
- 在闭合区域内单击一点，生成闭合多段线。
- 多段线的反转（形状不变，起点和终点颠倒，其他顶点按顺序颠倒，如图 3.17 所示）。

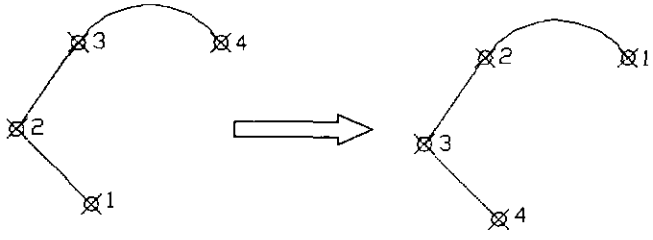


图 3.17　多段线反转的结果

3.6.2 编程思路

1. 获得顶点数、连接多段线、生成闭合多段线

普通多段线（AcadPolyline）和轻量多段线（AcadLWPolyline）均包含一个 Coordinates 方法，能够返回一个数组，其中包含了多段线所有顶点的坐标。普通多段线和轻量多段线的不同之处在于，普通多段线每个顶点保存三个坐标值（X、Y 和 Z 坐标），而轻量多段线每个顶点仅保存两个坐标值（X 和 Y 坐标）。将 Coordinates 方法返回的数组的维数除以每个顶点的坐标维数，即可获得多段线的顶点数。

ActiveX 模型本身没有提供连接多段线的功能，只能使用 SendCommand 方法，使用 AutoCAD 自身的命令 PEDIT 来完成连接多段线的操作。

在几个首尾相接的实体所形成的封闭区域内单击一点，创建该闭合区域的边界多段线，

ActiveX 模型本身也没有这种功能，只能使用 SendCommand 方法调用 AutoCAD 中的一 Boundary 命令来实现。

2. 多段线的反转

多段线的反转包括两个部分：顶点坐标的反转和凸度的反转。

对于图 3.17 所示的轻量多段线，在反转之前，其所有顶点的坐标数组（使用 Coordinates 方法即可获得）为：

(703,261,549,412,673,585,966,585)

从图形中观察，反转后的顶点坐标应改为：

(966,585,673,585,549,412,703,261)

从两个坐标数组可以看出，X 坐标反转后的索引是“数组维数＋1－当前索引－1”，Y 坐标反转后的索引是“数组维数＋1－当前索引＋1”，按照这个规律就可以根据原来的坐标数组获得反转后的坐标数组。

仍然以这条轻量多段线为例，在反转之前，其凸度数组为：

(0,0,－0.512,0)

从图形中观察，反转之后的凸度数组为：

(0.512,0,0,0)

结合图形进行分析，可以得出结论，凸度数组变换的规律是：将原来的凸度数组最后一个元素删除，在第一个元素之前插入一个元素 0；将变换后的凸度数组反转，在反转的过程中将不为 0 的凸度值取相反数。

3.6.3 步骤

(1) 在 AutoCAD 2005 中，打开 VBA 管理器，创建一个新工程，保存在适当的位置。进入 VBA 集成开发环境，在“工程资源管理器”窗口中双击 ThisDrawing 图标，打开代码窗口，添加一个函数 GetVertexCount，用于获得多段线的顶点数：

```
Private Function GetVertexCount(ByVal objPline As AcadEntity) As Long
    If TypeOf objPline Is AcadLWPolyline Then
        GetVertexCount = (UBound(objPline.Coordinates) + 1)/2
    ElseIf TypeOf objPline Is AcadPolyline Then
        GetVertexCount = (UBound(objPline.Coordinates) + 1)/3
    End If
End Function
```

(2) 添加一个宏 JoinPoly，提示用户选择若干首尾连接的多段线，将其合并成一条多段线，其实现代码为：

```
Public Sub JoinPoly( )
    '提示用户选择多条需要连接的多段线
    On Error Resume Next
    Dim SSet As AcadSelectionSet
    If Not IsNull(ThisDrawing.SelectionSets.Item("JoinPoly")) Then
        Set SSet = ThisDrawing.SelectionSets.Item("JoinPoly")
```

```
        SSet.Delete
    End If
    Set SSet = ThisDrawing.SelectionSets.Add("JoinPoly")

    '选择实体,并将选择集中的数据转换成可在 AutoLISP 中使用的形式
    SSet.SelectOnScreen
    Dim det As String
    det = axSSet2lspEnts(SSet)
    SSet.Delete

    '使用 SendCommand 方法完成连接操作
    ThisDrawing.SendCommand "_PEDIT" & vbCr & "M" & vbCr & det & vbCr & vbCr & "J" & vbCr & vbCr & vbCr
End Sub
```

其中使用了自定义函数 axSSet2lspEnts,该函数用于将选择集中的多个实体转化成可在 AutoLISP 中使用的变量,其定义为:

```
Public Function axSSet2lspEnts(ByVal SSet As AcadSelectionSet) As String
    If SSet.Count = 0 Then Exit Function

    Dim entHandle As String
    Dim strEnts As String
    entHandle = SSet.Item(0).Handle
    strEnts = "(handent " & Chr(34) & entHandle & Chr(34) & ")"

    If SSet.Count > 1 Then
        Dim i As Integer
        For i = 1 To SSet.Count - 1
            entHandle = SSet.Item(i).Handle
            strEnts = strEnts & vbCr & "(handent " & Chr(34) & entHandle & Chr(34) & ")"
        Next i
    End If

    axSSet2lspEnts = strEnts
End Function
```

(3) 添加一个宏 ClickAddPolyline,执行该宏之后,用户在若干首尾相接的实体形成的封闭区域内单击,就能创建该区域的边界多段线,其实现代码为:

```
Public Sub ClickAddPolyline( )
    '获得当前的实体数量
    Dim n As Long
    n = ThisDrawing.ModelSpace.Count

    '提示用户拾取一点
    Dim pt As Variant
    pt = ThisDrawing.Utility.GetPoint(, "指定内部点: ")
```

```
    '创建多段线
    ThisDrawing.SendCommand "-Boundary" & vbCr & pt(0) & "," & pt(1) & vbCr & vbCr

    '将生成的多段线颜色变为红色
    Dim objPoly As AcadLWPolyline
    If ThisDrawing.ModelSpace.Count > n Then
        Set objPoly = ThisDrawing.ModelSpace.Item(ThisDrawing.ModelSpace.Count - 1)
        objPoly.color = acRed
    Else
        MsgBox "未发现有效的边界。"
    End If
End Sub
```

(4) 添加一个函数 GetAllBulges,用于获得多段线的凸度数组,将其保存在一个集合中,其实现代码为:

```
Private Function GetAllBulges(ByVal objPoly As AcadEntity) As Collection
    If TypeOf objPoly Is AcadLWPolyline Or TypeOf objPoly Is AcadPolyline Then
        Dim bulgeCollection As New Collection
        Dim i As Long
        For i = 0 To GetVertexCount(objPoly) - 1
            bulgeCollection.Add objPoly.GetBulge(i)
        Next i

        Set GetAllBulges = bulgeCollection
    Else
        MsgBox "objPoly 不是多段线!"
    End If
End Function
```

GetBulge 函数用于获得多段线上指定顶点的凸度,其具体含义可以参考 3.5 节(直线、圆和圆弧转化为多段线)的内容。

(5) 添加一个函数 RevCollection,用于反转集合中的元素,并将不为 0 的元素取相反数,其实现代码为:

```
Private Function RevCollection(ByVal bulgeCollection As Collection) As Collection
    Dim newCollection As New Collection

    Dim i As Long
    For i = 1 To bulgeCollection.Count
        Dim bulge As Double
        bulge = bulgeCollection.Item(bulgeCollection.Count + 1 - i)
        If bulge <> 0 Then
            newCollection.Add - bulgeCollection.Item(bulgeCollection.Count + 1 - i)
        Else
            newCollection.Add 0
```

```
        End If
    Next i

    Set RevCollection = newCollection
End Function
```

(6) 添加一个函数 SetAllBulges，用于将给定的凸度数组应用于多段线，其实现代码为：

```
Private Sub SetAllBulges(ByVal objPoly As AcadEntity, ByVal bulgeCollection As Collection)
    If TypeOf objPoly Is AcadLWPolyline Or TypeOf objPoly Is AcadPolyline Then
        Dim i As Long
        For i = 0 To GetVertexCount(objPoly) - 1
            objPoly.SetBulge i, bulgeCollection(i + 1)
        Next i
    Else
        MsgBox "objPoly 不是多段线!"
    End If
End Sub
```

SetBulge 函数用于设置多段线上指定顶点的凸度，其具体含义可以参考 3.7 节(直线、圆和圆弧转化为多段线)的内容。

(7) 创建一个宏 RevPline，用于反转多段线的操作，其实现代码为：

```
Public Sub RevPline( )
    Dim ent As AcadEntity
    Dim pnt As Variant
    Dim NewCoord( ) As Double     '新的顶点坐标数组
    Dim i As Integer

    '拾取多段线
    On Error Resume Next
    Do
        ThisDrawing.Utility.GetEntity ent, pnt, "选择多段线："
        If Err Then Exit Sub
        If TypeName(ent) Like "IAcad * Polyline" Then Exit Do
    Loop

    '反转多段线的顶点
    Dim Coord As Variant     '原来的顶点坐标数组

    If TypeOf ent Is AcadLWPolyline Then
        Coord = ent.Coordinates
        ReDim NewCoord(UBound(Coord)) As Double
        For i = 0 To UBound(Coord) - 1 Step 2
            NewCoord(UBound(Coord) - i - 1) = Coord(i)
            NewCoord(UBound(Coord) - i) = Coord(i + 1)
```

```
        Next
    ElseIf TypeOf ent Is AcadPolyline Then
        Coord = ent.Coordinates
        ReDim NewCoord(UBound(Coord)) As Double
        For i = 0 To UBound(Coord) - 1 Step 3
            NewCoord(UBound(Coord) - i - 2) = Coord(i)
            NewCoord(UBound(Coord) - i - 1) = Coord(i + 1)
            NewCoord(UBound(Coord) - i) = Coord(i + 2)
        Next
    End If

    ent.Coordinates = NewCoord                          '更新顶点坐标集合

    '修改凸度
    Dim bulgeCollection As New Collection
    Set bulgeCollection = GetAllBulges(ent)             '获得凸度集合
    '删除最后一个元素,在开头增加一个 0 元素
    bulgeCollection.Remove bulgeCollection.Count
    bulgeCollection.Add 0, , 1
    Dim newBulges As New Collection                     '新的凸度集合
    Set newBulges = RevCollection(bulgeCollection)      '反转集合
    Call SetAllBulges(ent, newBulges)                   '重新设置凸度

    ThisDrawing.Regen acActiveViewport

    End
End Sub
```

3.6.4 实例效果

(1) 在 VBA 集成开发环境中,添加一个宏 TestVertexCount,测试 GetVertexCount 函数,其实现代码为:

```
Public Sub TestVertexCount( )
    Dim objSelect As Object
    Dim ptPick As Variant
    ThisDrawing.Utility.GetEntity objSelect, ptPick, "选择多段线:"

    If TypeOf objSelect Is AcadLWPolyline Then
        MsgBox GetVertexCount(objSelect)
    End If
End Sub
```

执行该宏之后,在图形窗口中选择一条多段线,即可在弹出的对话框中获得该多段线的顶点数。

(2) 执行 JoinPoly 宏,在图形窗口中选择若干首尾连接的多段线,即可将其合并成一条

多段线。

(3) 在图形窗口中创建若干首尾连接且形成闭合区域的实体，执行 ClickAddPolyline 宏，在闭合区域内部单击一点，即可创建该区域的边界多段线。

(4) 在图形窗口中创建一条多段线(包含圆弧段)，在 AutoCAD 中选择“工具”|“对象特性管理器”菜单项，在弹出的“属性”窗口中查看其顶点坐标，如图 3.18 所示。

(5) 执行 RevPline 宏，在图形窗口中选择步骤(4)创建的多段线，即可对其进行反转操作。操作完成后，多段线的形状不会改变，但是在属性窗口中可以观察到其坐标已经反转过来。

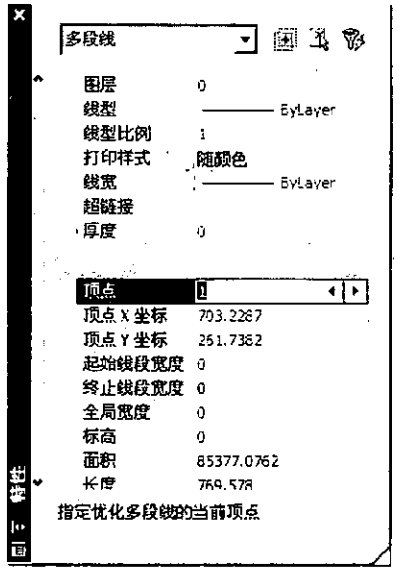


图 3.18　查看多段线的顶点坐标

3.6.5　小结

本节所介绍的不少知识经过一点改动后，就可以实现其他的功能，最典型的就是 ClickAddPolyline 宏(单击一点获得闭合多段线)，经过改动后可以实现单击一点创建面域、单击一点创建图案填充、单击一点获得闭合区域的面积等，下面的代码实现了单击一点获得闭合区域的面积的功能。

```
Public Sub GetArea( )
    '当前图样的实体数目
    Dim n As Long
    n = ThisDrawing.ModelSpace.Count

    '调用 Boundary 命令获取某一点处的边界
    Dim pt As Variant
    pt = ThisDrawing.Utility.GetPoint(, "指定内部点：")
    ThisDrawing.SendCommand "-Boundary" & vbCr & pt(0) & "," & pt(1) & vbCr & vbCr

    '如果存在边界，则会生成新的实体
    Dim objPoly As AcadLWPolyline
    If ThisDrawing.ModelSpace.Count > n Then
        Set objPoly = ThisDrawing.ModelSpace.Item(ThisDrawing.ModelSpace.Count - 1)
        MsgBox objPoly.Area
        objPoly.Delete
    Else
        MsgBox "未发现有效的边界。"
    End If
End Sub
```

3.7 直线、圆和圆弧转化为多段线

3.7.1 实例说明

在 AutoCAD 中，直线、圆和圆弧都不带有线宽的信息，那么在需要具有线宽的场合，只能将其转化成多段线来处理。本节的实例能够将选择的直线、圆和圆弧转化成对应的多段线，得到的多段线与原来的实体形状完全相同。

3.7.2 编程思路

1. 基础知识

将直线、圆和圆弧转化为多段线，实际上是获得这些图形对象的关键参数，创建一个与原来的实体形状相同的多段线，再删除原来的对象即可。

根据直线创建一条多段线比较简单，只需获得直线的起点和终点，然后创建仅包含一条直线段的多段线即可。

根据圆创建一条多段线，需要获得圆的圆心和半径，然后创建一条具有三个顶点的多段线，将各个顶点的凸度都设置为 1 即可。

根据圆弧创建一条多段线，需要获得圆弧的圆心、半径、起始角度和终止角度，然后创建一条具有两个顶点的多段线，设置第一个顶点的凸度为圆弧所包含角度的 1/4 的正切值，第二个顶点的凸度为 0。

凸度是在多段线中指定顶点与下一个顶点之间形成的弧之间角度的 1/4 的正切值，用于设置一段多段线的弯曲程度。在 ActiveX 对象模型中，SetBulge 方法用于设置多段线某一段的凸出参数。凸度为 0 代表一个直线段，凸度为 1 代表一个半圆段，介于 0～1 之间为劣弧，大于 1 为优弧。

2. 设计思路

这个程序设计过程中最难确定的就是根据圆弧创建多段线的问题，创建的多段线究竟应该是几个顶点？两个还是三个？每个顶点的凸度应该如何设置？

(1) 可以直接观察系统生成的带有圆弧的多段线的凸度，程序代码如下所示(这段代码放置在 ThisDrawing 模块中)：

```
Private Const VK_ESCAPE = &H1B                '代表 Esc 键
Private Declare Function GetAsyncKeyState Lib "user32" (ByVal vKey As Long) As Integer

    '功能：判断用户是否按某个键
    '输入：代表键的常量(从 API Viewer 中获得)
    '调用：API 函数 GetAsyncKeyState
    '返回：如果用户按下了指定的键，返回 True；否则返回 False
    '示例：' If CheckKey(&H1B) = True Then do sth
    Private Function CheckKey(lngKey As Long) As Boolean
```

```
        If GetAsyncKeyState(lngKey) Then
            CheckKey = True
        Else
            CheckKey = False
        End If
    End Function

    '设计过程中的测试
    Sub Test( )
        Dim objSelect As AcadEntity
        Dim basePnt As Variant
        '选择对象并判断类型
        On Error Resume Next

    Retry:
    ThisDrawing.Utility.GetEntity objSelect, basePnt, "选择对象："

    '处理按 Esc 键的错误
    If objSelect Is Nothing Then
        If CheckKey(VK_ESCAPE) = True Then
            Exit Sub
        Else
            GoTo Retry
        End If
    End If

    '处理未选择到实体的错误
    If Err <> 0 Then
        Err.Clear
        GoTo Retry
    End If

    '执行用户的操作
    Dim var As Variant
    var = objSelect.Coordinates

    Dim i As Integer
    Debug.Print "Start"
    If TypeOf objSelect Is AcadLWPolyline Then
        For i = 0 To 3
            Debug.Print objSelect.GetBulge(i) & vbCrLf
        Next i
    End If
    Debug.Print "End"
End Sub
```

上面的这段代码在书中已经多次使用，复杂的部分无非是加上了对 GetEntity 方法的错误处理，核心的部分只有下面几句：

```
If TypeOf objSelect Is AcadLWPolyline Then
    For i = 0 To 3
        Debug.Print objSelect.GetBulge(i) & vbCrLf
    Next i
End If
```

很明显，它的意图是显示一条多段线各个顶点的凸度。

(2) 在 AutoCAD 中执行 PLINE 命令，按照命令提示进行操作：

```
命令：pl PLINE
指定起点：
当前线宽为 0.0000
指定下一个点或[圆弧(A)/半宽(H)/长度(L)/放弃(U)/宽度(W)]：
指定下一点或[圆弧(A)/闭合(C)/半宽(H)/长度(L)/放弃(U)/宽度(W)]：a
指定圆弧的端点或[角度(A)/圆心(CE)/闭合(CL)/方向(D)/半宽(H)/直线(L)/半径(R)/第二个点
    (S)/放弃(U)/宽度(W)]：
指定圆弧的端点或[角度(A)/圆心(CE)/闭合(CL)/方向(D)/半宽(H)/直线(L)/半径(R)/第二个点
    (S)/放弃(U)/宽度(W)]：l    【结果 1：可以在这里不输入 L，而是直接按 Enter 键完成创建】
指定下一点或[圆弧(A)/闭合(C)/半宽(H)/长度(L)/放弃(U)/宽度(W)]：    【结果 2】
```

无论在“结果 1”处还是在“结果 2”处结束 PLINE 命令的执行，都会得到同样形状的一条多段线，如图 3.19 所示。

(3) 执行测试宏 Test，选择步骤(2)创建的多段线，能够在 VBA 集成开发环境的立即窗口中得到多段线每一段的凸度，如图 3.20 所示。

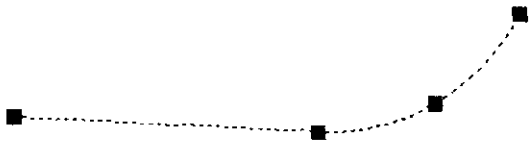


图 3.19 创建多段线的结果

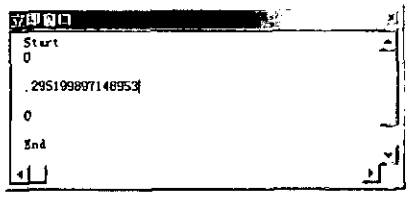


图 3.20 显示多段线每一段的凸度

如果在创建多段线时从“结果 1”处结束执行 PLINE 命令，那么得到的第三个顶点处的凸度肯定不是 0。但是，无论第三个顶点的凸度是多少，都不会对这条多段线的形状产生影响，因为第三个顶点影响的是多段线第三段的形状(第三段实际上不存在)。

(4) 在 AutoCAD 中，单击“修改”工具栏中的“打断于点”按钮，按照命令提示进行操作。

```
命令：_break 选择对象：        【选择多段线】
指定第二个打断点 或[第一点(F)]：_f
指定第一个打断点：    【选择多段线的第二个顶点，也就是直线和圆弧的公共点】
指定第二个打断点：@
```

完成操作后，多段线被分成直线和圆弧两部分。

(5) 再次执行 Test 宏，选择圆弧部分的多段线，在 VBA 集成开发环境的立即窗口中得

到如图 3.21 所示的结果。

到此为止，可以得出结论：要根据圆弧创建多段线，可以创建一条包含两个顶点的多段线，起点和终点与圆弧的两端点重合；根据圆弧包含的角度设置多段线的凸度，设置第一个顶点的凸度为圆弧所包含角度的 1/4 的正切值，第二个顶点的凸度为 0（设置为其他的数值也可以）。

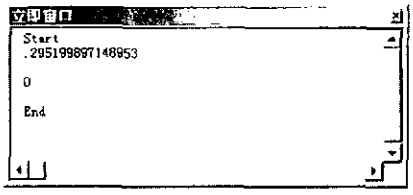


图 3.21　仅包含圆弧的多段线的各顶点的凸度

3.7.3　步骤

（1）在 VBA 集成开发环境中，创建一个新工程，保存在适当的位置。选择“插入”|“模块”菜单项，向当前工程中添加一个标准模块，在其中创建一个函数 AddLWPlineSeg，用于创建一条仅包含一个直线段的多段线，其实现代码为：

```
Private Function AddLWPlineSeg(ByVal ptSt As Variant,
    ByVal ptEn As Variant, ByVal width As Double) As AcadLWPolyline
    Dim objPline As AcadLWPolyline
    Dim ptArr(0 To 3) As Double

    ptArr(0) = ptSt(0)
    ptArr(1) = ptSt(1)
    ptArr(2) = ptEn(0)
    ptArr(3) = ptEn(1)

    Set objPline = ThisDrawing.ModelSpace.AddLightWeightPolyline(ptArr)
    objPline.ConstantWidth = width
    objPline.Update
    Set AddLWPlineSeg = objPline
End Function
```

（2）创建一个函数 AddLWPlineCircle，用于创建一条多段线，该多段线的形状是一个圆，其实现代码为：

```
Private Function AddLWPlineCircle(ByVal ptCen As Variant, ByVal radius As Double, ByVal width
    As Double) As AcadLWPolyline
    Dim objPline As AcadLWPolyline
    Dim ptArr(0 To 5) As Double

    ptArr(0) = ptCen(0) + radius
    ptArr(1) = ptCen(1)
    ptArr(2) = ptCen(0) - radius
    ptArr(3) = ptCen(1)
    ptArr(4) = ptCen(0) + radius
    ptArr(5) = ptCen(1)

    Set objPline = ThisDrawing.ModelSpace.AddLightWeightPolyline(ptArr)
```

```
    objPline.ConstantWidth = width

    '将各顶点的凸度均设置为 1
    objPline.SetBulge 0, 1
    objPline.SetBulge 1, 1
    objPline.SetBulge 2, 1

    objPline.Closed = True
    objPline.Update

    Set AddLWPlineCircle = objPline
End Function
```

(3) 创建一个函数 AddLWPlineArc,能够根据指定的圆弧参数,创建一个形状与圆弧一致的多段线,其实现代码为:

```
Private Function AddLWPlineArc(ByVal ptCen As Variant, ByVal radius As Double,
    ByVal angleSt As Double, ByVal angleEn As Double, _
                                    ByVal width As Double) As AcadLWPolyline
    Dim objPline As AcadLWPolyline
    Dim ptArr(0 To 3) As Double

    ptArr(0) = ptCen(0) + radius * Cos(angleSt)
    ptArr(1) = ptCen(1) + radius * Sin(angleSt)
    ptArr(2) = ptCen(0) + radius * Cos(angleEn)
    ptArr(3) = ptCen(1) + radius * Sin(angleEn)

    Set objPline = ThisDrawing.ModelSpace.AddLightWeightPolyline(ptArr)
    objPline.ConstantWidth = width

    '如果圆弧通过了 0°
    If angleEn < angleSt Then
        angleSt = angleSt - 8 * Atn(1)
    End If

    '注意凸度的含义
    objPline.SetBulge 0, Tan((angleEn - angleSt)/4)
    objPline.SetBulge 1, 0

    objPline.Update

    Set AddLWPlineArc = objPline
End Function
```

(4) 创建一个宏 TransformToPolyline,提示用户选择所要转化的对象,将选中的实体修改为相同形状的多段线,其实现代码为:

```
Public Sub TransformToPolyline( )
    '创建选择集
    On Error Resume Next
    Dim SSet As AcadSelectionSet
    If Not IsNull(ThisDrawing.SelectionSets.Item("Example")) Then
        Set SSet = ThisDrawing.SelectionSets.Item("Example")
        SSet.Delete        '及时删除不用的选择集非常重要
    End If
    Set SSet = ThisDrawing.SelectionSets.Add("Example")

    '设置选择过滤器
    Dim FilterType(0 To 6) As Integer
    Dim FilterData(0 To 6) As Variant
    FilterType(0) = -4
    FilterData(0) = "<or"      '逻辑运算符的开始,表示一个或多个运算对象
    FilterType(1) = 0
    FilterData(1) = "Arc"
    FilterType(2) = 0
    FilterData(2) = "Circle"
    FilterType(3) = 0
    FilterData(3) = "Line"
    FilterType(4) = 0
    FilterData(4) = "Polyline"
    FilterType(5) = 0
    FilterData(5) = "LWPolyline"
    FilterType(6) = -4
    FilterData(6) = "or>"      '逻辑运算符的结束

    '提示用户选择对象
    ThisDrawing.Utility.Prompt "选择要改变线宽的对象(直线、圆、弧和多段线):"
    SSet.SelectOnScreen FilterType, FilterData

    Dim width As Double
    width = ThisDrawing.Utility.GetReal("输入对象的线宽:")

    '遍历选择集中的对象
    Dim ent As AcadEntity
    Dim objPline As AcadLWPolyline
    Dim ptStart, ptCenter, ptEnd
    Dim radius As Double
    Dim angleSt As Double, angleEn As Double

    For Each ent In SSet
        Select Case ent.ObjectName
            Case "AcDbLine"        '获取直线的两个端点
```

```
                ptStart = ent.StartPoint
                ptEnd = ent.EndPoint

                AddLWPlineSeg ptStart, ptEnd, width
                ent.Delete

            Case "AcDbArc"  '获取圆弧的圆心、半径、起始角度和终止角度
                ptCenter = ent.Center
                radius = ent.radius
                angleSt = ent.StartAngle
                angleEn = ent.EndAngle

                AddLWPlineArc ptCenter, radius, angleSt, angleEn, width
                ent.Delete

            Case "AcDbCircle"       '获取圆的圆心和半径
                ptCenter = ent.Center
                radius = ent.radius

                AddLWPlineCircle ptCenter, radius, width
                ent.Delete

            Case "AcDb2dPolyline", "AcDb3dPolyline", "AcDbPolyline"
                ent.ConstantWidth = width
                ent.Update
        End Select
    Next ent

    '删除选择集
    SSet.Delete
End Sub
```

3.7.4 实例效果

(1) 在 AutoCAD 2005 中,创建若干个直线、圆和圆弧对象,如图 3.22 所示。

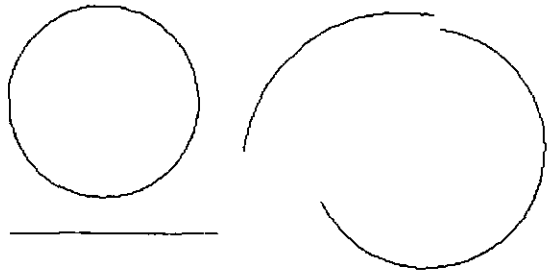


图 3.22 创建若干直线、圆和圆弧

(2) 在 VBA 集成开发环境中，将光标定位到 TransformToPolyline 宏内部，按 F5 键运行该宏，然后在图形窗口中选择步骤(1)创建的对象，并按 Enter 键，当系统提示“输入对象的线宽：”时，输入 4 后按 Enter 键。完成操作后，得到如图 3.23 所示的结果。

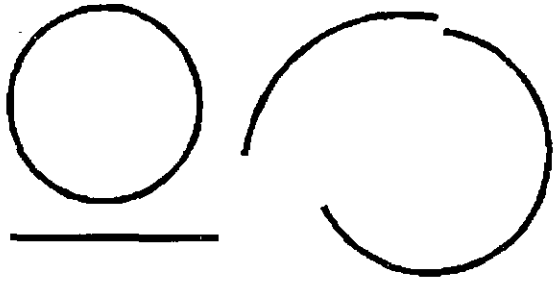


图 3.23　将对象线宽设置为 4 的结果

3.7.5　小结

学习本节的内容之后，应该熟练掌握下面的几个知识点：

- 安全创建和使用选择集。
- 选择集过滤器中逻辑运算符的使用。
- 合理使用凸度来修改多段线的形状。

3.8　样条曲线转化为多段线

3.8.1　实例说明

在很多情况下都需要将样条曲线转化为多段线，但是 AutoCAD 自身没有提供这样的功能，本节的实例以 AutoCAD R12 版的 DXF 文件为中介，实现将样条曲线转化为多段线的功能，并且保证了足够的精度。

3.8.2　编程思路

样条曲线是 AutoCAD R13 版开始引进的一种实体，并不属于 AutoCAD 初期的几种实体，而在 AutoCAD 2002 以后的版本，甚至 AutoCAD 2005 中仍然提供了将图形另存为 AutoCAD R12 版 DXF 文件的功能。

在文件的向后兼容性上，Autodesk 一直都做得不错，通过将图形另存为 R12 版 DXF 文件，图形中存在的 R12 版不支持的实体都被转化成其支持的实体类型。通过手工测试，发现样条曲线在这个过程中被转化为多段线，这就是本节程序的理论基础。

将样条曲线转化为多段线，可以使用下面的操作步骤：

(1) 创建一个选择集，选择图形中需要转换的样条曲线，将选择集中的对象导出到外部块(实际上就是一个 DWG 文件)中。

(2) 打开步骤(1)生成的外部块(DWG 文件)，然后将其另存为 AutoCAD R12 版的 DXF 文件，关闭该图形文件。

(3) 将 DXF 文件导入到当前图形中，并且删除原来图形中所要转换的样条曲线，被删除样条曲线的位置被导入的多段线代替。

3.8.3 步骤

(1) 在 AutoCAD 2005 中，打开 VBA 管理器，创建一个新工程，保存在适当的位置。进入 VBA 集成开发环境，双击“工程资源管理器”窗口中的 ThisDrawing 图标，打开代码窗口，首先使用 Visual Studio 6.0 的一个工具软件 API Text Viewer 获得 Windows API 函数 GetTempPath 的声明，将其添加到代码窗口中：

```
Private Declare Function GetTempPath Lib "kernel32" Alias "GetTempPathA" _
    (ByVal nBufferLength As Long, ByVal lpBuffer As String) As Long
```

该函数的作用是获得当前操作系统中临时文件夹的位置。

(2) 添加一个函数 CreateSSetFilter，这个函数实际上在 3.4 节(选择集)中已经介绍过，用于简化选择集过滤器的操作，实现代码为：

```
Public Sub CreateSSetFilter(ByRef filterType As Variant, ByRef filterData As Variant,
    ParamArray filter( ))
    If UBound(filter) Mod 2 = 0 Then
        MsgBox "filter 参数无效!"
        Exit Sub
    End If

    Dim fType( ) As Integer      '过滤器规则
    Dim fData( ) As Variant      '过滤器参数
    Dim count As Integer
    count = (UBound(filter) + 1)/2
    ReDim fType(count - 1)
    ReDim fData(count - 1)

    Dim i As Integer
    For i = 0 To count - 1
        fType(i) = filter(2 * i)
        fData(i) = filter(2 * i + 1)
    Next i

    filterType = fType
    filterData = fData
End Sub
```

(3) 添加一个宏 Spline2Pline，提示用户选择样条曲线，将其转化为多段线，其实现代码为：

```
Public Sub Spline2Pline( )
    '获得 Windows 临时文件夹的位置
    Dim tempDir As String
```

```
Dim retCode As Long
tempDir = Space(256)
retCode = GetTempPath(Len(tempDir), tempDir)
If retCode = 0 Then
    MsgBox "获得 Windows 临时文件夹失败!"
    Exit Sub
End If
tempDir = Left(Trim$(tempDir), Len(Trim$(tempDir)) - 1)

'安全创建选择集
On Error Resume Next
Dim SSet As AcadSelectionSet
If Not IsNull(ThisDrawing.SelectionSets.Item("Spline2Pline")) Then
    Set SSet = ThisDrawing.SelectionSets.Item("Spline2Pline")
    SSet.Delete
End If
Set SSet = ThisDrawing.SelectionSets.Add("Spline2Pline")

'构建过滤器
Dim fType As Variant, fData As Variant
Call CreateSSetFilter(fType, fData, 0, "Spline")

'提示用户选择样条曲线
SSet.SelectOnScreen fType, fData

'将选择集中的对象导出为外部块
ThisDrawing.Wblock tempDir & "temp.dwg", SSet
SSet.Erase          '删除要转化的样条曲线
SSet.Delete

'打开导出的外部块,将其转化为 R12 DXF
ThisDrawing.Application.Documents.Open tempDir & "temp.dwg"
ThisDrawing.SaveAs tempDir & "temp.dxf", acR12_dxf
ThisDrawing.Close

'重新导入创建的 DXF 文件
Dim pt(0 To 2) As Double
pt(0) = 0: pt(1) = 0: pt(2) = 0
ThisDrawing.Import tempDir & "temp.dxf", pt, 1

'删除临时文件
If Len(Dir(tempDir & "temp.dwg")) > 0 Then
    Kill tempDir & "temp.dwg"
End If
If Len(Dir(tempDir & "temp.dxf")) > 0 Then
```

```
        Kill tempDir & "temp.dxf"
    End If
End Sub
```

首先用 GetTempPath 函数获得当前操作系统的临时文件夹位置，如果函数的返回值为0，表示函数执行失败。在调用该函数时，必须首先指定一个足够大小的字符串来放置所获得路径，因此使用 Space 函数定义了一个包含 256 个空白字符的字符串，用来存放临时文件夹所在的路径。之所以定义长为 256 的字符串，是因为 Windows 操作系统能够支持的最长路径为 256 个字符。

CreateSSetFilter 函数构建的过滤器非常简单，仅选择当前图形中符合条件的样条曲线，而不能选择其他类型的实体。Wblock 方法可以将一个选择集内的所有实体写入一个新图形。

打开新图形使用了 Documents 集合的 Open 方法，将图形导出为 R12 版 DXF 文件使用了文档对象的 SaveAs 方法，完成操作后使用 Close 方法关闭了图形。

最终，使用文档对象的 Import 方法将前面导出的 DXF 文件导入到当前图形，这样 DXF 文件中的多段线就取代了原来样条曲线的位置。

程序的最后，使用 Kill 函数删除了宏执行过程中创建的临时文件，不能给系统留下垃圾文件。

3.8.4 实例效果

在 AutoCAD 2005 中，使用 SPLINE 命令可以创建若干条样条曲线。进入 VBA 集成开发环境，为了观察转换的精度，特将 Spline2Pline 中的一句代码作为注释，这样生成多段线的同时不会删除原来的样条曲线，可以对比两者的轮廓来查看精度。转化之后的结果如图 3.24 所示。

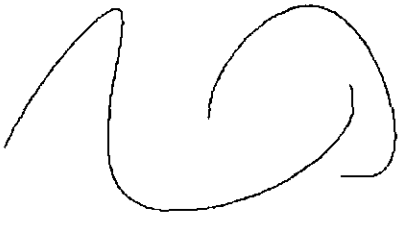


图 3.24 样条曲线转化为多段线的结果

3.8.5 小结

很多情况下，VBA 编程的用途是为了提高工作效率，因此其执行步骤可以看作是手工操作步骤的翻版（利用这种方法，手工操作同样可以完成样条曲线到多段线的转换），SendCommand 方法同样是手工操作的程序实现。

在 ActiveX 模型没有提供相关访问接口的情况下，在程序中执行这些手工操作的步骤，不失为一种好的解决方案。

3.9 在 AutoCAD 2004 以前版本中计算优化多段线长度

3.9.1 实例说明

虽然从 AutoCAD 2004 版本后，ActiveX 模型为多段线增加了 Length 属性来获得其长度，但在该版本之前，并没有直接取得多段线长度的方法。

本节将利用多段线的各种属性和方法，计算轻量(优化)多段线的长度，同时也通过本节了解多段线的结构。

本节程序在 AutoCAD 2004 及以上版本中均可正常运行。

3.9.2 编程思路

对于普通的多段线(指未经拟合或样条化的多段线)，具有各顶点坐标以及凸度等数据。

多段线的各段可以是直线段，也可以是弧段。它们是通过凸度来控制的，凸度为 0 时表明该段为直线段，凸度不等于 0 则表示该段为弧段。圆弧的大小长度与凸度值有最直接的关系。

本节将通过多段线上各段的起点、终点及凸度来计算第一段的长度，计算各段的长度之和，也就是多段线的总长度。

3.9.3 步骤

(1) 在 AutoCAD 中，打开 VBA 管理器，新建一个工程，保存在适当的位置，进入 VBA 集成开发环境。

(2) 在“工程资源管理器”窗口中双击 ThisDrawing 图标，打开其代码窗口，在其中添加宏的实现代码：

```
Sub PolyLineLength( )
    Dim Pnt As Variant
    Dim Ent As AcadEntity
    On Error Resume Next
    '提示用户选择多段线，程序中判断选定的对象是否为多段线
    '如果不是多段线则重复提示用户选择对象，如果未选中对象
    '则退出程序
    Do
        ThisDrawing.Utility.GetEntity Ent, Pnt, "选择多段线："
        If Err Then Exit Sub
        If TypeName(Ent) Like "IAcad * Polyline" Then Exit Do
    Loop
    '计算多段线的顶点数，多段线分为优化多段线、二维多段线和三维多段线
    '这里分别进行计算
    Dim CoordinateCount As Long
    '计算优化多段线的顶点数
    If TypeName(Ent) = "IAcadLWPolyline" Then
        CoordinateCount = (UBound(Ent.Coordinates) + 1)/2
    '计算多段线的顶点数，注意这里只能计算未拟合或样条化的多段线
    ElseIf (TypeName(Ent) = "IAcadPolyline" And Ent.Type = acSimplePoly) Or _
    (TypeName(Ent) = "IAcad3DPolyline" And Ent.Type = acSimple3DPoly) Then
        CoordinateCount = (UBound(Ent.Coordinates) + 1)/3
    Else
        Exit Sub
    End If
    Dim i As Long
```

```
    Dim TotalLength As Double
    Dim Bugle As Double
    '将每段线段的长度累加
        '计算三维多段线的长度,三维多段线并没有凸度,所以设凸度为 0
    If TypeName(Ent) = "IAcad3DPolyline" Then
        For i = 0 To CoordinateCount - 2
            TotalLength = TotalLength + GetArcLeng(Ent.Coordinate(i), Ent.Coordinate(i + 1), 0)
        Next
                '多段线闭合时,需要计算最后闭合段的长度
        If Ent.Closed Then TotalLength = TotalLength + _
            GetArcLeng(Ent.Coordinate(CoordinateCount - 1), _
                                    Ent.Coordinate(0), 0)
    Else
                    '计算优化多段线及二维多段线的长度
        For i = 0 To CoordinateCount - 2
            TotalLength = TotalLength + _
                    GetArcLeng(Ent.Coordinate(i), Ent.Coordinate(i + 1), Ent.GetBulge(i))
        Next
                    '多段线闭合时,需要计算最后闭合段的长度
        If Ent.Closed Then TotalLength = TotalLength + _
            GetArcLeng(Ent.Coordinate(CoordinateCount - 1), _
                Ent.Coordinate(0), Ent.GetBulge(CoordinateCount - 1))
    End If
    MsgBox "选定多段线的总长度为: " & TotalLength
End Sub
```

首先在一个 Do 语句中对 GetEntity 方法进行错误处理,并且确保用户选择的是轻量多段线。TypeName 函数返回一个字符串,用于判断一个 Variant 变量的具体类型。Like 运算符用于比较两个字符串,返回的结果是 True 或者 False。

在计算多段线的顶点个数时,由于 ActiveX 模型并未提供相关的属性来获得其顶点数,只能利用 Coordinates 属性获得一个包含多段线顶点所有坐标的数组,将此数组的维数除以每个顶点的坐标维数(轻量多段线是 2,普通多段线是 3)就可得到多段线的顶点数。

(3) 创建一个函数 GetArcLength,用于根据起点、终点和凸度来计算多段线中某一段的长度,其实现代码为:

```
Private Function GetArcLeng(PointS As Variant, PointE As Variant, Bugle As Double) As Double
    Dim Angle As Double
    Dim Radius As Double
    Dim Length As Double
    Dim Dist As Double
    Dim i As Integer
    '计算起点到终点的长度
    For i = LBound(PointS) To UBound(PointS)
        Dist = Dist + ((PointS(i) - PointE(i)) ^ 2)
    Next
```

```
    Length = Sqr(Dist)
    '如果凸度为 0,则为直线段,所以起点到终点的长度就是需要的长度
    If Bugle = 0 Then
        GetArcLeng = Length
    Else
        '如果凸度不为零,则计算弧段的长度。按照凸度的定义,凸度为包角的 1/4 的正切值
        Angle = 4 * Atn(Abs(Bugle))
        '计算弧段的半径
        Radius = (Length/2)/Sin(Angle/2)
        '计算弧段的长度
        GetArcLeng = Radius * Angle
    End If
End Function
```

如果是直线段,两个顶点之间的距离就是该段的长度;如果是圆弧段,两顶点之间的距离是该圆弧的弦长,根据弦长计算出圆弧的半径,然后计算圆弧的长度。

3.9.4　实例效果

(1) 在 AutoCAD 界面中绘制一条多段线。

(2) 在 VBA 集成开发环境中,按 F5 键执行程序。

(3) 回到 AutoCAD 界面,可看到命令行中提示"选择多段线",选择图形中刚刚绘制的多段线。

(4) 完成以上操作后,程序自动计算多段线的长度,得到如图 3.25 所示的结果。



图 3.25　获得多段线长度的结果

3.9.5　小结

在 AutoCAD 中,多段线可以分成下面几个种类:

- 优化多段线(LightweightPolyline),属简单的二维多段线,是 2000 版本的新产物,它改善了多段线的数据存储方式,减少了空间的占用。它只由直线段和圆弧段组成,并未拟合或样条化。它的顶点坐标只有 X 和 Y 坐标,没有 Z 坐标,因为整条多段线属二维多段线,Z 坐标可由标高值取得。如果将优化多段线进行拟合或样条化,则

该多段线将自动转化成普通多段线(Polyline),即旧版本多段线格式。

- 多段线(Polyline),旧版本的多段线,属二维多段线,该类型多段线在未拟合或样条化前,也是由直线段和圆弧段组成,顶点坐标为三维坐标。它可直接拟合或样条化而不会改变图元类型。
- 三维多段线(3Dpolyline),与多段线类似,但该类型多段线可在三维空间中绘制。该多段线在未拟合或样条化前,只能由直线段组成,不能含有弧段,所以该三维多段线并没有 GetBulge 和 SetBulge 方法。它的顶点为三维坐标,它可拟合或样条化。

本节程序已经将三种多段线都做了考虑。因为三维多段线并没有凸度,所以在计算时将三维多段线的凸度设为 0 来计算长度,这样程序比较通用。

在对象模型中,将多段线和三维多段线合在一起,在编程上增加了难度,因为二维多段线和三维多段线的不同点在于是否具有凸度,所以只有使用 TypeName 函数来判断多段线属于哪种类型,以便进行不同的操作。

只有未经拟合或样条化的多段线才能通过顶点及凸度来计算每一段的长度。

通过线段的起点、终点及凸度值可以计算出弧段的所有属性,包括角度、弧半径、弧长、弦长等。

多段线是由多个线段组成,每一段具有自己的起点、终点、凸度、起点宽度、终点宽度,这些属性均可设置。

3.10 圆管的放样展开

3.10.1 实例说明

尽管现在很多三维软件的"钣金"功能都很强,但由于三维软件对电脑硬件的要求远比 AutoCAD 高(很多单位的电脑配置仍较低);而且让许多习惯使用 AutoCAD 的老工程师立即掌握三维软件也不是一件容易的事,所以用 AutoCAD 创建展开图仍有其实际意义。

本节的实例介绍用 VBA 创建圆管放样展开图的方法,关于创建展开图的理论依据可参阅有关机械制图的书籍。

3.10.2 编程思路

要想画出展开图,就必须知道展开图的"模样"。对于一个半径为 R,长度为 H 的等径圆管,展开后是一长度为 $2\pi R$、宽度为 H 的长方形,故用 AddLightWeightPolyline 方法很容易将其画出,如图 3.26 所示。

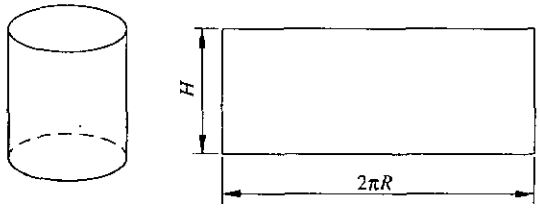


图 3.26 等径圆管展开示意图

变径管展开图是一“扇环”形，由两直线段和两圆弧段组成，如图 3.27 所示。因此，仍然可用 AddLightWeightPolyline 方法画出梯形 $ABDC$，再用 SetBulge 方法调节 AB 段和 CD 段的凸度，其关键就是求出相关的各个参数。

如图 3.28 所示，先从变径管的前视图开始着手，假设变径管下底半径 $\overline{FC}=R_1$，上底半径 $\overline{EA}=R_2$，长度 $\overline{EF}=H$。延长 CA、DB 交于 O，$\overline{AO}$、$\overline{CO}$ 就是图 3.27 中两段圆弧的半径。

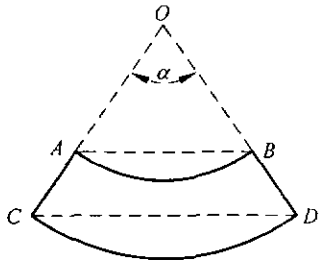


图 3.27　变径圆管展开示意图

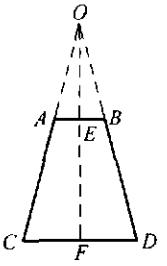


图 3.28　变径管的前视图

根据相似三角形的性质，得

$$\frac{\overline{FO}}{\overline{FO}-H}=\frac{R_2}{R_1}$$

故

$$\overline{FO}=\frac{R_1}{R_1-R_2}H$$

$$\overline{CO}=\sqrt{\overline{FO}^2+R_1}$$

$$\overline{AO}=\frac{R_2}{R_1}\overline{CO}$$

求出 $\overline{AO}$，$\overline{CO}$ 后，再回到图 3.27 中，就可求出圆心角 α：

$$\alpha=\frac{2\pi R_1}{\overline{CO}}$$

$$\overline{CD}=2\,\overline{CO}\sin(\alpha/2),\overline{AB}=2\,\overline{AO}\sin(\alpha/2)$$

求出了各个参数，写程序就是水到渠成的事了。

3.10.3　步骤

(1) 在 VBA 集成开发环境中，创建一个新工程，保存在适当的位置上。在“工程资源管理器”窗口中，双击 ThisDrawing 图标，打开该模块的代码窗口。

(2) 在 ThisDrawing 窗口中，添加几个公有变量和常量 π 的定义：

```
Dim radius1 As Double, radius2 As Double      '圆管两端的半径
Dim length As Double, tmp As Double           '圆管的长度
Dim arcLength1 As Double, arcLength2 As Double, ang As Double
'扇型环的内部和外部弧长，圆心角
Dim ptCorner As Variant, ptBase(0 To 2) As Double
'展开图的左下角点、计算顶点的起始基点
Dim ptArray(7) As Double                      '多段线各顶点坐标数组
```

```
Dim objPline As AcadLWPolyline

Const PI As Double = 3.1415926535897              '圆周率 π
```

(3) 创建一个宏 UnwrapPipe,提示用户输入变径管的各项参数,并且根据参数值创建符合要求的展开图,其实现代码为:

```
Public Sub UnwrapPipe( )
    On Error Resume Next

    ptCorner = ThisDrawing.Utility.GetPoint(, vbCrLf & "输入展开图的左下角点 <0,0>: ")
    If Err Then
        Err.Clear
        ptBase(0) = 0: ptBase(1) = 0: ptBase(2) = 0
    Else
        ptBase(0) = ptCorner(0): ptBase(1) = ptCorner(1): ptBase(2) = ptCorner(2)
    End If

RETRY:
    radius1 = ThisDrawing.Utility.GetDistance(ptBase, vbCrLf & "输入管一头的半径: ")
    radius2 = ThisDrawing.Utility.GetDistance(ptBase, vbCrLf & "请输入管另一头的半径: ")
    length = ThisDrawing.Utility.GetDistance(ptBase, vbCrLf & "输入管的长度: ")

    If radius1 <= 0 Or radius2 <= 0 Or length <= 0 Then        '错误处理
        MsgBox ("半径和长度必须为正,请重新输入!")
        GoTo RETRY
    End If

    If radius1 = radius2 Then
        Call EqualRadiusPipe
    Else
        Call AlterRadiusPipe
    End If
End Sub
```

其中调用了两个自定义过程 EqualRadiusPipe 和 AlterRadiusPipe,分别用于创建变径管和等径管的展开图。

(4) 自定义过程 EqualRadiusPipe 用于创建等径管的展开图,其实现代码为:

```
Private Sub EqualRadiusPipe( )
    '计算各点坐标
    ptArray(0) = ptBase(0)
    ptArray(1) = ptBase(1)
    ptArray(2) = ptBase(0) + 2 * PI * radius1
    ptArray(3) = ptArray(1)
    ptArray(4) = ptArray(2)
    ptArray(5) = ptArray(1) + length
```

```
    ptArray(6) = ptArray(0)
    ptArray(7) = ptArray(5)

    '画出展开图
    Set objPline = ThisDrawing.ModelSpace.AddLightWeightPolyline(ptArray)
    objPline.Closed = True
    objPline.Update
End Sub
```

(5) 自定义过程 AlterRadiusPipe 用于创建变径管的展开图，其实现代码为：

```
Private Sub AlterRadiusPipe( )
    'radius1 为粗头半径
    If radius1 < radius2 Then
        tmp = radius1
        radius1 = radius2
        radius2 = tmp
    End If

    '计算各点的坐标
    arcLength1 = radius1 * Sqr((length/(radius1 - radius2)) ^ 2 + 1)
    '扇型环的内部弧长
    arcLength2 = (radius2/radius1) * arcLength1
    '外部弧长
    ang = 2 * PI * radius1/arcLength1
    '圆心角等于弧长除以半径

    ptArray(0) = ptBase(0)
    ptArray(1) = ptBase(1)
    ptArray(2) = ptArray(0) + 2 * arcLength1 * Sin(ang/2)
    ptArray(3) = ptArray(1)
    ptArray(6) = ptArray(0) + (arcLength1 - arcLength2) * Sin(ang/2)
    ptArray(7) = ptArray(1) + (arcLength1 - arcLength2) * Cos(ang/2)
    ptArray(4) = ptArray(6) + 2 * arcLength2 * Sin(ang/2)
    ptArray(5) = ptArray(7)

    '画梯形
    Set objPline = ThisDrawing.ModelSpace.AddLightWeightPolyline(ptArray)
    objPline.Closed = True

    '设置梯形上下底的凸度，完成展开图
    objPline.SetBulge 0, Tan(ang/4)
    objPline.SetBulge 2, - Tan(ang/4)
    objPline.Update
    ZoomExtents
End Sub
```

3.10.4 实例效果

在 VBA 集成开发环境中，执行 UnwrapPipe 宏，按照命令提示进行操作。

命令：
输入展开图的左下角点 <0,0>：　【任意拾取一点】
命令：
输入管一头的半径：30

命令：
请输入管另一头的半径：70

命令：
输入管的长度：50

完成操作后，得到如图 3.29 所示的结果。

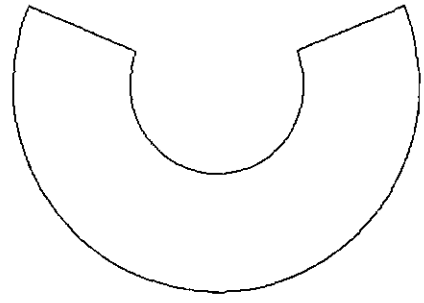


图 3.29　创建变径管展开的结果

3.10.5 小结

画展开图，首先要知道展开图是什么形状，只有知道是什么形状，才能正确地编写程序。在写展开图程序时，如不能正确地求出各个关键参数，就写不出正确的程序，这就要求编程者一定要有数学基础和空间想象能力。

通过这个程序，应掌握以下几方面的内容。

- 放样展开的正确思路。
- 用 AddLightWeightPolyline 方法绘制多段线。
- 用 SetBulge 方法设置多段线的凸度。

3.11 天圆地方的放样展开

3.11.1 实例说明

“天圆地方”是一种特殊形状的实体，其顶面是一个圆，底面却是一个正方形，如图 3.30 所示。要创建该实体的放样展开图不是一个简单的任务，本节的实例采用多点模拟的方法创建了天圆地方实体的展开图。

3.11.2 编程思路

天圆地方（方圆过度管）由前、后、左、右四个相同的等腰三角形和四个相同的斜锥面组成。如果沿 GI 展开，展开图形如图 3.31 所示。

在图 3.30 中，将“天圆”进行 16 等分，连接等分点和相应的“地方”顶点（机械制图中称为素线，图中只画出其中的一部分），展开后，这些素线的长度不变。同时，各相邻的等分点之间的距离也同样不变。这就是画展开图的一般性原理。

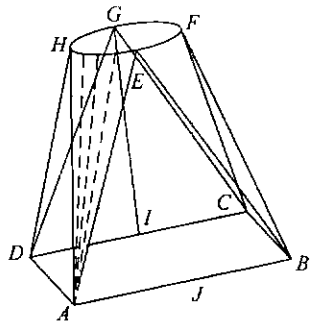


图 3.30　天圆地方的形状

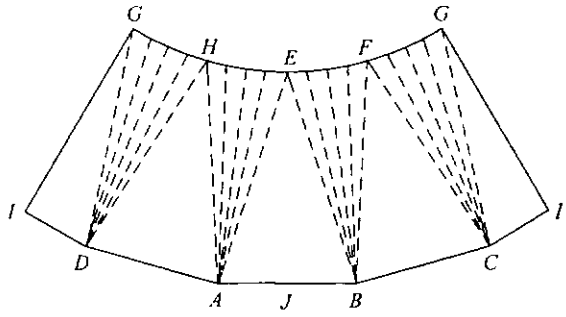


图 3.31　天圆地方的展开图

值得注意的是，展开图的上部并非圆弧。必须通过“描点”的方法来绘制这段曲线。为了获得较高的精度，本节程序进行的是 360 等分而不是 16 等分。程序的关键就是求出展开图中各点的坐标。

3.11.3　步骤

在 AutoCAD 2005 中，打开 VBA 管理器，创建一个新工程，保存在适当的位置。进入 VBA 集成开发环境，双击“工程资源管理器”窗口中的 ThisDrawing 图标，打开代码窗口，在其中添加绘制天圆地方展开图的宏 Main：

```
Public Sub Main( )
    Const PI As Double = 3.1415926535

    On Error Resume Next
    '得到图 3.31 中的 J 点坐标
    Dim pt0 As Variant, ptBase(2) As Double
    pt0 = ThisDrawing.Utility.GetPoint(, vbCrLf & "请输入 "天圆地方 "展开图下边中点 <0,0>: ")

    If Err Then
        Err.Clear
        ptBase(0) = 0: ptBase(1) = 0
    Else
        ptBase(0) = pt0(0): ptBase(1) = pt0(1)
    End If

    '获得天圆地方实体的半径、高度和底面边长
    Dim radius As Double, height As Double, length As Double
RETRY:
    radius = ThisDrawing.Utility.GetDistance(ptBase, vbCrLf & "请输入"天圆"的半径: ")
    height = ThisDrawing.Utility.GetDistance(ptBase, vbCrLf & "请输入"天圆地方"的高度: ")
    length = ThisDrawing.Utility.GetDistance(ptBase, vbCrLf & "请输入"地方"的边长: ")

    If radius <= 0 Or height <= 0 Or length <= 0 Then
```

```
    MsgBox ("输入数据必须为正,请重新输入!")
    GoTo RETRY
End If

'先画展开图中的"曲线"

'得到图 3.31 中的 A 点
Dim pt1 As Variant, pt2 As Variant
pt1 = ThisDrawing.Utility.PolarPoint(ptBase, 0, -0.5 * length)

'得到图 3.31 中的 B 点
pt2 = ThisDrawing.Utility.PolarPoint(ptBase, 0, 0.5 * length)

'得到图 3.31 中的 AE、BE 的长度
Dim dist0 As Double
dist0 = Sqr(0.25 * length ^ 2 + (0.5 * length - radius) ^ 2 + height ^ 2)

'得到图 3.31 中的角 EAJ,为 EH 段各等分点坐标的计算做准备
Dim ang1, ang2 As Double
ang1 = Atn((Sqr(height ^ 2 + (0.5 * length - radius) ^ 2)/(0.5 * length)))

'角 EAJ 的补角,为 EF 段各等分点坐标的计算做准备
ang2 = PI - ang1

Dim dist(90) As Double, i As Integer, tmp As Double
Dim angle1(90) As Double, angle2(90) As Double
For i = 0 To 90
    If i = 0 Then     '初值
        dist(i) = dist0
        angle1(i) = ang1
        angle2(i) = ang2
    Else
        '计算与 A 点与 EH 段各等分点、B 点与 EF 段各等分点的距离
        dist(i) = Sqr((height ^ 2 + (0.5 * length - radius * Sin(i * PI/180)) ^ 2) _
                    + (0.5 * length - radius * Cos(i * PI/180)) ^ 2)

        '计算与 A 点与 EH 段各等分点连线和 X 轴正向的夹角
        tmp = (dist(i) ^ 2 + dist(i - 1) ^ 2 - (radius * PI/180) ^ 2)/(2 * dist(i) * dist(i - 1))
            angle1(i) = angle1(i - 1) + Atn( - tmp/Sqr( - tmp * tmp + 1)) + 2 * Atn(1)

        '计算与 B 点与 EF 段各等分点连线和 X 轴正向的夹角
        angle2(i) = angle2(i - 1) - Atn( - tmp/Sqr( - tmp * tmp + 1)) - 2 * Atn(1)
    End If
Next
```

```
'计算组成展开图的曲线部分的各点的坐标
Dim point1(721) As Double
For i = 0 To 2 * 360 + 1 Step 2
    If i < 180 Then
        '计算 EH 段各等分点的坐标
        point1(i + 180) = pt1(0) + dist(90 - i/2) * Cos(angle1(90 - i/2))
        point1(i + 181) = pt1(1) + dist(90 - i/2) * Sin(angle1(90 - i/2))

    ElseIf i < 360 Then
        '计算 EF 段各等分点的坐标
        point1(i + 180) = pt2(0) + dist(i/2 - 90) * Cos(angle2(i/2 - 90))
        point1(i + 181) = pt2(1) + dist(i/2 - 90) * Sin(angle2(i/2 - 90))

    ElseIf i <= 540 Then
        '计算 FG 段各等分点的坐标
        tmp = (dist(90) ^ 2 + 0.25 * length ^ 2 - height ^ 2 - (0.5 * length - radius) ^ 2)/
            (dist(90) * length)
        Dim ang3 As Double
        ang3 = angle2(90) - Atn( - tmp/Sqr( - tmp * tmp + 1)) - 2 * Atn(1)

        Dim pt3(2) As Double
        pt3(0) = pt2(0) + length * Cos(ang3)
        pt3(1) = pt2(1) + length * Sin(ang3)

        point1(i + 180) = pt3(0) + dist(i/2 - 180) * Cos(angle2(i/2 - 180) + ang3)
        point1(i + 181) = pt3(1) + dist(i/2 - 180) * Sin(angle2(i/2 - 180) + ang3)

    Else
        '计算 HG 段各等分点的坐标
        Dim ang4 As Double
        ang4 = angle1(90) + Atn( - tmp/Sqr( - tmp * tmp + 1)) + 2 * Atn(1)

        Dim pt4(2) As Double
        pt4(0) = pt1(0) + length * Cos(ang4)
        pt4(1) = pt1(1) + length * Sin(ang4)

        point1(0) = pt4(0) + dist(0) * Cos(angle1(90) + ang4 - PI)
        point1(1) = pt4(1) + dist(0) * Sin(angle1(90) + ang4 - PI)

        point1 (i - 540) = pt4(0) + dist(360 - i/2) * Cos(angle1(360 - i/2) + ang4 - PI)
        point1 (i - 539) = pt4(1) + dist(360 - i/2) * Sin(angle1(360 - i/2) + ang4 - PI)
    End If
Next

Dim objPoly1 As AcadLWPolyline
Set objPoly1 = ThisDrawing.ModelSpace.AddLightWeightPolyline(point1)
```

```
    '再画展开图中的"折线"
    Dim point2(15) As Double
    point2(0) = point1(0)
    point2(1) = point1(1)

    Dim ang5 As Double
    ang5 = 2 * ang4 - PI
    point2(2) = pt4(0) + 0.5 * length * Cos(ang5)
    point2(3) = pt4(1) + 0.5 * length * Sin(ang5)

    point2(4) = pt4(0)
    point2(5) = pt4(1)
    point2(6) = pt1(0)
    point2(7) = pt1(1)
    point2(8) = pt2(0)
    point2(9) = pt2(1)
    point2(10) = pt3(0)
    point2(11) = pt3(1)

    Dim ang6 As Double
    ang6 = 2 * ang3
    point2(12) = pt3(0) + 0.5 * length * Cos(ang6)
    point2(13) = pt3(1) + 0.5 * length * Sin(ang6)

    point2(14) = point1(720)
    point2(15) = point1(721)

    Dim objPoly2 As AcadLWPolyline
    Set objPoly2 = ThisDrawing.ModelSpace.AddLightWeightPolyline(point2)
    ZoomExtents
End Sub
```

对于曲线段的绘制，首先提示用户输入 J 点的位置，然后根据 J 点的位置使用 PolarPoint 方法获得 A、B 两点的坐标，根据这两点坐标分别计算出 EH、EF 段曲线上的点的坐标。然后根据 A、B 两点的位置计算出 D、C 两点的坐标，并依次计算 HG、FG 段曲线上点的坐标。

比起曲线段的绘制，折线段的绘制就简单一些，在前面已经计算出 A、B、C、D 四点的基础上，再获得被剪开的 G、I 点在两端的坐标即可。

3.11.4 实例效果

在 VBA 集成开发环境中，执行 Main 宏，按照命令提示输入图形底面的中心，并且输入 30 作为顶面半径，输入 60 作为实体的高度，输入 60 作为底面的半径，能够得到如图 3.32 所示的结果。

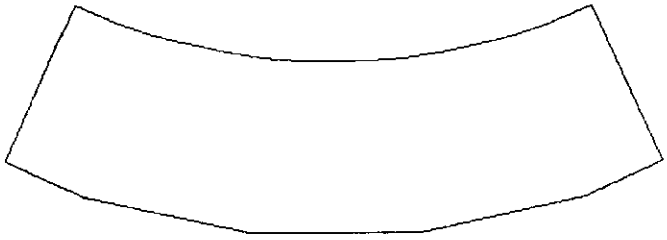


图 3.32　获得的天圆地方展开图

3.11.5　小结

很多展开图，其部分轮廓不是 AutoCAD 的基本图元，只有通过“描点”的方法，才能绘制出。所以在写展开图程序时，求出这些点的坐标就成为关键。以上程序主要的依据就是展开前后“母线相等”这个原则，通过这个程序，应掌握以下几方面的内容。

- 放样展开的正确思路。
- 用描点方法绘制多段线。

第4章 对象特性

在AutoCAD二次开发中，仅仅使用对象特性的程序实在不多，对象特性更多的时候是一种辅助工具，按图层进行缩放的程序就是这样。ObjectDBX能够在不打开图形的情况下访问图形中的多种信息，本章的实例使用ObjectDBX技术导入其他图形中的信息，这实际上提供了一种后期"绑定"样板文件的手段。

4.1 按图层进行缩放

4.1.1 实例说明

在《AutoCAD VBA开发精彩实例教程》中，曾经编写了根据对象进行缩放的程序。本节所介绍的程序能够提示用户输入要全部显示的图层，将该图层中所有的实体显示在图形窗口中，其效果类似于对某个图层进行范围(Extents)缩放。

此外，与根据对象进行缩放的程序对比，本节所介绍的程序结构更为简单，通用性更强。

4.1.2 编程思路

无论要在图形窗口中完全显示多少个实体，其原理都一样，就是计算符合条件的所有实体的最小包围框，将包围框的两个角点设置为窗口缩放的角点。

获得图形中某个图层上所有的实体并不困难，对模型空间所有的实体进行遍历，并且判断其所在的图层即可。问题的关键就是获得多个实体的最小包围框。

获得一个实体的最小包围框可以使用GetBoundingBox方法，可以将所有实体的最小包围框角点写入到一个点数组中，在点数组中计算出左下角和右上角的角点。

4.1.3 步骤

(1) 在AutoCAD 2005中，打开VBA管理器，创建一个新工程，保存在适当的位置。进入VBA集成开发环境，双击"工程资源管理器"窗口中的ThisDrawing图标，打开代码窗口，添加一个函数GetLeftBottomPt，根据一组对象的左下角点集合获得包围框的左下角点，其实现代码为：

```
Private Function GetLeftBottomPt(ByRef ptArr( ) As Variant) As Variant
    Dim ptLeftBottom(0 To 2) As Double     '左下角点

    Dim i As Long
    For i = 0 To UBound(ptArr)
    If i = 0 Then
            ptLeftBottom(0) = ptArr(i)(0)
```

```
            ptLeftBottom(1) = ptArr(i)(1)
        End If

        '确保 ptLeftBottom 的 X、Y 坐标值均为最小
        If ptArr(i)(0) < ptLeftBottom(0) Then ptLeftBottom(0) = ptArr(i)(0)
        If ptArr(i)(1) < ptLeftBottom(1) Then ptLeftBottom(1) = ptArr(i)(1)
    Next i

    ptLeftBottom(2) = 0

    GetLeftBottomPt = ptLeftBottom
End Function
```

其基本原理就是逐个比较数组中的各个元素，不断获得其中较小的坐标。

(2) 添加一个函数 GetRightTopPt，根据一组对象的左下角点集合获得包围框的右上角点，其实现代码为：

```
Private Function GetRightTopPt(ByRef ptArr( ) As Variant) As Variant
    Dim ptRightTop(0 To 2) As Double        '右上角点

    Dim i As Long
    For i = 0 To UBound(ptArr)
        If i = 0 Then
            ptRightTop(0) = ptArr(i)(0)
            ptRightTop(1) = ptArr(i)(1)
        End If

        '确保 ptLeftBottom 的 X、Y 坐标值均为最大
        If ptArr(i)(0) > ptRightTop(0) Then ptRightTop(0) = ptArr(i)(0)
        If ptArr(i)(1) > ptRightTop(1) Then ptRightTop(1) = ptArr(i)(1)
    Next i

    ptRightTop(2) = 0

    GetRightTopPt = ptRightTop
End Function
```

(3) 添加一个过程 LayerZoom，用于根据指定名称的图层的所有实体进行缩放。

```
Private Sub LayerZoom(ByVal strLayer As String)
    Dim ptArr1( ) As Variant      '所有对象的左下角点数组
    Dim ptArr2( ) As Variant      '所有对象的右上角点数组

    Dim ent As AcadEntity
    Dim i As Long
    Dim count As Long             '当前点数组的维数
    count = - 1                   '注意一定要从 - 1 开始，保证第一个元素索引为 0
```

```
    For i = 0 To ThisDrawing.ModelSpace.count - 1
        Set ent = ThisDrawing.ModelSpace.Item(i)
        If StrComp(ent.Layer, strLayer, vbTextCompare) = 0 Then
            count = count + 1
            ReDim Preserve ptArr1(count)
            ReDim Preserve ptArr2(count)
            ent.GetBoundingBox ptArr1(count), ptArr2(count)
        End If
    Next i

    '获得图层中所有实体的包围框角点
    Dim ptLeftBottom As Variant, ptRightTop As Variant
    ptLeftBottom = GetLeftBottomPt(ptArr1)
    ptRightTop = GetRightTopPt(ptArr2)

    '缩放的操作
    ZoomWindow ptLeftBottom, ptRightTop
End Sub
```

(4) 添加 ZoomToLayer 宏,用于提示用户输入图层名称,并且根据该图层上所有的实体进行缩放,其相关代码为:

```
Public Sub ZoomToLayer( )
    Dim strLayer As String
    strLayer = ThisDrawing.Utility.GetString(True, "输入图层名称: ")

    If HasLayer(strLayer) Then
        Call LayerZoom(strLayer)
    Else
        ThisDrawing.Utility.Prompt "不存在指定名称的图层……" & vbCrLf
    End If

    End
End Sub
```

其中,HasLayer 是一个自定义函数,用于判断当前图形中是否包含指定名称的图层,其实现代码为:

```
Private Function HasLayer(ByVal strLayer As String) As Boolean
    HasLayer = False

    Dim objLayer As AcadLayer
    For Each objLayer In ThisDrawing.Layers
        If StrComp(objLayer.Name, strLayer, vbBinaryCompare) = 0 Then
            HasLayer = True
            Exit Function
        End If
```

```
    Next objLayer
End Function
```

4.1.4 实例效果

在 VBA 集成开发环境中，执行 ZoomToLayer 宏，AutoCAD 会提示“输入图层名称：”，输入要缩放的图层名称，系统会在图形窗口中完全显示该图层上的所有实体。如果输入的图层名称不存在，系统会提示“不存在指定名称的图层……”。

4.1.5 小结

GetBoundingBox 方法很容易被人忽略，但是它实际上却是很有用的，例如要创建一个带有圆圈的文字，并且保证文字正好位于圆的内部，就必须使用 GetBoundingBox 方法获得文字的包围框，然后根据包围框来创建圆，如图 4.1 所示。

下面的宏能够创建一个文字的外接圆。



图 4.1　带有圆圈的文字

```
Public Sub CreateCircleToText( )
    Dim objText As AcadText
    Dim ptPick As Variant
    ThisDrawing.Utility.GetEntity objText, ptPick, "拾取文字："

    '获得文字的包围框
    Dim ptMin As Variant, ptMax As Variant
    objText.GetBoundingBox ptMin, ptMax

    '获得圆心和半径
    Dim ptCenter(0 To 2) As Double
    ptCenter(0) = (ptMin(0) + ptMax(0))/2
    ptCenter(1) = (ptMin(1) + ptMax(1))/2
    ptCenter(2) = 0
    Dim radius As Double
    radius = Sqr((ptMin(0) - ptMax(0)) ^ 2 + (ptMin(1) - ptMax(1)) ^ 2)/2

    '创建圆
    Dim objCircle As AcadCircle
    Set objCircle = ThisDrawing.ModelSpace.AddCircle(ptCenter, radius)
End Sub
```

4.2 使用 ObjectDBX 技术导入其他图形中的信息

4.2.1 实例说明

很多时候，用户在创建图形时都必须从一个样板图形开始，这样的目的是为了保证整个小组工作的协调，使用同样的图层配置、文字样式和标注样式等。如果用户在创建图形时忘

记从一个标准的样板开始，那么手工修改这些图形信息将会很麻烦。

本节介绍的实例允许用户将其他图形中存在的图层、文字样式和标注样式等信息复制到当前图形中，使当前图形的配置与指定的图形保持一致，其运行界面如图 4.2 所示。

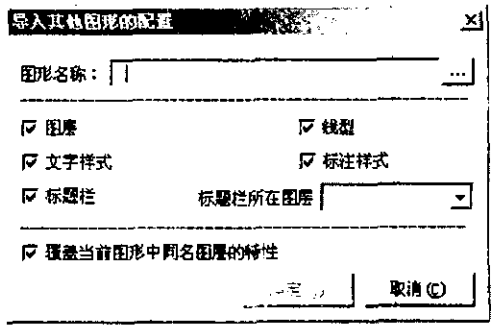


图 4.2 程序运行界面

4.2.2 编程思路

在《AutoCAD VBA 开发精彩实例教程》中，已经利用 ObjectDBX 插入未打开的图形文件中的块，那在 ObjectDBX 的编程中已经是比较复杂的方法，本节要使用的技术相对简单一些，直接使用 ObjectDBX 获得外部图形的特性，并使用 CopyObjects 方法将一些对象复制到当前图形。

引用 ObjectDBX 的类型库（参见步骤中的操作）之后，就可以使用对象浏览器（在 VBA 集成开发环境中，按 F2 键即可打开）来查看它所包含的对象、属性和方法，如图 4.3 所示。

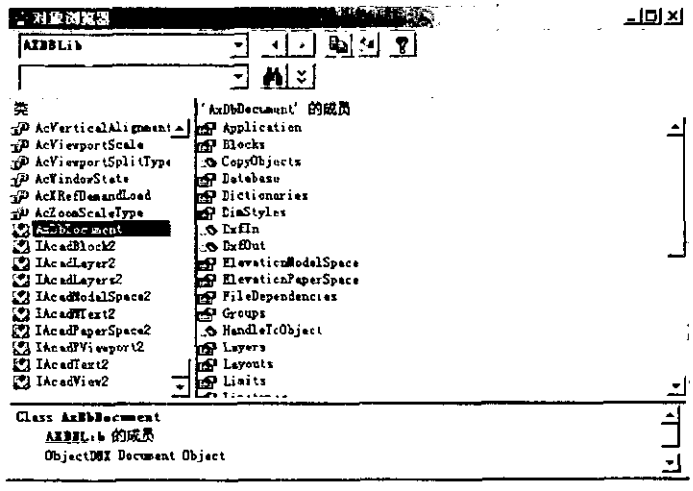


图 4.3 查看 ObjectDBX 类型库的内容

AxDbDocument 对象是 ObjectDBX 中一个最重要的对象，访问它的属性可以获得外部图形中的很多信息，例如 DimStyles 属性返回图形中所有标注样式的集合，Layers 属性返回图形中所有图层的集合，ModelSpace 返回图形的模型空间所有实体的集合，这与 AutoCAD 的 ActiveX 模型都是一致的。

要将外部图形中的一个实体复制到当前图形中，可以使用下面的代码：

```
objDBX.CopyObjects ent, ThisDrawing.ModelSpace
```

4.2.3　步骤

(1) 在 AutoCAD 2005 中，打开 VBA 管理器，创建一个新工程，保存在适当的位置。进入 VBA 集成开发环境，选择“工具”|“引用”菜单项，在弹出的对话框中引用标准对话框(Microsoft Common Dialog Control 6.0)和 ObjectDBX(AutoCAD/ObjectDBX Common 16.0 Type Library)的组件，如图 4.4 所示。

(2) 选择“插入”|“用户窗体”菜单项，向工程中添加一个对话框，将其名称修改为 frmMain，并在其中添加如图 4.5 所示的控件。其中，文本框的名称修改为 txtFileName，浏览按钮的名称为 cmdBrowser，六个复选框的名称分别为 chkLayer、chkLinetype、chkTextStyle、chkDimStyle、chkTitle 和 chkOverWrite，组合框的名称为 cboLayers，两个命令按钮的名称为 cmdOk 和 cmdCancel，标准对话框的名称为 dlgCommon。

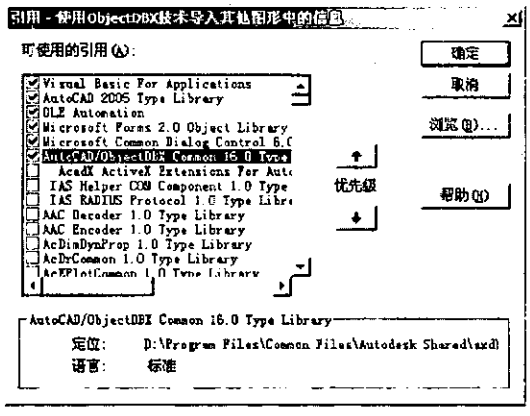


图 4.4　引用标准对话框和 ObjectDBX 类型库

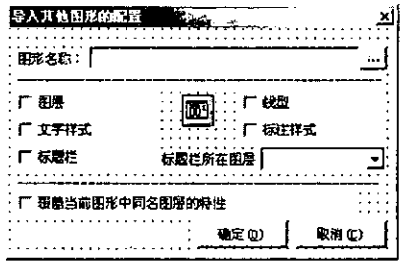


图 4.5　设计窗体

此外，左下角有一个标签控件，其名称为 lblErrMsg，其 Caption 属性被设置为空字符串。在对话框中使用了水平分隔线，实际上就是一个 Caption 属性为“----------”的标签控件。在属性窗口中，将 cboLayers 的 Style 属性设置为 2-fmStyleDropDownList。

(3) 在“工程资源管理器”窗口中双击 frmMain 图标，然后按 F7 键，打开窗体的代码窗口，在通用区添加一个模块级的变量：

```
Dim objDBX As Object      'ObjectDBX 对象
```

(4) 在窗体初始化的事件中，创建 ObjectDBX 对象，并且设置窗体中控件的初始状态，其相关代码为：

```
Private Sub UserForm_Initialize( )
    '根据 AutoCAD 的版本来确定使用 ObjectDBX 的版本
    If Left(Version, 2) = "15" Then
        Set objDBX = CreateObject("ObjectDBX.AxDbDocument.1")
```

```
    ElseIf Left(Version, 2) = "16" Then
        Set objDBX = CreateObject("ObjectDBX.AxDbDocument.16")
    End If

    '设置控件的状态
    cmdOk.Enabled = False
    chkLayer.Value = True
    chkLinetype.Value = True
    chkTextStyle.Value = True
    chkDimStyle.Value = True
    chkTitle.Value = True
    chkOverWrite.Value = True
End Sub
```

需要注意的是，AutoCAD 2000 和 AutoCAD 2002 对应的 ObjectDBX 进程 ID 是 ObjectDBX.AxDbDocument.1。这是有一点历史原因的，Autodesk 在 AutoCAD 2000/2002 版本中开始创建 ObjectDBX，本想从 R1 开始对其进行编号(就像当时出现过 ObjectARX 2.0)，但是后来又想让 ObjectDBX 的版本号和 AutoCAD 核心版本号保持一致，因此在 AutoCAD 2004 中就成了 ObjectDBX.AxDbDocument.16。

(5) 在浏览按钮的单击事件中，系统弹出“选择图形文件”对话框，用户可以选择所要导入配置的外部图形文件，其相关代码为：

```
Private Sub cmdBrowser_Click( )
    '设置标准对话框
    With dlgCommon
        .DialogTitle = "选择图形文件"
        .Filter = "图形文件(*.dwg)|*.dwg|所有文件(*.*)|*.*"
        .InitDir = ThisDrawing.Application.Path
        .ShowOpen
    End With

    '文件类型错误的判断
    If StrComp(Right(dlgCommon.FileName, 3), "dwg", vbTextCompare) <> 0 Then
        MsgBox "不支持的文件类型!", vbCritical, "警告"
        Exit Sub
    End If

    '在文本框中显示文件的名称
    txtFileName.Text = dlgCommon.FileName
End Sub
```

上面的代码中，使用了标准对话框来实现图形文件的选择，该控件的使用在《AutoCAD VBA 开发精彩实例教程》一书中已经详细介绍，这里不再多说。

(6) 当用户修改“图形名称”文本框中的图形文件时，系统会自动判断输入的文件名是否有效，并对窗体中控件的状态进行修改。其相关代码为：

```
Private Sub txtFileName_Change( )
    '根据选择文件的有效性,设置控件的状态
    If Len(Dir(txtFileName.Text)) <> 0 Then
        cmdOk.Enabled = True

        获得选择的图形的图层列表
        objDBX.Open txtFileName.Text

        Dim objLayer As AcadLayer
        For Each objLayer In objDBX.Layers
            cboLayers.AddItem objLayer.Name
        Next objLayer

        cboLayers.ListIndex = 0
    Else
        lblErrMsg.ForeColor = vbRed
        lblErrMsg.Caption = "所指定的图形文件不存在…"

        '修改控件状态
        cmdOk.Enabled = False
        cboLayers.Clear
    End If
End Sub
```

(7) 在窗体的代码窗口中添加一个过程 ImportLayers,用于导入外部图形中的图层信息,其相关代码为:

```
Private Sub ImportLayers(ByVal bOverWrite As Boolean)
    Dim objLayer As AcadLayer
    For Each objLayer In objDBX.Layers
        If Not HasLayer(objLayer.Name) Then
            ThisDrawing.Layers.Add objLayer.Name
            Call CopyLayerSetting(ThisDrawing.Layers(objLayer.Name), objLayer)
        Else
            If bOverWrite Then
                Call CopyLayerSetting(ThisDrawing.Layers(objLayer.Name), objLayer)
            End If
        End If
    Next objLayer
End Sub
```

对于外部图形中的某个图层,如果当前图形中已经存在与其同名的图层,用户可以设置是否覆盖当前图形中同名图层的配置,这是由参数 bOverWrite 来控制的。HasLayer 和 CopyLayerSetting 是两个自定义函数,其中,HasLayer 函数用于判断当前图形中是否存在指定名称的图层,其实现代码为:

```
Private Function HasLayer(ByVal strLayer As String) As Boolean
```

```
    HasLayer = False

    Dim objLayer As AcadLayer
    For Each objLayer In ThisDrawing.Layers
        If StrComp(objLayer.Name, strLayer) = 0 Then
            HasLayer = True
            Exit Function
        End If
    Next objLayer
End Function
```

CopyLayerSetting 函数用于在两个图层之间复制属性，其实现代码为：

```
Private Sub CopyLayerSetting(ByVal objLayer1 As AcadLayer, ByVal objLayer2 As AcadLayer)
    objLayer1.linetype = objLayer2.linetype
    objLayer1.LayerOn = objLayer2.LayerOn
    objLayer1.Lineweight = objLayer2.Lineweight
    objLayer1.Lock = objLayer2.Lock
    objLayer1.TrueColor = objLayer2.TrueColor
    objLayer1.ViewportDefault = objLayer2.ViewportDefault
End Sub
```

(8) 添加一个过程 ImportLinetypes，用于导入外部图形中的线型，其实现代码为：

```
Private Sub ImportLinetypes( )
    Dim linetype(0) As AcadLineType
    Dim objLinetype As AcadLineType
    For Each objLinetype In objDBX.Linetypes
        If Not HasLinetype(objLinetype.Name) Then
            Set linetype(0) = objLinetype
            objDBX.CopyObjects linetype, ThisDrawing.ModelSpace
        End If
    Next objLinetype
End Sub
```

如果当前图形中已经存在某个线型，则不进行导入操作。HasLinetype 函数是一个自定义函数，用于判断当前图形中是否包含指定名称的线型，其实现代码为：

```
Private Function HasLinetype(ByVal strLinetype As String) As Boolean
    HasLinetype = False

    Dim objLinetype As AcadLineType
    For Each objLinetype In ThisDrawing.Linetypes
        If StrComp(objLinetype.Name, strLinetype) = 0 Then
            HasLinetype = True
            Exit Function
        End If
    Next objLinetype
```

```
End Function
```

(9) 添加一个过程 ImportTextStyles，用于导入外部图形中的文字样式，其实现代码为：

```
Private Sub ImportTextStyles( )
    Dim textStyle(0) As AcadTextStyle
    Dim objTextStyle As AcadTextStyle
    For Each objTextStyle In objDBX.TextStyles
        If Not HasTextStyle(objTextStyle.Name) Then
            Set textStyle(0) = objTextStyle
            objDBX.CopyObjects textStyle, ThisDrawing.ModelSpace
        End If
    Next objTextStyle
End Sub
```

如果当前图形中已经存在某个文字样式，则不进行导入操作。HasTextStyle 函数是一个自定义函数，用于判断当前图形中是否包含指定名称的文字样式，其实现代码为：

```
Private Function HasTextStyle(ByVal strTextStyle As String) As Boolean
    HasTextStyle = False

    Dim objTextStyle As AcadTextStyle
    For Each objTextStyle In ThisDrawing.TextStyles
        If StrComp(objTextStyle.Name, strTextStyle) = 0 Then
            HasTextStyle = True
            Exit Function
        End If
    Next objTextStyle
End Function
```

(10) 添加一个过程 ImportDimStyles，用于导入外部图形中的标注样式，其实现代码为：

```
Private Sub ImportDimStyles( )
    Dim dimStyle(0) As AcadDimStyle
    Dim objDimStyle As AcadDimStyle
    For Each objDimStyle In objDBX.DimStyles
        If Not HasDimStyle(objDimStyle.Name) Then
            Set dimStyle(0) = objDimStyle
            objDBX.CopyObjects dimStyle, ThisDrawing.ModelSpace
        End If
    Next objDimStyle
End Sub
```

如果当前图形中已经存在某个标注样式，则不进行导入操作。HasDimStyle 函数是一个自定义函数，用于判断当前图形中是否包含指定名称的标注样式，其实现代码为：

```
Private Function HasDimStyle(ByVal strDimStyle As String) As Boolean
```

```
    HasDimStyle = False

    Dim objDimStyle As AcadDimStyle
    For Each objDimStyle In ThisDrawing.DimStyles
        If StrComp(objDimStyle.Name, strDimStyle) = 0 Then
            HasDimStyle = True
            Exit Function
        End If
    Next objDimStyle
End Function
```

(11) 添加一个过程 ImportEnts,用于导入外部图形中某个图层上的实体,其实现代码为:

```
Private Sub ImportEnts(ByVal strLayer As String)
    Dim objEnt(0) As AcadEntity
    Dim ent As AcadEntity
    For Each ent In objDBX.ModelSpace
        If StrComp(ent.Layer, strLayer) = 0 Then
            Set objEnt(0) = ent
            objDBX.CopyObjects objEnt, ThisDrawing.ModelSpace
        End If
    Next ent
End Sub
```

(12) 在窗体中"确定"按钮的单击事件中,调用前面定义的几个过程,根据用户的要求进行各种图形信息的导入,其实现代码为:

```
Private Sub cmdOk_Click( )
    If chkLinetype.Value = True Then
        Call ImportLinetypes(chkOverWrite.Value)
    End If

    '如果导入图层,必须首先导入线型
    If chkLayer.Value = True Then
        If chkLinetype.Value = False Then Call ImportLinetypes(chkOverWrite.Value)
        Call ImportLayers(chkOverWrite.Value)
    End If

    If chkTextStyle.Value = True Then
        Call ImportTextStyles(chkOverWrite.Value)
    End If

    If chkDimStyle.Value = True Then
        Call ImportDimStyles(chkOverWrite.Value)
    End If
```

```
    If chkTitle.Value = True Then
        Call ImportEnts(cboLayers.Text)
    End If

    End
End Sub
```

(13) 在“取消”按钮的单击事件中，使用 End 函数来结束程序的执行，其实现代码为：

```
Private Sub cmdCancel_Click( )
    End
End Sub
```

(14) 在“工程资源管理器”窗口中双击 ThisDrawing 图标，打开其代码窗口，在其中添加启动应用程序的宏：

```
Public Sub Main( )
    frmMain.Show
End Sub
```

4.2.4　实例效果

(1) 在 AutoCAD 2005 中，可以手工创建一个新图形，增加图层、线型、图框等对象，也可以打开配套光盘中第 4 章文件夹中的“ObjectDBX 测试图形.dwg”，查看该图形中包含了哪些图层、线型和图框等对象。查看图形之后，关闭它。

(2) 在 AutoCAD 2005 中创建一个新图形，作为导入图形信息的目标图形。进入 VBA 集成开发环境，执行 Main 宏，系统会弹出如图 4.6 所示的对话框。

(3) 单击“…”按钮，从系统弹出的“选择图形文件”对话框中，选择“ObjectDBX 测试图形.dwg”文件，单击“打开”按钮，关闭“选择图形文件”对话框，此时组合框中已经读取了该图形的图层列表，如图 4.7 所示。

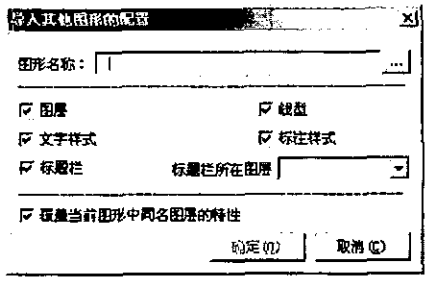


图 4.6　程序运行界面

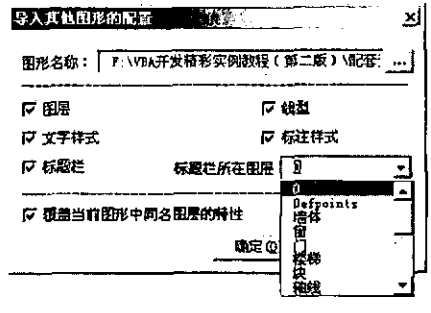


图 4.7　获得图层列表

(4) 选中对话框中所有的复选框，从组合框中选择 0(图框所在的层)，单击“确定”按钮，完成图形信息的导入。在 AutoCAD 界面中选择“格式”|“图层”菜单项，系统会弹出如图 4.8 所示的“图层特性管理器”对话框，其中已经包含了“ObjectDBX 测试图形.dwg”图形中的所有图层。

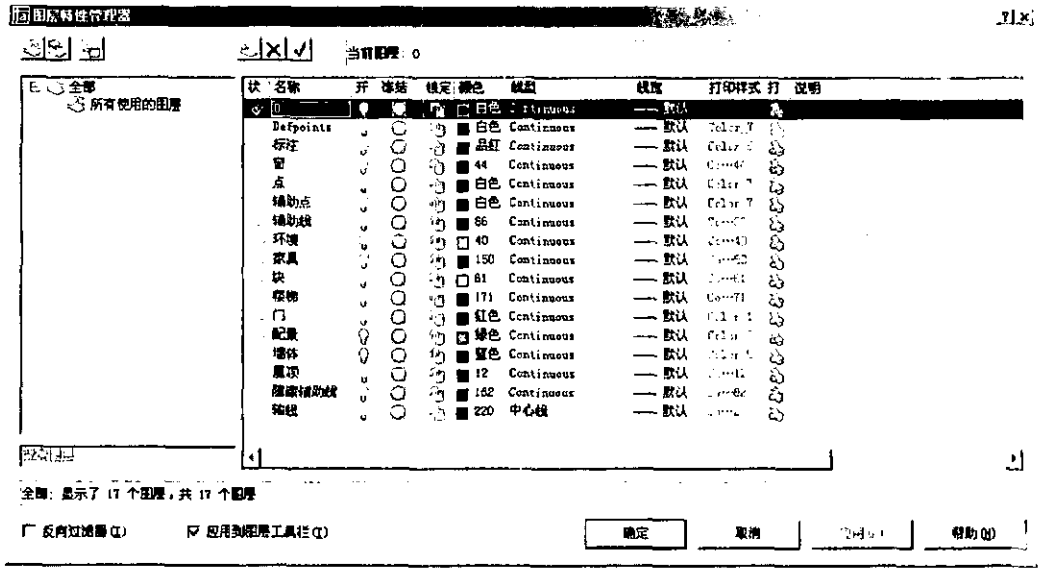


图 4.8 查看当前图形中的图层

(5) 分别执行 STYLE、DIMSTYLE 命令，查看当前图形中所包含的文字样式和标注样式。执行 ZOOM 命令，对图形进行范围缩放，可以发现图形中已经包含了导入的图框。

4.2.5 小结

ObjectDBX 是一个功能强大的类型库，能够在不打开图形的情况下对图形进行多种操作，但是需要注意的是 ObjectDBX 类型库与 AutoCAD 类型库的区别：

- ObjectDBX 库中没有 Application 对象，也就是说没有应用程序对象，所以一切与该对象有关的属性方法都不可用，如窗口的缩放、程序的加载卸载、图形的缩放等。
- ObjectDBX 库中没有事件，所以不要指望用事件来触发一些动作。
- ObjectDBX 库中没有打印项，但有打印配置。
- ObjectDBX 库中没有选择集，所以要对图形中的某些特征对象进行修改时，只能遍历整个图形。
- ObjectDBX 库中没有菜单和工具栏项。
- ObjectDBX 库中没有 Utility(实用工具)项，即该项中的各种数据的输入方法都不可用。
- ObjectDBX 中无法设置当前标注样式、文字样式、图层、布局、线型、视口、选择集等项。
- 由于 ObjectDBX 库中的 SaveAs 方法只能另存为当前版本的 DWG 文件，没有 Export 和 Import 方法，所以增加了 DXFIn 和 DXFOut 两个方法来输入和输出 DXF 文件。
- ObjectDBX 库中没有保存 SetVariable 和读取 GetVariable 系统变量的方法。
- ObjectDBX 库中没有 EndUndoMark、LoadShapeFile、PurgeAll、Regen、SendCommand、StartUndoMark 、WBlock 方法。

因为 SaveAs 方法没有版本选项，所以不能通过 ObjectDBX 来将图形保存成低版本的图形。没有符号表的当前活动项，因此不能在不打开图形的情况下修改活动的配置让以后图形打开后直接显示活动配置。没有 Wblock 方法将部分图形输出成文件，但可以用 CopyObjects 方法将图形中的某些内容复制到打开的图形中。用 ObjectDBX 保存的文件不能形成缩略图，因为它不知道当前窗口的内容。

第5章 文 字

一般的二次开发应用程序都会提供特殊字体和文字样式的创建，字体的复制是安装打包程序所要解决的问题，本章提供了创建文字样式的功能，该程序的独特之处在于读取了当前计算机中可用的各种字体，允许用户选择合适的字体来创建新的文字样式。

文字按插入点自动对齐是一个非常有用的工具，在上一个版本(《AutoCAD VBA 开发精彩实例教程》中提供)的基础上，允许文字按插入点的位置自动排列，提供了对自定义数据类型的数组进行排序的一个思路。

多行文字的格式很麻烦，但是由于它的用途实在太广，所以本章就将对其格式进行详细的剖析，从包含了格式代码的多行文字字符串中提取有效的字符。

使用 ObjectDBX 实现的批量文字替换，与上一个版本比较起来，修正了一些 Bug，并且是在不打开文件的情况下完成操作，因此速度有了明显的提高。

5.1 创建文字样式

5.1.1 实例说明

本节的实例能够读取当前操作系统和 AutoCAD 可用的所有字体名称，并显示在列表框中，用户可以选择一种字体，并设置一些文字特性，创建新的文字样式。程序的运行界面如图 5.1 所示。

5.1.2 编程思路

获得当前操作系统中所有可用的字体，可以使用 VB 内置对象 Printer 的 Fonts 属性，该属性能够返回当前显示器或活动打印机可用的所有字体名。但是 VBA 中不存在 Printer 对象，因此只能在 VB 中创建一个 ActiveX DLL，其中一个函数获得当前所有可用的字体名称，然后在 VBA 中调用即可。

AutoCAD 中可用的 SHX 字体都保存在 AutoCAD 安装路径的 Fonts 和当前系统所在盘符的\Documents and Settings\Owner\Application Data\Autodesk\AutoCAD 2005\R16.1\chs\Support 文件夹中，可以搜索所有支持文件夹中的 SHX 文件。如果用记事本打开一个 SHX 字体文件，会得到如图 5.2 所示的结果。在该文件的第一行，会用几个关键字 unifont、shapes 和 bigfont 来显示字体文件的类型，其中，bigfont 关键字表示当前字体是大字体。

要创建一个文字样式，可以使用 TextStyles 集合中的 Add 方法，然后使用 BigFontFile 或 FontFile 属性来设置 SHX 字体，或者使用 SetFont 方法设置 TrueType 字体。此外，还可以使用 Width 属性来设置宽度比例，用 ObliqueAngle 属性来设置文字的倾斜角度。

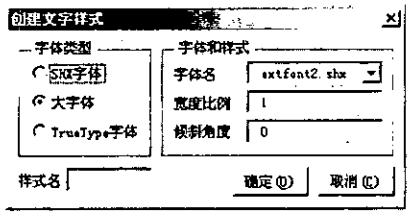


图 5.1　程序运行界面

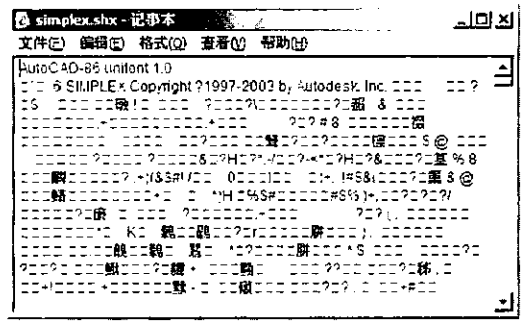


图 5.2　用记事本打开字体文件的内容

5.1.3　步骤

1. 获得操作系统中可用的字体

(1) 在 VB 6.0 中，创建一个 ActiveX DLL 工程，在属性窗口中将工程的名称修改为 SystemFonts，将系统自动建立的类模块的名称修改为 Fonts。

(2) 在 Fonts 类模块中，添加一个函数 GetSysFonts，使用 Printer 对象获得操作系统中的可用字体，其实现代码为：

```
Public Function GetSysFonts( ) As Collection
    Dim fontsCol As New Collection
    Dim i As Long
    For i = 0 To Printer.FontCount - 1
        fontsCol.Add Printer.Font(i)
    Next i

    Set GetSysFonts = fontsCol
End Function
```

(3) 将工程保存在合适的位置，选择“File”|“Make SystemFonts.dll”菜单项，创建一个 DLL 文件。

2. 获得 AutoCAD 中的 SHX 字体

在 AutoCAD 2005 中，打开 VBA 管理器，创建一个新工程，保存在适当的位置。进入 VBA 集成开发环境，选择“插入”|“模块”菜单项，向工程中添加一个标准模块，在其中添加一个函数 GetShxFont，用于获得 AutoCAD 中的 SHX 字体名称列表，其实现代码为：

```
Public Function GetShxFont(ByVal bBigFont As Boolean) As Variant
    Dim strFontFileName( ) As String                    '所有字体名称的数组
    Dim strFontPath( ) As String                        'AutoCAD 的字体文件路径

    '获得所有的支持文件路径
    strFontPath = Split(ThisDrawing.Application.Preferences.Files, ";")
```

```
'遍历所有的支持文件路径
Dim i As Integer
Dim bFirst As Boolean                    '是否是第一次查找该文件夹
Dim strFont As String                    '字体文件名称
Dim strTemp As String                    '读取到的字体文件的一行
Dim intCount As Integer                  '字体数组的维数
Dim strFontFile As String                '字体文件的位置
intCount = - 1
For i = 0 To UBound(strFontPath)
    bFirst = True
    '确保最后一个字符是"\"
    strFontPath(i) = IIf(Right(strFontPath(i), 1) = "\", strFontPath(i), strFontPath(i) & "\")

    Do
        If bFirst Then
            strFont = Dir(strFontPath(i) & " * .shx")
            bFirst = False
        Else
            strFont = Dir
        End If

        If Len(strFont) <> 0 Then
            '打开字体文件
            strFontFile = strFontPath(i) & strFont
            Open strFontFile For Input As #1
            Line Input #1, strTemp
            Close #1

            '判断字体的类型
            If bBigFont Then
                If Mid(strTemp, 12, 7) = "bigfont" Then
                    intCount = intCount + 1
                    ReDim Preserve strFontFileName(intCount)
                    strFontFileName(intCount) = strFont
                End If
            Else
                If Mid(strTemp, 12, 7) = "unifont" Or Mid(strTemp, 12, 6) = "shapes" Then
                    intCount = intCount + 1
                    ReDim Preserve strFontFileName(intCount)
                    strFontFileName(intCount) = strFont
                End If
            End If
        Else
            Exit Do
        End If
```

```
        Loop
    Next i

    GetShxFont = strFontFileName
End Function
```

函数的参数 bBigFont 用于指定所要获得的字体是否是大字体。

3. 创建文字样式

(1) 选择“插入”|“用户窗体”菜单项，向工程中添加一个对话框，并在其中添加如图 5.3 所示的控件。

其中，三个单选按钮分别命名为 optShx、optBigFont 和 optTTF，组合框名称修改为 cboFonts，三个文本框的名称为 txtStyleName、txtWidthFactor 和 txtAngle，两个命令按钮的名称修改为 cmdOk 和 cmdCancel。

(2) 选择“工具”|“引用”菜单项，在系统弹出的“引用-创建文字样式”对话框中，选中 SystemFonts 组件，如图 5.4 所示。单击“确定”按钮，关闭该对话框，完成组件的引用。

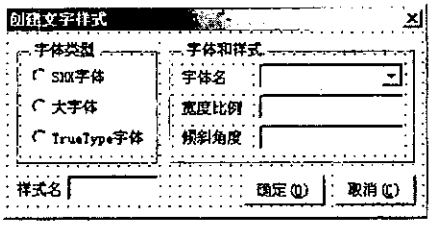


图 5.3　设计用户窗体

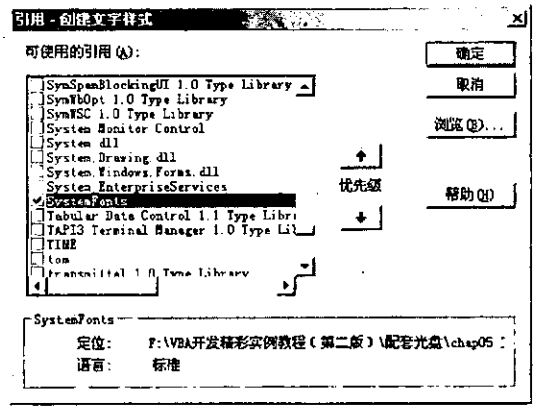


图 5.4　引用 SystemFonts 组件

(3) 在窗体的初始化事件中，设置默认选中的单选按钮以及文本框的默认值，其相关代码为：

```
Private Sub UserForm_Initialize( )
    '设置默认情况下选择的字体类型
    optBigFont.Value = True
    Call optBigFont_AfterUpdate

    txtWidthFactor.Text = 1#
    txtAngle.Text = 0
End Sub
```

(4) 在“SHX 字体”单选按钮状态更新的时候，读取 AutoCAD 中所有的 SHX 字体(不包含大字体)，并且显示在列表框中，其相关代码为：

```
Private Sub optShx_AfterUpdate( )
    If optShx.Value = True Then
        cboFonts.Clear

        Dim fonts As Variant
        fonts = GetShxFont(False)

        Dim i As Integer
        For i = 0 To UBound(fonts)
            cboFonts.AddItem fonts(i)
        Next i
        cboFonts.ListIndex = 0
    End If
End Sub
```

(5) 在“大字体”单选按钮更新的时候，读取 AutoCAD 中所有的大字体，并且显示在列表框中，其相关代码为：

```
Private Sub optBigFont_AfterUpdate( )
    If optBigFont.Value = True Then
        cboFonts.Clear

        Dim fonts As Variant
        fonts = GetShxFont(True)

        Dim i As Integer
        For i = 0 To UBound(fonts)
            cboFonts.AddItem fonts(i)
        Next i
        cboFonts.ListIndex = 0
    End If
End Sub
```

(6) 在“TrueType 字体”单选按钮更新的时候，读取当前操作系统中所有的可用字体，并且显示在列表框中，其相关代码为：

```
Private Sub optTTF_AfterUpdate( )
    If optTTF.Value = True Then
        cboFonts.Clear

        '利用 DLL 组件获得当前操作系统中的字体
        Dim fonts As New Collection
        Dim objSysFont As New SystemFonts.fonts
        Set fonts = objSysFont.GetSysFonts

        Dim i As Integer
        For i = 1 To fonts.Count
```

```
            cboFonts.AddItem fonts(i)
        Next i
        cboFonts.ListIndex = 0
    End If
End Sub
```

(7) 在“确定”按钮的单击事件中，根据用户输入的参数创建一个文字样式，其相关代码如下所示。

```
Private Sub cmdOk_Click( )
    If Len(txtStyleName.Text) = 0 Or Len(txtWidthFactor.Text) = 0 _
        Or Len(txtAngle.Text) = 0 Or Len(cboFonts.Text) = 0 Then
        MsgBox "缺少参数!"
        Exit Sub
    End If

    If HasTextStyle(txtStyleName.Text) Then
        MsgBox "图形中已经存在名为""" & txtStyleName.Text & """的文字样式!"
        Exit Sub
    End If

    '创建文字样式
    Dim objTextStyle As AcadTextStyle
    Set objTextStyle = ThisDrawing.TextStyles.Add(txtStyleName.Text)

    If optShx.Value = True Then
        objTextStyle.FontFile = cboFonts.Text
    ElseIf optBigFont.Value = True Then
        objTextStyle.BigFontFile = cboFonts.Text
    ElseIf optTTF.Value = True Then
        '获得当前字体的样式
        Dim typeFace As String
        Dim Bold As Boolean
        Dim Italic As Boolean
        Dim charSet As Long
        Dim PitchandFamily As Long
        ThisDrawing.ActiveTextStyle.GetFont typeFace, Bold, Italic, charSet, PitchandFamily

        objTextStyle.SetFont cboFonts.Text, Bold, Italic, charSet, PitchandFamily
    End If

    objTextStyle.Width = CDbl(txtWidthFactor.Text)

    Dim PI As Double
```

```
    PI = 4 * Atn(1)
    objTextStyle.ObliqueAngle = CDbl(txtAngle.Text)/180 * PI

    End
End Sub
```

设置字体时，对于普通的 SHX 字体，可以使用 FontFile 属性；对于大字体，可以使用 BigFontFile 属性来设置；对于 TrueType 字体，则可以先使用 GetFont 方法获得当前文字样式的特性，再使用 SetFont 方法设置字体。TrueType 字体不包含大字体，但是使用 FontFile 属性有时会出错，因此只得使用 SetFont 方法设置。

上面的代码中，HasTextStyle 是一个自定义函数，用于判断当前图形中是否包含指定名称的文字样式，其实现代码为：

```
Private Function HasTextStyle(ByVal strStyle As String) As Boolean
    HasTextStyle = False

    Dim objTextStyle As AcadTextStyle
    For Each objTextStyle In ThisDrawing.TextStyles
        If StrComp(objTextStyle.Name, strStyle, vbTextCompare) = 0 Then
            HasTextStyle = True
            Exit Function
        End If
    Next objTextStyle
End Function
```

(8) 在“取消”按钮的单击事件中，添加退出应用程序的代码：

```
Private Sub cmdCancel_Click( )
    End
End Sub
```

(9) 在 ThisDrawing 模块中，添加显示窗体的宏：

```
Public Sub CreateTextStyle( )
    frmMain.Show
End Sub
```

5.1.4 实例效果

(1) 在 VBA 集成开发环境中，执行 CreateTextStyle 宏，系统会弹出如图 5.5 所示的对话框。

(2) 选择一种字体类型，从“字体名”列表中选择一种字体，输入样式名、宽度比例和倾斜角度，单击“确定”按钮，完成文字样式的创建。

(3) 在 AutoCAD 中，选择“格式”|“文字样式”菜单项，系统会弹出如图 5.6 所示的“文字样式”对话框。在“样式名”列表框中，已经包含了新建的文字样式“汉字”。

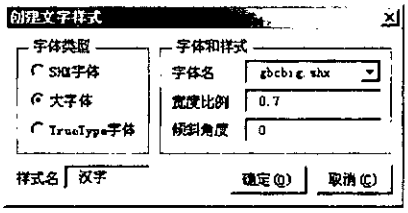


图 5.5　程序运行界面

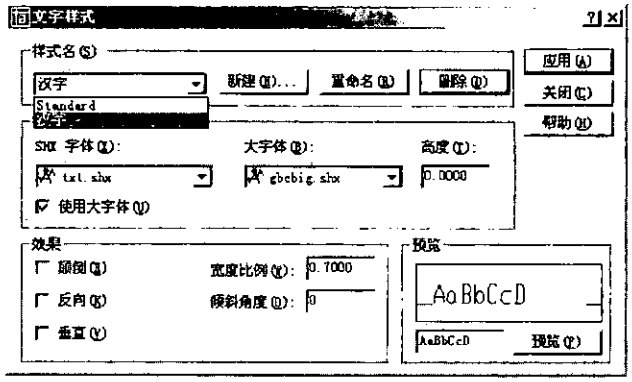


图 5.6　显示已经建立的文字样式

5.1.5　小结

学习本节内容之后，需要掌握下面的几个知识点：

- 获得当前操作系统可用的所有字体。
- 获得 AutoCAD 中可用的所有 SHX 字体。
- 创建新的文字样式。

5.2　文字按插入点自动对齐

5.2.1　实例说明

本节介绍的程序是针对文字的特性对文字进行排列，它可自动按照文字上下的顺序调整文字间的行距，用户可以指定行距，也可以由文字的范围自动计算需要的行距，如图 5.7 所示。同时可实现文字的左中右对齐方式。

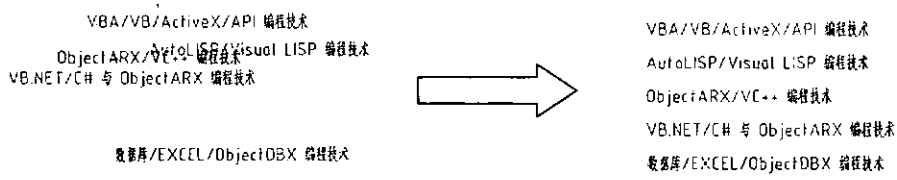


图 5.7　自动排列文字

5.2.2　编程思路

普通图元的排列，可以通过 GetBoundingBox 方法取得对象外框，再调整间距，但文字对象却不能，因为文字，特别是英文字母，是需要按照基线来排列的，如图 5.8 所示。如果按照外框调整间距，则排列后的文字会显得间距不均匀。

通过对文字对象的分析发现，文字的插入点(InsertionPoint 属性获得)所处的位置就是文字的基线位置，所以这里使用的是按照插入点调整行距。

为了使文字能够按照其原先的先后顺序调整行距，需要对选定的文字对象按它的插入点 Y 坐标进行排序，将选定的文字自上而下进行排列。排序算法有多种，这里使用的是冒泡排序法。



图 5.8 按基线排列的文字

在排序过程中，程序通过自定义数据数组保存了整个选择集文字对象，并进行排序。

文字的左右对齐则使用了文字的左边界、右边界进行对齐。

为了让程序更加直观，程序把文字中所需要使用到的一些属性都保存到自定义数据中。

5.2.3 步骤

(1) 在 AutoCAD 2005 中，打开 VBA 管理器，创建一个新工程，保存在适当的位置，进入 VBA 集成开发环境。

(2) 在“工程资源管理器”窗口中，双击 ThisDrawing 图标，打开代码窗口，在其中添加宏的定义代码：

```
Sub TextAlign( )
    Dim TextSSet As AcadSelectionSet

    '选择文字，程序只处理旋转角度为 0 的单行文字
    Dim typeArray As Variant
    Dim dataArray As Variant
    BuildFilter typeArray, dataArray, 0, "TEXT", 51, 0
    Set TextSSet = CreateSSet
    ThisDrawing.Utility.Prompt vbCr & "选择单行文字："
    TextSSet.SelectOnScreen typeArray, dataArray

    '将文字内容填写入自定义数据类型中
    Dim SSetNum As Long
    SSetNum = TextSSet.Count
    If SSetNum < 2 Then ThisDrawing.Utility.Prompt vbCr & "未选定对象，退出。": Exit Sub
    ReDim OrgTexts(SSetNum - 1)
    Dim i As Long
    Dim MinPnt As Variant
    Dim MaxPnt As Variant
    Dim IntPnt As Variant
    For i = 0 To SSetNum - 1
        TextSSet(i).GetBoundingBox MinPnt, MaxPnt
        IntPnt = TextSSet(i).InsertionPoint
        With OrgTexts(i)
            .Index = i
            Set .TextObj = TextSSet(i)
            .PntIntX = IntPnt(0)
            .PntIntY = IntPnt(1)
            .PntLefX = MinPnt(0)
```

```
            .PntRigX = MaxPnt(0)
            .PntMidX = (MinPnt(0) + MaxPnt(0))/2
        End With
    Next

    '将文字按照插入点 Y 坐标进行排序
    Dim BestText As TextWithPnt
    Dim Pnt1 As Double
    Dim Pnt2 As Double
    Dim Min As Long
    Dim Max As Long
    Min = LBound(OrgTexts)
    Max = UBound(OrgTexts)
    Dim j As Long
    Dim Best_Value As Double
    Dim Best_j As Long
    For i = Min To Max - 1
        Best_Value = OrgTexts(i).PntIntY
        BestText = OrgTexts(i)
        Best_j = i
        For j = i + 1 To Max
            If OrgTexts(j).PntIntY > Best_Value Then
                Best_Value = OrgTexts(i).PntIntY
                BestText = OrgTexts(j)
                Best_j = j
            End If
        Next j
        OrgTexts(Best_j) = OrgTexts(i)
        OrgTexts(i) = BestText
    Next i

    '计算自动调整行距时的行距距离
    Dim Num As Long
    Dim Dist As Double
    Num = TextSSet.Count
    Dist = (OrgTexts(0).PntIntY - OrgTexts(Num - 1).PntIntY)/(Num - 1)

    '要求用户输入行距及对齐方式
    Dim InputDist As Double
    Dim AlignMode As String
    On Error Resume Next
    ThisDrawing.Utility.InitializeUserInput 6
    InputDist = ThisDrawing.Utility.GetDistance(OrgTexts(0).TextObj.InsertionPoint, vbCr &
        "请指定行距<自动调整为" & Dist & ">: ")
    If Err Then Err.Clear
```

```
If InputDist = 0 Then InputDist = Dist
ThisDrawing.Utility.InitializeUserInput 0, "Left Middle Right"
AlignMode = ThisDrawing.Utility.GetKeyword(vbCr & "选择对齐方式[左对齐(L)/对中(M)/右对
    齐(R)]<不对齐>: ")

'按照行距进行排列
Dim FromPnt(2) As Double
Dim ToPnt(2) As Double
For i = 1 To Num - 1
    FromPnt(1) = OrgTexts(i).PntIntY
    ToPnt(1) = OrgTexts(0).PntIntY - InputDist * i
    OrgTexts(i).TextObj.Move FromPnt, ToPnt
Next
FromPnt(1) = 0: ToPnt(1) = 0

'按照选择的对齐方式进行排列
If AlignMode <> "" Then
    Dim BoundBox As Variant
    BoundBox = ssExtents(TextSSet)
    Dim LeftX As Double
    Dim MidX As Double
    Dim RightX As Double
    LeftX = BoundBox(0)(0)
    RightX = BoundBox(1)(0)
    MidX = (LeftX + RightX)/2

    Select Case AlignMode
        Case "Left"
            ToPnt(0) = LeftX
            For i = 0 To Num - 1
                FromPnt(0) = OrgTexts(i).PntLefX
                OrgTexts(i).TextObj.Move FromPnt, ToPnt
            Next
        Case "Right"
            ToPnt(0) = RightX
            For i = 0 To Num - 1
                FromPnt(0) = OrgTexts(i).PntRigX
                OrgTexts(i).TextObj.Move FromPnt, ToPnt
            Next
        Case "Middle"
            ToPnt(0) = MidX
            For i = 0 To Num - 1
                FromPnt(0) = OrgTexts(i).PntMidX
                OrgTexts(i).TextObj.Move FromPnt, ToPnt
            Next
```

```
        End Select
    End If
    End Sub
```

(3) 选择“插入”|“模块”菜单项,向程序中添加一个标准模块,并命名为 Functions,在其中添加宏的代码:

```
'定义文字的自定义数据类型
Public Type TextWithPnt
    Index As Long '索引序号
    TextObj As AcadText '文字对象
    PntIntX As Double '插入点的 X 坐标
    PntIntY As Double '插入点的 Y 坐标
    PntLefX As Double '左边界 X 坐标
    PntMidX As Double '居中 X 坐标
    PntRigX As Double '右边界 X 坐
End Type
Public OrgTexts( ) As TextWithPnt

'创建空白选择集的函数
Public Function CreateSSet(Optional SS As String = "mjtd") As AcadSelectionSet
    On Error Resume Next
    ThisDrawing.SelectionSets(SS).Delete
    Set CreateSSet = ThisDrawing.SelectionSets.Add(SS)
End Function

'建立选择集过滤器列表数组的函数
Public Sub BuildFilter(typeArray, dataArray, ParamArray gCodes( ))
    Dim fType( ) As Integer, fData( )
    Dim Index As Long, i As Long

    Index = LBound(gCodes) - 1

    For i = LBound(gCodes) To UBound(gCodes) Step 2
        Index = Index + 1
        ReDim Preserve fType(0 To Index)
        ReDim Preserve fData(0 To Index)
        fType(Index) = CInt(gCodes(i))
        fData(Index) = gCodes(i + 1)
    Next
    typeArray = fType: dataArray = fData
End Sub

'获取选择集对象的外框坐标的函数
Public Function ssExtents(SS As AcadSelectionSet) As Variant
    Dim Points( ), C As Long
```

```
    Dim Min As Variant, Max As Variant
    Dim i As Long, j As Long
    C = 0
    For i = 0 To SS.Count - 1
        SS.Item(i).GetBoundingBox Min, Max
        ReDim Preserve Points(0 To C + 1)
        Points(C) = Min: Points(C + 1) = Max
        C = C + 2
    Next
    ssExtents = Extents(Points)
End Function

'获取点数组中的最大和最小位置点的函数
Public Function Extents(Points)
    Dim Min As Variant, Max As Variant
    Dim i As Long, j As Long, Pt, RetVal(0 To 1)
    Min = Points(LBound(Points))
    Max = Points(LBound(Points))
    For i = LBound(Points) To UBound(Points)
        Pt = Points(i)
        For j = LBound(Pt) To UBound(Pt)
            If Pt(j) < Min(j) Then Min(j) = Pt(j)
            If Pt(j) > Max(j) Then Max(j) = Pt(j)
        Next
    Next
    RetVal(0) = Min: RetVal(1) = Max
    Extents = RetVal
End Function
```

5.2.4 实例效果

(1) 在 VBA 集成开发环境中，按 F5 键运行程序，系统在命令行提示：

```
选择单行文字：
选择对象：
```

(2) 可随意选定需要排列的文字对象，不需要有先后顺序，按 Enter 键完成选择。此时系统会提示：

```
请指定行距<自动调整为 250>：
```

其中默认的数值为程序自动计算的值，用户可按照自己的需要重新输入，或直接按下 Enter 键取默认值。这个值是在保持文字的第 1 行和最后 1 行的文字不变的情况下，调整中间文字的行距，使所有的文字均匀分布。

(3) 数值输入后按下 Enter 键，则系统会出现以下提示：

```
选择对齐方式[左对齐(L)/对中(M)/右对齐(R)]<不对齐>：
```

用户选定了对齐方式后按下 Enter 键(此处选择的是 L),则能得到如图 5.7 所示的结果。

5.2.5　小结

学习本节之后,还需要掌握下面的知识点:

- 自定义数据类型只能在标准模块中定义,而不能在其他模块中定义。
- 对象的排序不像简单数字的排序,它是通过对象的某些特征来排列对象,就像本程序是通过文字的插入点 Y 坐标,来对文字按上下位置进行排序。这样的排序不能只使用一个普通的数组来完成,一般可以通过两个数组来完成,一个为对象的数组,一个为数据的数组,在对数据的数组排序的同时,调整对象数组的顺序。也可以像本程序一样,使用自定义数据类型,操作更为简单。
- 使用自定义数据类型可以将对象中需要使用到的数据都收集在自定义数据类型中,避免程序多次调用对象的属性,这样会影响速度也影响程序的可读性。
- 文字是一种特殊的对象,它的插入点位置不是文字的左下角位置,而是左下角偏上位置,对于不同的字体或不同的文字,插入点和左下角点的相对位置会有些区别。插入点永远是在文字的基线上,所以在对文字进行排列时,使用插入点才能确保文字排列后真正行距相同。注意,这里指的不是间距,而是行距。

5.3　提取多行文字中的文字内容

5.3.1　实例说明

在 ActiveX 模型中,无法通过 TextString 属性获得多行文字对象的文字内容,该属性返回的是包含了格式代码的文字内容。例如,创建一个多行文本,输入"明经通道(http://www.mjtd.com)"之后,使用 TextString 属性获得多行文本内容的结果是"明经通道({\fArial|b0|i0|c0|p32; http}://{\fArial|b0|i0|c0|p32; www}.{\fArial|b0|i0|c0|p32; mjtd}.{\fArial|b0|i0|c0|p32; com})"。

本节实例使用了 VBScript 中的 RegExp 组件,对包含格式代码的文字内容进行处理,获得多行文字中有效的文字内容。

5.3.2　编程思路

1. 多行文字的格式代码处理

关于 AutoCAD 的多行文字可以使用的格式代码,可以参考 AutoCAD 帮助文档中的相关说明,如图 5.9 所示。

根据格式代码对多行文字所起的修饰作用,可以将其分为两类:

- 在多行文字中插入特殊字符。例如"\\"可以在多行文字中插入反斜杠,"\{...\}"可在多行文字中插入一对大括号。对这类格式代码去掉一个特殊字符"\"即可。
- 修改多行文字的字体、颜色等特性。例如"\File name; "用于修改部分文字的字体,"\S...^...; "用于对指定部分的文字进行堆叠。对这类格式代码可将"\"和"; "之间

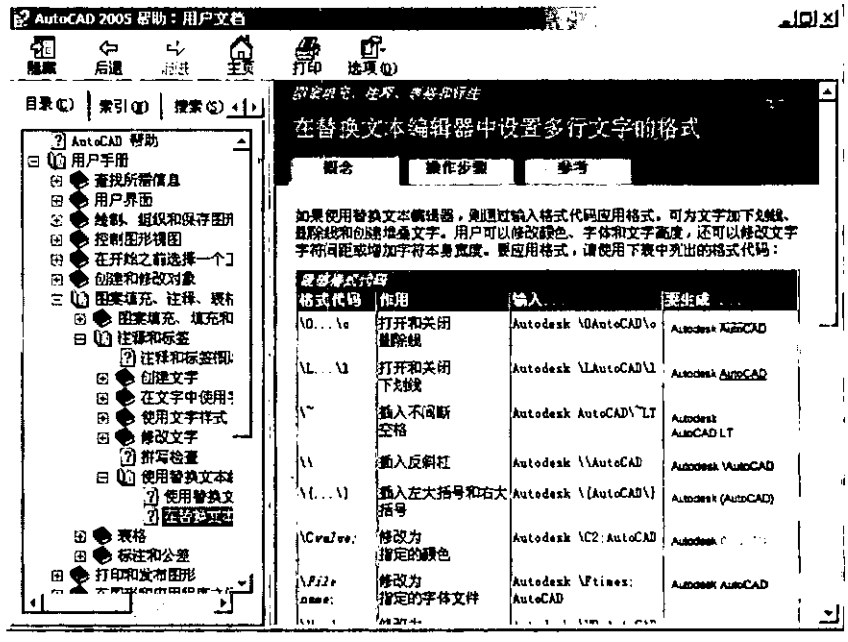


图 5.9 帮助系统中对多行文字格式代码的说明

的字符从字符串中全部删除即可。

2. VBScript 的 RegExp 组件

RegExp 是用来处理正则表达式的一个组件。正则表达式对象用来规范表达式(也就是表达式符不符合特定的要求，比如是不是 Email 地址格式等)，它具有用来检查给出的字符串是否符合规则的属性和方法。

这里不是要检查字符串是否符合一定的规则，而是要利用 RegExp 组件提供的强大的根据规则进行字符串替换的功能。例如，下面的代码构建一个 RegExp 对象，并且将字符串中所有的 am 和 is 替换成 are：

```
Public Sub EasyReplace( )
    Dim s As String
    Dim RE As Object
    '获取 Regular Expressions 组件
    Set RE = ThisDrawing.Application.GetInterfaceObject("Vbscript.RegExp")
    '查找匹配字符串时忽略大小写
    RE.IgnoreCase = True
    '搜索整个字符串
    RE.Global = True
    s = "I am a writer and he is a worker."

    '替换\\字符
    RE.Pattern = "am|is"
    s = RE.Replace(s, "are")
```

```
    MsgBox s
End Sub
```

上面的代码很短，却已经显示了使用 RegExp 组件最重要的几个属性和方法。

- IgnoreCase 属性：设置搜索匹配字符串时是否忽略大小写，True 表示忽略大小写，False 则表示大小写敏感。该属性默认值为 False。
- Global 属性：设置是搜索整个字符串替换还是只替换搜索到的第一个位置，True 表示搜索整个字符串，False 表示只替换搜索到的第一个位置。默认值为 False。
- Pattern 属性：设置或返回要搜索的正则表达式的样式。
- Replace 方法：在正则表达式的搜索中替换字符串。

3. 正则表达式中的特殊字符(Pattern 属性)

在设置 RegExp 对象的 Pattern 属性时，可以使用一些特殊的字符来指定正则表达式的规则，下面列出了可以使用的字符：

- 字符“\”：表示紧接着的字符是一个特殊字符。例如 n 匹配的是字符 n，\n 匹配的是一个换行字符，\\匹配的是\字符，\(匹配的是(字符。
- 字符^：表示匹配的字符必须在最前边，例如^A 仅会匹配字符串 Africa 中的第一个字符 A，而不会匹配第二个字符 a。
- 字符 $：与^类似，匹配最末的字符。
- 字符 *：匹配任意数量的字符。
- 字符?：匹配“?”前面的字符 0 次或者 1 次，例如，e? le?匹配 angel 中的 el 和 angle 中的 le。
- 字符.：匹配除换行符外的所有单个的字符。
- 字符(pattern)：匹配 pattern 并记录匹配的值，匹配子串能被结果数组中的元素[1]，…，[n]返回，或被 RegExp 对象的属性 $1，…，$9 返回。
- 字符 x|y：匹配字符串 x 或者 y。
- 字符{n}：这里 n 是一个正整数，精确匹配前面的 n 个字符。例如，o{2}不会匹配 Bob 中的 o，但会匹配 foooooood 中的前两个 o。
- 字符{n,}：这里 n 是一个正整数，匹配至少 n 个前面的字符。例如 o{2}不会匹配 Bob 中的 o，但是会匹配 food 和 fooooood 中所有的 o。
- 字符{n,m}：这里的 n 和 m 都是正整数。匹配至少 n 个最多 m 个前面的字符。
- 字符[xyz]：一字符列表，匹配列出的任一字符。可以通过连字符“-”指出一个字符范围。
- 字符[^xyz]：一字符补集，也就是说，它匹配除了列出的字符外的所有东西。可以使用连字符“-”指出一个字符范围。
- 字符[a-z]：指定一个字符范围。
- 字符[^m-z]：一个相反的字符范围，匹配所有不在该范围内的字符。
- 字符\b：匹配一个单词的分界线，比如一个空格(不要与[b]混淆)。例如，er\b 匹配 never 中的 er，但是不会匹配 verb 中的 er。

- 字符\B：与\b 相反，匹配一个单词的非分界线。
- 字符\d：匹配一个数字，等价于[0-9]。
- 字符\D：匹配一个非数字字符，等价于[0-9]。
- 字符\f：匹配一个表单符。
- 字符\n：匹配一个换行符。
- 字符\r：匹配一个回车符。
- 字符\s：匹配包括空格符、Tab、制表符等在内的所有空白字符。
- 字符\S：匹配除空白字符以外的任何一个单个字符。
- 字符\t：匹配一个制表符。
- 字符\v：匹配一个垂直制表符。
- 字符\w：匹配所有的数字和字母以及下划线。
- 字符\W：匹配除数字、字母外及下划线外的其他字符。
- 字符\num：匹配 num，其中 num 是一个正整数。对所获取的匹配的引用。例如，(.)匹配两个连续的相同字符。
- 字符\n：标识一个八进制转义值或一个后向引用。如果\n 之前至少 n 个获取的子表达式，则 n 为后向引用。
- 字符\xn：匹配 n，其中 n 为十六进制转义值。十六进制转义值必须为确定的两个数字长。例如，\x41 匹配 A。\x041 则等价于\x04。

关于这些特殊字符如何在 Pattern 属性中使用，可参见操作步骤中的相关内容。

5.3.3 步骤

在 AutoCAD 2005 中，打开 VBA 管理器，创建一个新工程，保存在适当的位置。进入 VBA 集成开发环境，双击“工程资源管理器”窗口中的 ThisDrawing 图标，打开代码窗口，在其中添加一个函数，用于从多行文字的 TextString 属性返回值中提取有效的文字内容：

```
Public Function GetMTextUnformatString(MTextString As String) As String
    Dim s As String
    Dim RE As Object
    '获取 Regular Expressions 组件
    Set RE = ThisDrawing.Application.GetInterfaceObject("Vbscript.RegExp")
    '忽略大小写
    RE.IgnoreCase = True
    '搜索整个字符串
    RE.Global = True
    s = MTextString

    '替换\\字符
    RE.Pattern = "\\\\"
    s = RE.Replace(s, Chr(1))
    '替换\{字符
    RE.Pattern = "\\{"
    s = RE.Replace(s, Chr(2))
```

```
    '替换\}字符
    RE.Pattern = "\\}"
    s = RE.Replace(s, Chr(3))

    '删除段落缩进格式
    RE.Pattern = "\\pi(.[^;]*);"
    s = RE.Replace(s, "")
    '删除制表符格式
    RE.Pattern = "\\pt(.[^;]*);"
    s = RE.Replace(s, "")
    '删除堆迭格式
    RE.Pattern = "\\S(.[^;]*)(\^|\/|\\)(.[^;]*);"
    s = RE.Replace(s, "$1$3")
    '删除字体、颜色、字高、字距、倾斜、字宽、对齐格式
    RE.Pattern = "(\\F|\\C|\\H|\\T|\\Q|\\W|\\A)(.[^;]*);"
    s = RE.Replace(s, "")
    '删除下划线、删除线格式
    RE.Pattern = "(\\L|\\O|\\l|\\o)"
    s = RE.Replace(s, "")
    '删除不间断空格格式
    RE.Pattern = "\\~"
    s = RE.Replace(s, " ")
    '删除换行符格式
    RE.Pattern = "\\P"
    s = RE.Replace(s, "")
    '删除换行符格式(针对 Shift + Enter 格式)
    RE.Pattern = vbLf
    s = RE.Replace(s, "")
    '删除{}
    RE.Pattern = "({|})"
    s = RE.Replace(s, "")

    '替换回\\,\{,\}字符
    RE.Pattern = "\x01"
    s = RE.Replace(s, "\")
    RE.Pattern = "\x02"
    s = RE.Replace(s, "{")
    RE.Pattern = "\x03"
    s = RE.Replace(s, "}")

    Set RE = Nothing

    GetMTextUnformatString = s
End Function
```

之所以先用其他的少见的字符 chr(1)、chr(2)和 chr(3)来替换\\、\{和\}等字符串，是

为了避免这些字符被意外删除，而在删除其他的格式字符之后，用\、{和}来代替临时的字符chr(1)、chr(2)和chr(3)，也就完成了第一种类型的格式字符的处理。

5.3.4 实例效果

(1) 在VBA集成开发环境中，添加一个宏GetMTextString，用于提示用户在图形窗口中拾取多行文字，并且显示其有效的文字内容，其相关代码为：

```
Public Sub GetMTextString( )
    Dim objMText As AcadMText
    Din ptPick As Variant
    ThisDrawing.Utility.GetEntity objMText, ptPick

    MsgBox GetMTextUnformatString(objMText.TextString)
End Sub
```

(2) 执行GetMTextString宏，在图形窗口中拾取文字内容为“明经通道(http://www.mjtd.com)”的多行文字，能够得到如图5.10所示的结果。

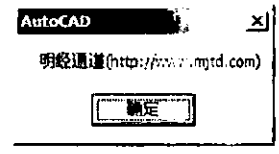


图5.10 获得多行文字有效文字内容

5.3.5 小结

非常遗憾，在MSDN中无法获得关于RegExp组件的详细资料，不过可以在MSDN的国际网站获得该组件的参考资料。

此外，本节的实例不需要引用Microsoft VBScript Regular Expressions 5.5类型库，而是在程序中使用下面的代码来获得相同的效果：

```
Set RE = ThisDrawing.Application.GetInterfaceObject("Vbscript.RegExp")
```

5.4 使用ObjectDBX实现批量文字替换

5.4.1 实例说明

本节所要介绍的实例，将用ObjectDBX技术来进行替换的操作，程序运行的界面如图5.11所示。

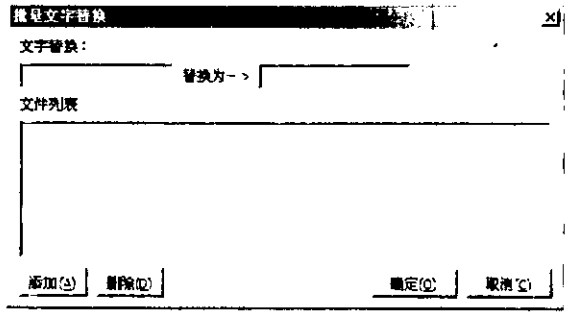


图5.11 程序运行界面

用户可以使用“添加”、“删除”按钮调整列表框中的图形名称，单击“确定”按钮之后，列表框中所有的图形文件都会被进行文字替换的操作。

5.4.2　编程思路

使用 ObjectDBX 技术在第 4 章已经进行介绍，本节的实例将使用 ObjectDBX 对象打开所要进行文字替换的文档，然后遍历其中的所有图元，对文字和多行文字进行替换的操作。

此外，本节介绍的程序使用了打开文件标准对话框，但是允许用户同时选择多个文件。与《AutoCAD VBA 开发精彩实例教程》中对应的实例相比，有如下两个区别。

- 使用 ObjectDBX 技术代替了以前打开文档，进行文字替换，然后再关闭文档的过程，提高了处理的速度。
- 原书中的程序有一个 Bug，就是在选择驱动器根目录下的文件时，会得到类似 F:\\test.dwg 的结果，很明显，这样的文件名是无效的。本节的实例就解决了这个问题。

5.4.3　步骤

(1) 在 AutoCAD 2005 中，打开 VBA 管理器，创建一个新工程，保存在适当的位置，进入 VBA 集成开发环境。

(2) 选择“插入”|“用户窗体”菜单项，向程序中添加一个用户窗体，并在其中放置如图 5.12 所示的控件。

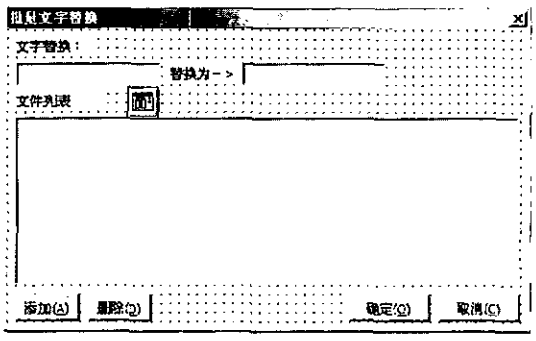


图 5.12　添加窗体中的控件

窗体的名称修改为 frmMain，文本框的名称修改为 txtFind 和 txtReplace，列表框的名称修改为 lstFile，四个命令按钮的名称修改为 cmdOpen、cmdDelete、cmdCancel 和 cmdOk，标准对话框控件的名称修改为 comDlg。

(3) 选择“插入”|“模块”菜单项，向程序中添加一个标准模块，其中添加宏的启动代码：

```
Sub ReplaceText( )
    frmMain.Show
End Sub
```

(4) 选择“工具”|“引用”菜单项，在弹出的对话框中添加对 ObjectDBX 类型库的引用。在窗体的初始化事件中，清空列表框的元素，其实现代码为：

```
Private Sub UserForm_Initialize( )
```

```
    lstFile.Clear

    '根据 AutoCAD 的版本来决定使用 ObjectDBX 的版本
    If Left(Version, 2) = "15" Then
        Set objDBX = CreateObject("ObjectDBX.AxDbDocument.1")
    ElseIf Left(Version, 2) = "16" Then
        Set objDBX = CreateObject("ObjectDBX.AxDbDocument.16")
    End If
End Sub
```

其中，objDBX 是一个模块级的变量，被定义在窗体代码模块的通用声明区内：

```
Dim objDBX As Object          'ObjectDBX 对象
```

(5) 窗体中“添加”按钮的单击事件，用于显示标准对话框，获取用户的选择，并将选择的结果添加到列表框中，其实现代码为：

```
Private Sub cmdOpen_Click( )
    On Error GoTo errHandle

    Dim i As Integer
    Dim Y As Integer
    Dim Z As Integer
    Dim fileNames( ) As String

    With comDlg
        .CancelError = True
        .MaxFileSize = 32767
        .Flags = cdlOFNHideReadOnly Or cdlOFNAllowMultiselect Or cdlOFNExplorer Or
            cdlOFNNoDereferenceLinks
        .DialogTitle = "选择图形文件"
        .Filter = "图形文件(*.dwg)|*.dwg|所有文件(*.*)|*.*"
        .FileName = ""
        .ShowOpen
    End With

    comDlg.FileName = comDlg.FileName & Chr(0) '这些文件名是用空字符 Chr(0)分隔符，而不是空
                                               '格分隔符隔开

    Z = 1
    For i = 1 To Len(comDlg.FileName)
        'InStr 函数，返回 Variant (Long)，指定一字符串在另一字符串中最先出现的位置
        '语法 InStr(起点位置, string1, string2)
        i = InStr(Z, comDlg.FileName, Chr(0))
        If i = 0 Then Exit For
        ReDim Preserve fileNames(Y)
        'Mid 函数，返回 Variant (String)，其中包含字符串中指定数量的字符
```

```
        '语法 Mid(string, start[, length])
        fileNames(Y) = Mid(comDlg.FileName, Z, i - Z)
        Z = i + 1
        Y = Y + 1
    Next i

    '向列表框中添加对象
    Dim count As Integer
    count = lstFile.ListCount
    If Y = 1 Then
        If Not HasItem(fileNames(Y - 1)) Then
            lstFile.AddItem fileNames(Y - 1), count
        End If
    Else
        For i = 1 To Y - 1
            If StrComp(Right$(fileNames(0), 1), "\") = 0 Then
                fileNames(i) = fileNames(0) & fileNames(i)
            Else
                fileNames(i) = fileNames(0) & "\" & fileNames(i)
            End If

            If Not HasItem(fileNames(i)) Then
                lstFile.AddItem fileNames(i), i - 1 + count
            End If
        Next i
    End If

errHandle:

End Sub
```

其中，HasItem是自定义函数，用于检查列表框中是否存在指定名称的项目，其实现代码为：

```
Private Function HasItem(ByVal strDwgName As String) As Boolean
    HasItem = False

    Dim i As Integer
    For i = 0 To lstFile.ListCount - 1
        If StrComp(lstFile.List(i), strDwgName, vbTextCompare) = 0 Then
            HasItem = True
            Exit Function
        End If
    Next i
End Function
```

(6) 窗体中“删除”按钮的单击事件，用于从列表框中删除指定的元素，其实现代码为：

```
Private Sub cmdDelete_Click( )
    '确认列表框包含列表项
    If lstFile.ListCount >= 1 Then
        '如果没有选中的内容,用上一次的列表项
        If lstFile.ListIndex = -1 Then
            MsgBox "请选择列表中的图形名称!"
            Exit Sub
        End If
        lstFile.RemoveItem (lstFile.ListIndex)
    End If

End Sub
```

(7) 窗体中"确定"按钮的单击事件,用于打开每一个列表框中显示的图形文件,并且对文件进行文字替换的操作,其实现代码为:

```
Private Sub cmdOk_Click( )
    Dim adText As AcadText
    Dim adMText As AcadMText
    Dim adSS As AcadSelectionSet
    Dim fType(0 To 1) As Integer, fData(0 To 1)
    Dim i As Integer

    If txtFind.Text = "" Or txtReplace.Text = "" Then
        MsgBox "输入所要替换的字符串内容!"
        Exit Sub
    End If
    If lstFile.ListCount = 0 Then
        MsgBox "请添加所要操作的图形!"
        Exit Sub
    End If

    '获得替换数据
    Dim strFind As String
    Dim strReplace As String
    strFind = txtFind.Text
    strReplace = txtReplace.Text

    '打开图形进行操作
    For i = 0 To lstFile.ListCount - 1
        Call ReplaceTextInDwg(lstFile.List(i), strFind, strReplace)
    Next i

    End
End Sub
```

其中,ReplaceTextInDwg 是一个自定义函数,用于打开指定的图形,并且使用 ObjectDBX

技术完成文字的查找和替换，其实现代码为：

```
Private Sub ReplaceTextInDwg(ByVal strDwgName As String, ByVal strFind As String, _
                             ByVal strReplace As String)
    '打开指定的图形
    objDBX.Open strDwgName

    Dim ent As AcadEntity
    For Each ent In objDBX.ModelSpace
        If TypeOf ent Is AcadText Or TypeOf ent Is AcadMText Then
            With ent
                If InStr(.TextString, strFind) Then .TextString = ReplaceStr(.TextString,
                   strFind, strReplace, False)
            End With
        End If
    Next ent

    objDBX.SaveAs strDwgName
End Sub
```

上面代码中使用了自定义函数 ReplaceStr，用来完成对图形中指定字符串的替换，其实现代码为：

```
Public Function ReplaceStr(ByVal searchStr As String, ByVal oldStr As String, _
        ByVal newStr As String, ByVal firstOnly As Boolean) As String
    '对错误操作的处理
    If searchStr = "" Then Exit Function
    If oldStr = "" Then Exit Function

    ReplaceStr = ""
    Dim i As Integer, oldStrLen As Integer, holdStr As String, StrLoc As Integer

    '计算原来字符串的长度
    oldStrLen = Len(oldStr)
    StrLoc = InStr(searchStr, oldStr)

    While StrLoc > 0
        '获得图形中文字对象位于查找字符串之前的字符串
        holdStr = holdStr & Left(searchStr, StrLoc - 1) & newStr
        '获得文字对象位于查找字符串之后的字符串
        searchStr = Mid(searchStr, StrLoc + oldStrLen)
        StrLoc = InStr(searchStr, oldStr)
        If firstOnly Then ReplaceStr = holdStr & searchStr: Exit Function
    Wend

    ReplaceStr = holdStr & searchStr
End Function
```

(8) 可能某些图形文件的名称过长，导致列表框中无法显示其完整的路径和文件名称，因此在列表框的双击事件中，实现了显示当前选择的列表项内容的功能：

```
Private Sub lstFile_DblClick(ByVal Cancel As MSForms.ReturnBoolean)
    MsgBox lstFile.List(lstFile.ListIndex)
End Sub
```

(9) 最后，在窗体“取消”按钮的单击事件中，添加用于退出程序的代码：

```
Private Sub cmdCancel_Click( )
    End
End Sub
```

5.4.4 实例效果

(1) 在 VBA 集成开发环境中，按 F5 键运行程序，系统会弹出如图 5.13 所示的“批量文字替换”对话框。

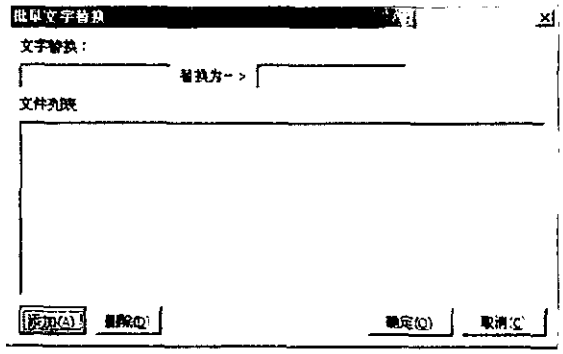


图 5.13 程序运行界面

(2) 单击“添加”按钮，系统会弹出如图 5.14 所示的“选择图形文件”对话框，用户可以从中选择多个图形文件，单击“打开”按钮，就可以将文件添加到“批量文字替换”对话框的文件列表中。

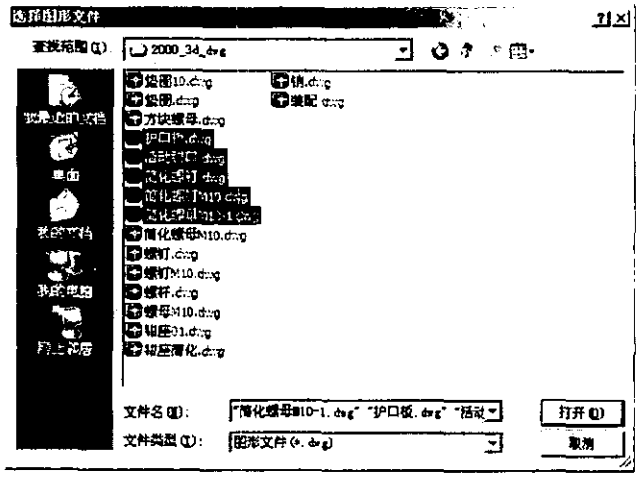


图 5.14 选择多个图形文件

(3) 在“批量文字替换”对话框中输入要替换的字符串和替换内容，单击“确定”按钮，就可以对指定的多个图形进行文字替换的操作。

5.4.5　小结

使用 ObjectDBX 技术来对图形中的文字进行替换，避免了文档打开和关闭的繁琐操作，在速度上有一定优势，但是使用 ObjectDBX 的时候需要注意，通过 ObjectDBX 打开的图形要确保它当前不被 AutoCAD 打开，否则就会出错。

第6章　标注和公差

AutoCAD直到2006版本才提供了圆弧长度标注的功能，因此在2006以前的版本中提供标注圆弧长度的功能仍有现实意义，并且该程序中涉及了多行文字格式代码的使用，可以加深对多行文字格式代码使用范围的了解。

锁定标注文字的程序对于创建局部放大图形时非常有用，技术方面涉及到匿名块、尺寸标注对象的组成、GetSubEntity方法的使用等。

该程序能够列举出AutoCAD中提供的所有公差符号，从程序中可以发现访问公差内容的方法。

6.1　标注圆弧长度

6.1.1　实例说明

在AutoCAD的标注中，没有标注圆弧长度这个功能，这里通过创建角度标注并修改角度值为弧长的方法来实现这个功能。

实际上在《AutoCAD VBA精彩实例教程》(清华大学出版社，2004年1月)一书中，已经在8.6节提供了类似功能的程序，这里提供的程序有较大的改进：

- 圆弧上增加圆弧标志部分，实现真正的弧长标注。
- 修正了标注文字小数点位数与实际不符的问题。
- 修正了只有整数时多出一个小数点符号的问题。

6.1.2　编程思路

通过ActiveX模型提供的AddDim3PointAngular方法创建一个角度标注对象(AcadDim3PointAngular)，然后使用角度标注对象的TextOverride属性来将角度值修改为圆弧的弧长，从而实现弧长标注。

6.1.3　步骤

(1) 在AutoCAD 2004中，打开VBA管理器，新建一个工程，保存在适当的位置，进入VBA集成开发环境。

(2) 在“工程资源管理器”窗口中双击ThisDrawing图标，打开代码窗口，在其中添加宏的实现代码，如下所示。

```
Public Sub DimArcLeng( )
    Dim Arc As AcadArc
    Dim Pnt As Variant
    Err.Clear
```

```
On Error Resume Next

'选择圆弧
ThisDrawing.Utility.GetEntity Arc, Pnt, "请选择圆弧："
If Err.Number <> 13 And Err.Number <> 0 Then Exit Sub

Do Until Arc.ObjectName = "AcDbArc"
    Err.Clear
    ThisDrawing.Utility.GetEntity Arc, Pnt, "你所选的不是圆弧,请重新选择圆弧："
    If Err.Number <> 13 And Err.Number <> 0 Then Exit Sub
Loop

'获取圆弧各属性
Dim Leng As Double
Dim SPnt As Variant
Dim EPnt As Variant
Dim CPnt As Variant
Leng = Arc.ArcLength
SPnt = Arc.StartPoint
EPnt = Arc.EndPoint
CPnt = Arc.Center

'选择标注点
Dim PntforDim As Variant
PntforDim = ThisDrawing.Utility.GetPoint(, "选择标注点位置：")

'对圆弧进行角度标注
Dim DimAng As AcadDim3PointAngular
Set DimAng = ThisDrawing.ModelSpace.AddDim3PointAngular(CPnt, SPnt, EPnt, PntforDim)

'获取角度标注的精度控制(小数点位)
Dim FormatDot As Integer
Dim FormatTxt As String
FormatDot = DimAng.TextPrecision

'转换为精度控制格式
FormatTxt = "0"
Dim I As Integer
For I = 1 To FormatDot
    If I > 1 Then
        FormatTxt = FormatTxt & "0"
    Else    'i = 1
        FormatTxt = FormatTxt & ".0"
    End If
Next
```

```
    '更改角度标注的文字内容
    DimAng.TextOverride = "{\Fgdt.shx; ^}\P" & Format(Leng, FormatTxt)
End Sub
```

首先在一个 Do...Loop 语句中对 GetEntity 方法进行错误处理，确保用户选择的是圆弧。可能会注意到下面的语句：

```
If Err.Number <> 13 And Err.Number <> 0 Then Exit Sub
```

错误号 13 指数据类型不对，因为已经定义对象为圆弧，如选的不是圆弧则会产生数据类型出错；错误号 0 指没有错误。

在这里安排这两项的主要目的是：选中对象但选的不是圆弧，则会重复提示用户选择；如果未选择到对象或直接按 Enter 键(或 Esc 键)，则直接退出程序。

弧长长度标注的精度由当前标注样式的角度标注精度决定，如图 6.1 所示。在程序中，可以通过角度标注对象的 TextPrecision 属性来获得精度值：

```
FormatDot = DimAng.TextPrecision
```

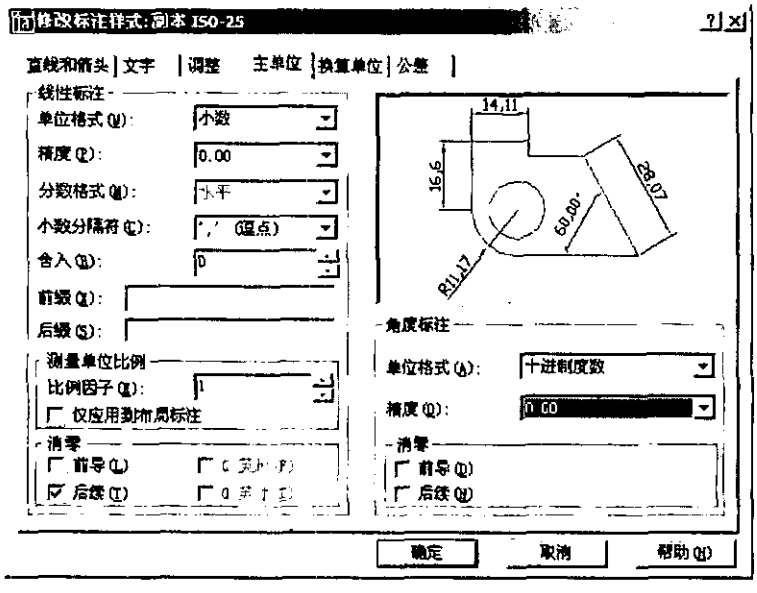


图 6.1 设置圆弧长度标注的精度

获得精度之后，就可以将其转化成精度控制的格式(形如 0.000 的格式)，其相关代码为：

```
FormatTxt = "0"
Dim i As Integer
For i = 1 To FormatDot
    If i > 1 Then
        FormatTxt = FormatTxt & "0"
    Else    'i = 1
        FormatTxt = FormatTxt & ".0"
```

```
    End If
Next
```

更改角度标注的文字内容使用了下面的代码：

```
DimAng.TextOverride = "{\Fgdt.shx;^}\P" & Format(Leng, FormatTxt)
```

其中，Format 函数根据格式表达式中的指令来格式化指定的字符串，其具体用法可以参考 MSDN 中的说明。这里，{\Fgdt.shx;^}\P 这串字符中包含了几个需要说明的元素：

- 大括号({})用于构建一个字符单元，内部和外部所使用的格式代码不会相互影响。
- \F 是一个格式代码，用于为指定的字符串指定字体文件，\Fgdt.shx;^表示字符^使用的字体由字体文件 gdt.shx 来决定。
- 括号外面的\P 是一个格式代码，用于结束当前段落(后面的字符会在一个新的段落中出现)。

关于 AutoCAD 中多行文字所使用的格式代码，可以参考 AutoCAD 帮助系统中的说明。

6.1.4　实例效果

(1) 在 VBA 集成开发环境中，按 F5 键执行程序。

(2) 回到 AutoCAD 界面，此时可看到命令行中提示“请选择圆弧：”，可以选择图形中的圆弧。选择后命令行中会提示“选择标注点位置：”，此时可以在圆弧的附近选择一个标注的位置。

(3) 完成以上操作后，程序自动标注圆弧的弧长，得到如图 6.2 所示的结果。

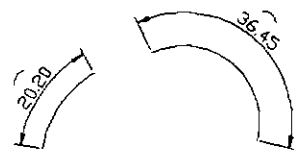


图 6.2　标注圆弧长度的结果

6.1.5　小结

学习本节内容之后，对尺寸标注的特性应会有更为深入的了解，特别是文字替代值的使用。如果读者有兴趣，可以创建一个用户，直接输入文字替代值的线性标注程序。

标注对象的文字部分其实是多行文字，其内容可以按照多行文字的编辑方法来编辑。这里就是利用多行文字的可以部分修改字体以及换行的特性，在标注文字上增加一个圆弧标志。

6.2　锁定标注文字不随对象缩放而动态变化

6.2.1　实例说明

标注对象的文字部分会随着标注点的变化而按其实际尺寸变化。如图 6.3 所示，矩形的长度为 10，所以标注的尺寸为 10，但如果将矩形放大一倍，为 20 时，标注的尺寸会跟随着

更改为 20。在某些情况下，如需要局部放大图形时，标注的尺寸是不允许改变的。所以，必须想办法把标注文字固定下来。

一般情况下，通过直接手工修改文字内容可以达到目的，但尺寸标注数量多、或本身尺寸还有公差的情况下，修改很麻烦。

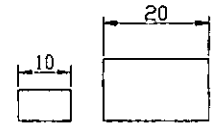


图 6.3 标注文字随标注对象的变化而变化

6.2.2 编程思路

对 AutoCAD 图形格式有较深认识的开发者可以发现，标注对象本身就是以“*D”名称开头的匿名块。标注使用匿名块的方式是为了方便图形的显示和组织。

在 DXF 组码中，标注对象具有组码 2，它包含了构成标注对象的图元的匿名块名称。所以在 AutoLISP 开发中，可以轻易地得到标注对象所对应的匿名块。而在 VBA 中，对象模型并没有提供相应的属性或方法来取得该匿名块名。

通过对 AutoCAD 图形数据规律的分析，当标注对象生成时，会同时生成一个保存该标注对象的匿名块。而在图形数据库中，最后生成的图元或对象，都是被存到相应数据类型中的最后的位置。

利用这样的特点，在编程时可以通过复制的方法把需要固定标注文字的标注对象复制一个副本，而在复制副本的同时，图形数据库中会同时生成与标注对象相关联的匿名块。通过获取图块集合中的最后一个对象，则可获取该匿名块。

得到保存标注对象的匿名块后，就可以遍历块中的对象，找到其中的文字对象。对于所有的标注对象，文字只有一项，所以在得到文字对象时，则可以得到所需要的文字内容。该文字内容就是显示给用户的文字内容。把该文字内容替换掉标注中的文字，则可以将文字锁定下来而不随缩放动态变化。

6.2.3 步骤

1. 通过复制的方法锁定文字内容

(1) 在 AutoCAD 中，打开 VBA 管理器，新建一个工程，命名为 FixDimByCopyDim. dvb，保存在适当的位置，进入 VBA 集成开发环境。

(2) 在“工程资源管理器”窗口中双击 ThisDrawing 图标，打开代码窗口，在其中添加宏的实现代码：

```
'单选标注对象锁定文字内容
Sub FixSingleDim( )
    Dim Ent As AcadEntity
    Dim Pnt As Variant
    '选择对象
    ThisDrawing.Utility.GetEntity Ent, Pnt, vbCr & "选择要锁定文字的标注对象："
    If Ent.ObjectName Like "AcDb*Dimension" Then
        '锁定标注的文字内容
        FixDimText Ent
```

```
    End If
End Sub

'将图形中所有的标注对象锁定文字内容
Sub FixAllDim( )
    Dim SSet As AcadSelectionSet
    On Error Resume Next
    '建立选择集
    ThisDrawing.SelectionSets("mccad").Delete
    Set SSet = ThisDrawing.SelectionSets.Add("mccad")
    '建立过滤器
    Dim fType(0) As Integer
    Dim fData(0) As Variant
    fType(0) = 0
    fData(0) = "DIMENSION"
    '选择过滤出图形中所有的标注对象
    SSet.Select acSelectionSetAll, , , fType, fData
    Dim i As Long
    For i = 0 To SSet.Count - 1
        '锁定标注的文字内容
        FixDimText SSet(i)
    Next
End Sub

'锁定标注文字的函数
'作者：明经通道 mccad
Function FixDimText(Dimension As AcadDimension) As String
    '在复制标注对象前先保存当前图形中的块数量
    Dim BlockCount As Long
    BlockCount = ThisDrawing.Blocks.Count
    '复制需要锁定文字内容的标注对象
    Dim CopyDimension As AcadDimension
    Set CopyDimension = Dimension.Copy
    '检查块数量是否增加，而且新增加的块名前缀是否为 * D
    Dim NewBlockCount As Long
    NewBlockCount = ThisDrawing.Blocks.Count
    If NewBlockCount = BlockCount + 1 And Left(ThisDrawing.Blocks(BlockCount).Name, 2) = " * D"
      Then
        '遍历块中的对象，取得文字内容
        Dim EntityInBlock As AcadEntity
        For Each EntityInBlock In ThisDrawing.Blocks(BlockCount)
            If EntityInBlock.ObjectName = "AcDbMText" Then
                '将文字内容替换掉标注对象的文字内容
                Dimension.TextOverride = EntityInBlock.TextString
                Exit For
```

```
            End If
        Next
    End If
    '删除复制的标注对象
    CopyDimension.Delete
    FixDimText = Dimension.TextOverride
End Function
```

为了让这个功能更为灵活，程序将锁定标注文字的功能放在自定义函数中使用，用户可直接复制该函数用于自己的程序中。

在上面的程序中，FixAllDim 过程可将图形中所有的标注对象的文字锁定，而 FixSingleDim 过程则只锁定指定的标注对象的文字。

2. 通过选择子图元的方法锁定文字内容

(1) 在 AutoCAD 中，打开 VBA 管理器，新建一个工程，命名为 FixDimBySubEntity.dvb，保存在适当的位置，进入 VBA 集成开发环境。

(2) 在“工程资源管理器”窗口中双击 ThisDrawing，打开代码窗口，在其中添加宏的实现代码：

```
'锁定标注对象的文字内容(通过选择子图元方法)
Sub FixDimText( )
    Dim Ent As AcadEntity
    Dim Pnt As Variant
    Dim transMatrix As Variant, contextData As Variant

    '选择标注对象中的子图元
    ThisDrawing.Utility.GetSubEntity Ent, Pnt, transMatrix, contextData, "选择标注对象："

    '获取选定子图元的的父对象
    Dim BlkId As Long
    BlkId = Ent.OwnerID
    Dim BlkName As String
    Dim TextString As String
    BlkName = ThisDrawing.ObjectIdToObject(BlkId).Name

    '判断子图元的父对象是否为标注对象的匿名块
    If Left(BlkName, 2) = "*D" Then
        Dim EntityInBlock As AcadEntity
        For Each EntityInBlock In ThisDrawing.ObjectIdToObject(BlkId)
            '取得块中的文字内容
            If EntityInBlock.ObjectName = "AcDbMText" Then
                TextString = EntityInBlock.TextString
                Exit For
            End If
        Next
```

```
            '用文字对象的文字内容替换标注中的文字内容
            If TextString <> "" Then
                ThisDrawing.ObjectIdToObject(contextData(0)).TextOverride = TextString
            End If
        End If
    End Sub
```

首先使用 GetSubEntity 方法提示用户选择标注对象的子图元，该方法定义为：

```
object.GetSubEntity Object, PickedPoint, TransMatrix, ContextData[, Prompt]
```

其中，第一个 Object 是 Utility 对象；第二个 Object 输出被选中的对象或子实体；PickedPoint 返回用户所拾取点的 WCS 坐标；TransMatrix 返回被选中的实体上所应用的变换矩阵；ContextData 是一个数组，包含了被选择的实体中所包含的所有嵌入对象的 ID；Prompt 则是显示在命令行中提示用户的语句。

用户选择某个子实体之后，使用 OwnerID 属性获得其父对象的 ID，通过 ObjectIdToObject 方法可根据对象 ID 获得对应的对象。判断一个块定义是否是标准对象的匿名块，可检测块定义名称的前两个字符，如果前两个字符是 * D，表示指定的块定义是标注对象的匿名块。

contextData 中放的数据是被选子图元的所有嵌套对象的对象 ID，也就是说包含该子图元的所有上级对象。对于标注中的子图元，标注则为其第一层上级对象。因此，ThisDrawing. ObjectIdToObject(contextData(0))所指代的就是用户所选择的尺寸标注对象。

6.2.4　实例效果

(1) 在 VBA 集成开发环境中，将光标放置在 FixSingleDim 过程中，按 F5 键执行程序。

(2) 回到 AutoCAD 界面，此时可看到命令行中提示“请选择需要固定的标注对象：”，可以选择图形中的标注对象。选择后按下 Enter 键，此时指定的标注对象将不知不觉地被锁定文字。

(3) 如果在 VBA 集成开发环境中运行 FixAllDim 宏，回到 AutoCAD 界面，此时图形中所有的标注对象将不知不觉地被锁定文字。

(4) 完成以上操作后，用户可缩放该标注对象或拉动标注对象的标注点，可以看到，标注对象上的文字不会随着标注对象的变化而变化，如图 6.4 所示。

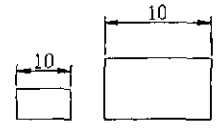


图6.4　标注对象上的文字不随标注对象的变化而变化

6.2.5　小结

学习本节之后，回顾下面的知识点：

- 标注对象是由匿名块形成的，AutoCAD 修改标注对象的内部机制是修改匿名的外观。
- 标注对象的文字部分就是匿名块中的多行文字对象，而标注块中的多行文字对象只有一个，所以取得该多行文字对象就可以得到标注对象的文字内容。

- 含公差的标注对象在匿名块中多行文字中的表现方式为堆叠方式，所以，这样锁定的文字内容不会使带公差的标注文字消失。
- 取得标注对象匿名块的方法无法通过 DXF 组码获取，因为对象模型并不直接提供该方法。所以只能通过复制标注对象，查看所新增的块的情况。
- 通过 GetSubEntity 方法则选择到标注对象中的子图元。而通过子图元的 OwnerID 属性可以得到其父对象的 ID 号，通过 ID 则可得到对象。所获得的对象就是匿名块。
- GetSubEntity 方法中，前两个参数与 GetEntity 方法的两个参数的用法相同，而后两个参数则有着特殊的用途。TransMatrix 参数代表着选定图元的转换矩阵，而 ContextData 参数则为被选择子图元的所有嵌套对象的对象 ID。

值得一提的是，出于程序简介性的考虑，没有为 GetSubEntity 方法提供错误处理，读者可以仿照书中多次出现的 GetEntity 方法错误处理，为本节的程序添加完善的错误处理。

6.3 列举所有的形位公差符号

6.3.1 实例说明

本节的实例将在图形窗口中列举所有的形位公差符号，通过这个实例的学习，将能够了解到形位公差在 AutoCAD 中的实现方式，修改和创建形位公差会变得简单。

6.3.2 编程思路

在 AutoCAD 2005 中，选择“标注”|“公差”菜单项，系统会弹出如图 6.5 所示的“形位公差”对话框。在对话框中选择图中的符号，并且输入公差值，单击“确定”按钮，在图形窗口中能够得到如图 6.6 所示的公差。

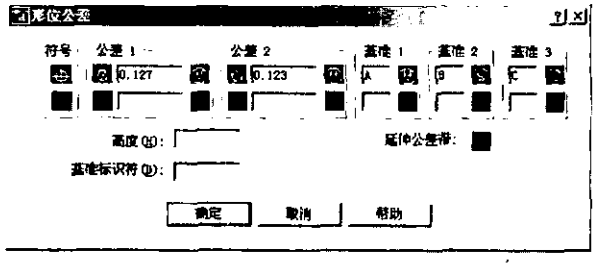


图 6.5 添加形位公差

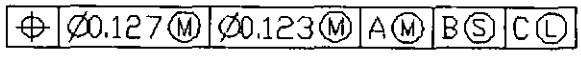


图 6.6 创建的形位公差

在图形窗口中选择创建的形位公差，选择“工具”|“对象特性管理器”菜单项，系统会弹出“特性”窗口，在其中的“文字”选项组的“文字替代”文本框中能够得到如图 6.7 所示的

结果。

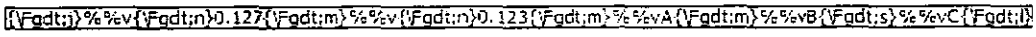


图 6.7　形位公差的文字替代值

从形位公差的文字替代值可以看出，形位公差的文字格式与多行文字类似，都使用了某些特殊的格式代码。

简单来说，{\Fgdt; j}中的"\Fgdt; "用于指定文字的字体，j 和前面的字体结合起来用于指定一种公差符号；%%v 用于指定公差之间的分隔符。

6.3.3　步骤

在 AutoCAD 2005 中，打开 VBA 管理器，创建一个新工程，保存在适当的位置。进入 VBA 集成开发环境，双击"工程资源管理器"窗口中的 ThisDrawing 图标，打开代码窗口，其实现代码如下：

```
Public Sub AddTolerance( )
    Dim toleranceObj As AcadTolerance
    Dim textString As String
    Dim insertionPoint(0 To 2) As Double
    Dim direction(0 To 2) As Double

    '定义文字替代值
    textString = "{\Fgdt; j}%%v位置度 - j" & vbCrLf & _
                 "{\Fgdt; r}%%v同轴度 - r" & vbCrLf & _
                 "{\Fgdt; i}%%v对称度 - i" & vbCrLf & _
                 "{\Fgdt; f}%%v平行度 - f" & vbCrLf & _
                 "{\Fgdt; b}%%v垂直度 - b" & vbCrLf & _
                 "{\Fgdt; a}%%v倾斜度 - a" & vbCrLf & _
                 "{\Fgdt; g}%%v圆柱度 - g" & vbCrLf & _
                 "{\Fgdt; c}%%v平面度 - c" & vbCrLf & _
                 "{\Fgdt; e}%%v圆度 - e" & vbCrLf & _
                 "{\Fgdt; u}%%v直线度 - u" & vbCrLf & _
                 "{\Fgdt; d}%%v面轮廓度 - d" & vbCrLf & _
                 "{\Fgdt; k}%%v线轮廓度 - k" & vbCrLf & _
                 "{\Fgdt; h}%%v圆跳动度 - h" & vbCrLf & _
                 "{\Fgdt; t}%%v全跳动度 - t" & vbCrLf & _
                 "{\Fgdt; l}%%vl" & vbCrLf & _
                 "{\Fgdt; m}%%vm" & vbCrLf & _
                 "{\Fgdt; s}%%vs" & vbCrLf

    '插入点和公差方向
    insertionPoint(0) = 5#: insertionPoint(1) = 5#: insertionPoint(2) = 0#
    direction(0) = 1#: direction(1) = 0#: direction(2) = 0#
```

```
    Set toleranceObj = ThisDrawing.ModelSpace.AddTolerance(textString, insertionPoint.
        direction)
    ZoomExtents
End Sub
```

至于每个公差符号对应的字母({\Fgdt; m}中的 m)的获得,也是通过手工创建所有的公差,然后打开对象特性管理器查看其文字替代值完成,Autodesk 并没有给出这方面的参考资料。

6.3.4 实例效果

在 VBA 集成开发环境中,执行 AddTolerance 宏,能够在图形窗口中得到如图 6.8 所示的结果。

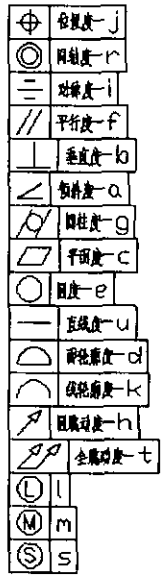


图6.8 列举出所有的公差符号

6.3.5 小结

学习本节的实例,希望读者能够获得下面几个问题的解决办法:

- 获得形位公差中的公差值。
- 按自己的要求修改形位公差。
- 在 VBA 中创建形位公差。

第7章 块和属性

块和属性的基本应用并不复杂，在简单块和属性块的使用两节内容中讨论了这些基础知识，将已经存在的实体添加到块定义中是一个比较热门的问题，匿名块是 AutoCAD 内部实现尺寸标注的机制，可用于解决一些特殊的问题。

块和属性在应用中总是很活跃，这与它设计的灵活性有关，属性和块就能组合出无限多种对象，自动标题栏是其中的一个重要应用。块属性生成表格的程序提取属性块中的属性值，根据这些属性值生成 AutoCAD 中的表格对象（AutoCAD 2005 开始新增的对象）。

实现"斜切"功能应该属于块应用的一个副产品，利用插入块参照时可以指定不同 X、Y、Z 坐标的特点，实现特殊的图形变换。

7.1 简单块的定义和插入

7.1.1 实例说明

所谓简单块，是相对于属性块而言。本节的实例演示简单图块的创建和插入，以及获得当前图形中所有块定义列表的方法。

7.1.2 编程思路

1. 创建块定义的方法

在 VBA 中创建块定义，可以按照下面的步骤进行：

(1) 使用 Blocks 集合的 Add 方法，向图形中添加新的块定义。

(2) 向块定义中添加实体，这与向模型空间中添加实体所用的方法完全一致。

Blocks 集合的 Add 方法定义为：

```
RetVal = object.Add(InsertionPoint, Name)
```

该方法接受两个参数，分别为块定义的基点和名称。

在块定义中添加实体的操作与在模型空间中添加实体的方法完全一致，因为模型空间实际上就是一个特殊的块定义。

2. AutoCAD 中的块

在 AutoCAD 中实际存在三种类型的块：普通块、外部参照块和布局块。

普通块是一组实体的集合，这组实体被关联起来组成了一个对象（块定义）。在图形中，可以插入、缩放、旋转、分解块参照，也可以修改或者重定义块。重定义块之后，图形中所有对应的块参照都会自动更新。

外部参照块实际上是一个从其他图形到当前图形的外部链接，因为外部参照块代表的是一个对图形的链接，而不是几何图形本身，因此外部参照会随着原始图形的修改而更新。

布局块代表的是模型空间和图纸空间中的几何对象，这些块拥有一个同它们关联的布局对象，布局中包含了打印设置和其他的布局定义信息。活动图纸空间的块名称一直都是 * PAPER_SPACE，其他的图纸空间布局对应的块名称是 * PAPER_SPACEn（这里 n 是一个整数）。块名称会从 * PAPER_SPACE0 开始，每次创建一个新的布局 n 的增幅是 1。

默认情况下，图形中会包含三个布局，其对应的布局块名称分别为 * MODEL_SPACE、* PAPER_SPACE 和 * PAPER_SPACE0。

3. 匿名块

AutoCAD 中还包含一种特殊的块，这种块的名称由 AutoCAD 按照一定的规则自动指定，所以称为匿名块。

与普通的块不同，AutoCAD 会在保存一个图形时检查其中的匿名块是否被引用，如果存在未被引用的匿名块，会自动将其从图形中删除。

7.1.3 步骤

（1）在 VBA 集成开发环境中，创建一个新工程，保存在适当的位置。打开 ThisDrawing 窗口，在其中添加创建块的函数 CreateBlkDef，其实现代码为：

```
Public Sub CreateBlkDef( )
    Dim ptBase(0 To 2) As Double
    ptBase(0) = 0: ptBase(1) = 0: ptBase(2) = 0

    '添加块定义
    Dim objBlkDef As AcadBlock
    Set objBlkDef = ThisDrawing.Blocks.Add(ptBase, "普通块")

    '向块定义中添加图形对象
    Dim pt1(0 To 2) As Double
    Dim pt2(0 To 2) As Double
    pt1(0) = -10: pt1(1) = 0: pt1(2) = 0
    pt2(0) = 10: pt2(1) = 0: pt2(2) = 0
    ThisDrawing.Blocks.Item("普通块").AddLine pt1, pt2

    pt1(0) = 0: pt1(1) = -10: pt1(2) = 0
    pt2(0) = 0: pt2(1) = 10: pt2(2) = 0
    ThisDrawing.Blocks.Item("普通块").AddLine pt1, pt2

    ThisDrawing.Blocks.Item("普通块").AddCircle ptBase, 6
End Sub
```

创建块定义时，必须首先向当前图形的块集合中添加一个块定义，然后向块中添加组成块的实体。

(2) 向当前程序中添加一个窗体，窗体的标题修改为“普通块的定义和插入”，窗体名称为 frmMain。向窗体中添加如图 7.1 所示的控件，并将组合框命名为 cboBlkDefs，两个命令按钮命名为 cmdInsert 和 cmdExit。

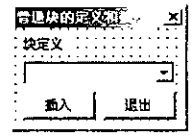


图 7.1　设计窗体

(3) 在 frmMain 窗体的初始化事件中，将当前图形中所有的块定义名称添加到列表框中，并将列表框中的第一个元素设置为当前选择的元素，其实现代码为：

```
Private Sub UserForm_Initialize( )
    '将当前图形中定义的所有块定义名称添加到组合框中
    Dim objBlkDef As AcadBlock
    For Each objBlkDef In ThisDrawing.Blocks
        '不将模型空间、图纸空间和匿名块添加到列表中
        If StrComp(Left(objBlkDef.Name, 1), "*") <> 0 Then
            cboBlkDefs.AddItem objBlkDef.Name
        End If
    Next objBlkDef

    '将列表框的第一个元素设置为被选择的元素
    If cboBlkDefs.ListCount > 0 Then
        cboBlkDefs.ListIndex = 0
    End If
End Sub
```

由前面的知识可以知道，布局块和匿名块的一个共同特征是以 * 作为名称的首字母，因此可以对块定义的名称首字母进行判断，从而不将布局块和匿名块的名称添加到列表框中。

(4) 在“插入”按钮的单击事件中，可以根据用户在列表框中的选择插入指定的块参照，其实现代码为：

```
Private Sub cmdInsert_Click( )
    '如果当前能组合框中没有元素
    If cboBlkDefs.ListCount < 1 Then
        MsgBox "图形中不包含任何的块定义!", vbCritical
        Exit Sub
    End If

    '提示用户拾取插入点
    Dim ptInsert As Variant
    frmMain.Hide
    ptInsert = ThisDrawing.Utility.GetPoint(, "拾取插入点：")

    ThisDrawing.ModelSpace.InsertBlock ptInsert, cboBlkDefs.Text, 1, 1, 1, 0
    ThisDrawing.Regen acActiveViewport

    frmMain.Show
End Sub
```

需要注意的是，在提示用户输入点时，AutoCAD 必须拥有输入焦点，因此在用户拾取点之前隐藏窗体(Hide 方法)，而在完成拾取之后显示窗体(Show 方法)。

(5) "退出"按钮用于退出当前程序，其实现代码为：

```
Private Sub cmdExit_Click( )
    End
End Sub
```

(6) 在 ThisDrawing 窗口中，添加用于启动 frmMain 窗体的宏 ShowAllBlkDefs，其实现代码为：

```
Public Sub ShowAllBlkDefs( )
    frmMain.Show
End Sub
```

在 VBA 中，这是典型的在宏中启动窗体的方法，以后的程序中还会多次遇到。

7.1.4 实例效果

(1) 在 VBA 集成开发环境中，将光标定位在 CreateBlkDef 宏内部，按 F5 键运行宏，选择 AutoCAD 中的"插入"|"块"菜单项，系统会弹出如图 7.2 所示的"插入"对话框，在列表中已经包含了块定义的名称"普通块"，也就是 CreateBlkDef 宏所创建的块。

(2) 使用 BLOCK 命令，在图形中创建两个块"树"和"电线杆"。运行 ShowAllBlkDefs 宏，得到如图 7.3 所示的结果。在块定义列表中，已经包含了前面创建的 3 个块。

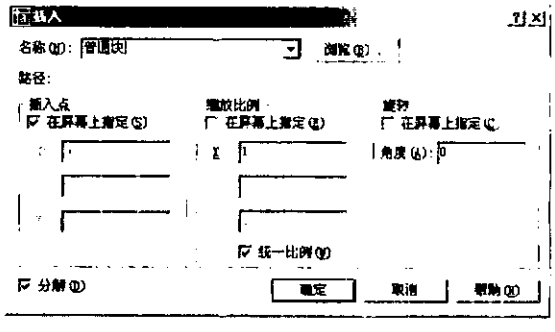


图 7.2 图形中已经包含"普通块"

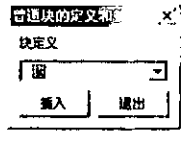


图 7.3 显示当前图形的块列表

(3) 从块列表中选择"普通块"，单击"插入"按钮，在图形窗口中拾取一点作为块参照的插入点，然后单击"退出"按钮退出程序，就能在图形窗口中添加一个块参照，其对应的块定义是"普通块"。

7.1.5 小结

ModelSpace 是一个集合对象，它实际上是一个布局块，它与普通块有一些区别：

- 布局块无法插入到图形中。
- 布局块无法通过 GetBoundingBox 方法来获得其最小包围框，而只能获得该空间中每一个对象的最小包围框，然后对其进行组合得到布局的最小包围框。

7.2　将已经存在的实体添加到块定义中

7.2.1　实例说明

使用前面介绍的方法，在创建块定义时只能创建新的实体，而在 AutoCAD 中用户可以通过选择已经存在的实体来定义块，本节的实例同样实现这个功能，此外，还用 SendCommand 方法模拟了动态插入块参照。

7.2.2　编程思路

1. CopyObjects 方法

ActiveX 模型中的 CopyObjects 方法能够复制多个对象，并可以指定复制对象的目的容器，这为将已有实体添加到块定义中、在不同图形之间复制实体提供了可能。CopyObjects 方法定义为：

```
RetVal = object.CopyObjects(Objects[, Owner][, IDPairs])
```

其中，object 代表方法的调用者，一般可以是文档（例如 ThisDrawing）或者数据库对象；Objects 是一个 Object 类型的数组，包含了所要复制的对象（不一定是实体，还可能是其他类型的对象）；Owner 是复制对象的目标容器，例如图形的模型空间、块定义等；IDPairs 是一个 IDPair 对象的数组，保存了复制及传送过程中所发生的信息；RetVal 为返回值，在一个数组中保存了新创建的对象。

2. 模拟 INSERT 命令的效果

应该说，单纯使用 VBA 的内部功能，无法实现在插入图块过程中动态选择插入点的效果，本节的实例仅仅是使用 SendCommand 方法对此进行模拟而已。

VBA 中的 SendCommand 方法类似于 AutoLISP 中的 Command 函数，但是实际上 SendCommand 方法功能要少得多。首先，无法向 SendCommand 传递参数（本书中已经通过 AutoLISP 解决）；其次，在某些情况下会导致运行顺序的混乱（本书中通过单独将 SendCommand 后面的代码封装解决）；最后，无法在 SendCommand 方法中和用户交互。

如果在 AutoCAD 命令行中输入下面的语句：

```
(command "line" pause pause "")
```

按下 Enter 键执行后，可以模拟 LINE 命令创建直线的效果。

那么，在 AutoCAD 命令行输入下面的语句：

```
(command "-insert" "BlockName" pause "" "" "")
```

按下 Enter 键执行后，可模拟 INSERT 命令插入块参照的效果。

7.2.3　步骤

(1) 在 AutoCAD 2005 中，打开 VBA 工程管理器，创建一个新工程，保存在适当的位置。进入 VBA 集成开发环境，双击“工程资源管理器”窗口的 ThisDrawing 图标，打开其代

码窗口，创建一个函数 HasBlkDef，用于判断当前图形中是否包含指定名称的块定义，其实现代码为：

```
Private Function HasBlkDef(ByVal blkName As String) As Boolean
    On Error Resume Next

    Dim objBlkDef As AcadBlock
    Set objBlkDef = ThisDrawing.Blocks.Item(blkName)

    '利用错误机制来判断是否包含指定的块定义
    If Err Then
        HasBlkDef = False
    Else
        HasBlkDef = True
    End If
End Function
```

(2) 在 ThisDrawing 模块的代码窗口中，创建一个函数 DeleteBlkDef，用于从图形中删除一个块定义，其实现代码为：

```
Private Sub DeleteBlkDef(ByVal blkName As String)
    '删除该块在图形中所有的块参照
    Dim objBlkRef As AcadBlockReference
    For Each objBlkRef In ThisDrawing.ModelSpace
        If StrComp(objBlkRef.Name, blkName) = 0 Then
            objBlkRef.Delete
        End If
    Next objBlkRef

    '删除块定义
    ThisDrawing.Blocks.Item(blkName).Delete
End Sub
```

需要注意的是，在从块集合中删除某个块定义之前，必须从图形中将其所有的块参照删除。

(3) 创建一个宏 CreateBlkDef，用于提示用户选择已有的实体，并将这些实体添加到块定义中，其实现代码为：

```
Public Sub CreateBlkDef( )
    '安全创建选择集
    On Error Resume Next
    Dim SSet As AcadSelectionSet
    If Not IsNull(ThisDrawing.SelectionSets.Item("Example")) Then
        Set SSet = ThisDrawing.SelectionSets.Item("Example")
        SSet.Delete
    End If
    Set SSet = ThisDrawing.SelectionSets.Add("Example")
```

```
    '提示用户选择对象
    SSet.SelectOnScreen

    '选择基点
    Dim ptBase As Variant
    ptBase = ThisDrawing.Utility.GetPoint(, "拾取基点: ")

    '创建块定义
    Dim objBlkDef As AcadBlock
    If HasBlkDef("SelectionSet") Then
        Call DeleteBlkDef("SelectionSet")
    End If
    Set objBlkDef = ThisDrawing.Blocks.Add(ptBase, "SelectionSet")

    '将选择集中的实体添加到数组中
    Dim objCollection( ) As Object
    ReDim objCollection(SSet.Count - 1)
    Dim i As Integer
    For i = 0 To SSet.Count - 1
        Set objCollection(i) = SSet.Item(i)
    Next i

    '将数组中的实体复制到块定义中
    ThisDrawing.CopyObjects objCollection, objBlkDef

    SSet.Delete          '及时删除不用的选择集非常重要
End Sub
```

(4) 创建 InsertBlock 宏,用于模拟 INSERT 命令插入块参照的效果,其实现代码为:

```
Public Sub InsertBlock( )
    '提示用户输入块定义的名称
    Dim blkName As String
    blkName = ThisDrawing.Utility.GetString(False, "输入块参照的名称: ")

    '插入块参照
    If HasBlkDef(blkName) Then
        Dim strLeft As String, strRight As String
        strLeft = "(command "" - insert"" """
        strRight = """ pause """" """" """") "
        '完整语句示例: (command " - insert" "SelectionSet" pause "" "" "")
        ThisDrawing.SendCommand strLeft & blkName & strRight
    End If
End Sub
```

上面的字符串看起来比较零乱,是有原因的。在 VBA 中,如果要创建一个包含引号的

字符串，例如 Say "Hello"，可以用下面的语句（两个引号来表示字符串内部的引号）：

```
strHello = "Say ""Hello"""
```



图 7.4 SendCommand 发送的字符串

可以在 SendCommand 语句之前添加下面的语句来查看 SendCommand 所发送的内容：

```
MsgBox strLeft & blkName & strRight
```

此语句执行的结果如图 7.4 所示。

7.2.4 实例效果

（1）在 AutoCAD 2005 中，创建几个实体。进入 VBA 集成开发环境，执行 CreateBlkDef 宏，按照系统提示依次选择所要组成块定义的实体、拾取基点，完成块的定义。

（2）执行 InsertBlock 宏，在命令行输入 SelectionSet 响应“输入块参照的名称：”提示，此时可以自由移动块参照选择插入点。单击指定插入点后，程序自动退出。

7.2.5 小结

前面已经介绍，使用 CopyObjects 方法可以将对象复制到外部图形中，因此可以用此方法实现在多个图形之间复制数据。

下面的宏演示了如何向外部图形中复制对象：

```
Public Sub CopyFromOuterDwg( )
    '判断图形是否存在
    If Len(Dir("C: \test.dwg")) = 0 Then
        MsgBox "指定的图形不存在!", vbCritical
        Exit Sub
    End If

    '保存目前的文档
    Dim objCurDoc As AcadDocument
    Set objCurDoc = ThisDrawing.Application.ActiveDocument

    '打开一个新图形
    Dim objNewDoc As AcadDocument
    Set objNewDoc = ThisDrawing.Application.Documents.Open("C: \test.dwg")

    '将当前图形(实际上是新打开的图形 test.dwg)的实体复制到其他图形
    Dim objCollection(0 To 1) As Object
    Set objCollection(0) = objNewDoc.ModelSpace.Item(0)
    Set objCollection(1) = objNewDoc.ModelSpace.Item(1)
    objNewDoc.CopyObjects objCollection, objCurDoc.ModelSpace

    '关闭打开的图形
    objNewDoc.Close
End Sub
```

7.3　使用匿名块

7.3.1　实例说明

匿名块是一种特殊的块，用户无法手动清理这种块，也无法手工向图形中插入这种块的参照。与普通块相比，匿名块的最大特征是对用户的隐蔽和自动清理，也就是说在使用 AutoCAD 的过程中，用户无法直接操作匿名块（实际上匿名块仅是一种内部的实现机制），并且 AutoCAD 会在关闭（或者打开）图形时自动清除未在图形中使用的匿名块。

在 AutoCAD 中，尺寸标注对象就是匿名块，在本节中将用一个宏来验证。

本节的实例演示了匿名块的创建、获得图形中创建的最后一个匿名块、插入匿名块，以及统计当前图形中匿名块的数量。

7.3.2　编程思路

创建匿名块的方法与普通图块类似，惟一的不同是指定块名称。匿名块的名称是由 AutoCAD 自己来确定的，用户在创建匿名块时需要将其名称设置为 * U（这里的 U 可以是 26 个英文字母的任何一个，大小写均可）。

AutoCAD 会自动以递增编号的顺序来给图形中的匿名块命名，要获得当前图形中最后创建的匿名块，可以通过判断匿名块的名称中的编号（ * U 后面的部分）来实现，编号最大的匿名块就是最后创建的。

7.3.3　步骤

（1）在 VBA 集成开发环境中，创建一个新工程，保存在适当的位置。打开 ThisDrawing 窗口，创建一个宏 CreateAnonymousBlk，用于在当前图形中创建一个匿名块，其实现代码为：

```
Public Sub CreateAnonymousBlk( )
    Dim ptBase(0 To 2) As Double
    ptBase(0) = 0: ptBase(1) = 0: ptBase(2) = 0

    '添加块定义
    Dim objBlkDef As AcadBlock
    Set objBlkDef = ThisDrawing.Blocks.Add(ptBase, " * U")

    '向块定义中添加图形对象
    Dim pt1(0 To 2) As Double
    Dim pt2(0 To 2) As Double
    pt1(0) = - 10: pt1(1) = 0: pt1(2) = 0
    pt2(0) = 10: pt2(1) = 0: pt2(2) = 0
    objBlkDef.AddLine pt1, pt2

    pt1(0) = 0: pt1(1) = - 10: pt1(2) = 0
```

```
    pt2(0) = 0: pt2(1) = 10: pt2(2) = 0
    objBlkDef.AddLine pt1, pt2

    objBlkDef.AddCircle ptBase, 6
End Sub
```

(2) 创建一个宏 GetLastAnonymousBlk,用于获得当前图形中创建的最后一个匿名块,其实现代码为:

```
Public Function GetLastAnonymousBlk( ) As AcadBlock
    Dim objBlkDef As AcadBlock
    Dim n As Integer

    For Each objBlkDef In ThisDrawing.Blocks
        '匿名块以 * 为起始字符
        If Left(objBlkDef.Name, 1) = "*" Then
            '消除布局块的影响
            If objBlkDef.Name <> "*Model_Space" And Left(objBlkDef.Name, 12) <> "*
                Paper_Space" Then
                '返回名称编号最大的一个块
                If Mid(objBlkDef.Name, 3) >= n Then
                    n = Mid(objBlkDef.Name, 3)
                    Set GetLastAnonymousBlk = objBlkDef
                End If
            End If
        End If
    Next

    Set objBlkDef = Nothing
End Function
```

在上面的代码中,判断最后创建的匿名块有三个条件:

- 名称的第一个字符是 * 。
- 不是布局块。
- 名称中的编号最大。

在判断名称中的编号时,逐个将当前块的编号与前面所有块编号的最大值比较,返回具有较大编号的匿名块,最终就得到当前图形中编号最大的匿名块。

(3) 创建一个宏 InsertAnonymousBlkRef,用于将最后创建的匿名块插入到当前图形中,其实现代码为:

```
Public Sub InsertAnonymousBlkRef( )
    Dim ptInsert(0 To 2) As Double
    ptInsert(0) = 100: ptInsert(1) = 100: ptInsert(2) = 0

    Dim objBlk As AcadBlock
    '获得图形中最后一个创建的匿名块
```

```
    Set objBlk = GetLastAnonymousBlk
    ThisDrawing.ModelSpace.InsertBlock ptInsert, objBlk.Name, 1, 1, 1, 0
End Sub
```

从代码中可以看出，插入匿名块的操作和插入普通块的操作是相同的。

(4) 创建一个宏 GetAnonymousBlkNumber，获得当前图形中所有匿名块的数量和名称，其实现代码为：

```
Public Sub GetAnonymousBlkNumber( )
    Dim objBlkDef As AcadBlock
    Dim n As Integer                    '匿名块的数量

    For Each objBlkDef In ThisDrawing.Blocks
        '匿名块以 * 为起始字符
        If Left(objBlkDef.Name, 1) = "*" Then
            '消除布局块的影响
            If objBlkDef.Name <> "*Model_Space" And Left(objBlkDef.Name, 12) <> "*
                Paper_Space" Then
                n = n + 1
                Debug.Print objBlkDef.Name
            End If
        End If
    Next

    MsgBox "当前图形中匿名块的数量是：" & CStr(n)
End Sub
```

CStr 函数能将指定的表达式转换成字符串变量。

7.3.4 实例效果

(1) 在 AutoCAD 2005 中，创建一个新图形，执行 GetAnonymousBlkNumber，会显示当前图形中没有匿名块，如图 7.5 所示。

(2) 执行宏 CreateAnonymousBlk，在图形中定义一个新的匿名块，此时执行 GetAnonymousBlkNumber 宏可发现，图形中匿名块的数量是 1。执行 InsertAnonymousBlkRef 宏，将最后创建的匿名块插入到当前图形中，如图 7.6 所示。



图 7.5　提示图形中不包含匿名块

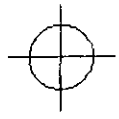


图 7.6　插入匿名块的结果

(3) 向图形中添加两个尺寸标注，然后执行 GetAnonymousBlkNumber 宏，可以发现图形中匿名块的数量是 3，并且能够在立即窗口中获得其名称，如图 7.7 所示。

这个结果也证明了尺寸标注确实是匿名块。

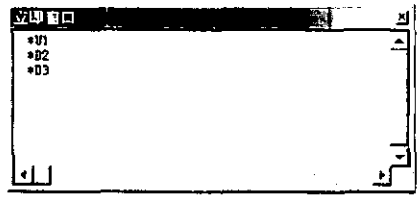


图 7.7 查看当前图形中所有匿名块的名称

7.3.5 小结

学习本节内容之后，需要掌握匿名块如下的几个特点：

- 匿名块对用户是隐蔽的。
- 未被参照的匿名块会在 AutoCAD 关闭(或打开)图形时被自动清除。
- AutoCAD 会以递增的顺序来管理匿名块的编号。

7.4 属性块的使用

7.4.1 实例说明

块通常和属性结合起来使用，也就是所谓的属性块。本节的实例演示了属性块的创建和插入。

7.4.2 编程思路

创建属性块的方法与普通块类似，不同之处是要在块定义中添加几个属性定义，可以通过 Block 对象的 AddAttribute 方法来实现，该方法定义为：

```
RetVal = object.AddAttribute(Height, Mode, Prompt, InsertionPoint, Tag, Value)
```

Height 是属性文本的高度；Mode 是属性的模式，可以设置为不可见、固定、验证和预置，这与 AutoCAD 中手工创建属性定义一致；Prompt 是属性的提示文字；InsertionPoint 指定属性文字的插入点；Tag 是属性的标记文字；Value 是属性的默认值；RetVal 返回新创建的属性对象。

插入属性块同样采用 InsertBlock 方法，在插入块参照之后，需要使用块参照对象的 GetAttributes 方法获得其中的属性，对属性值进行修改。

7.4.3 步骤

(1) 在 VBA 集成开发环境中，创建一个新工程，保存在适当的位置。打开 ThisDrawing 窗口，创建一个宏 CreateAttBlk，用于创建一个带有属性的块定义，其实现代码为：

```
Public Sub CreateAttBlk( )
    Dim ptBase(0 To 2) As Double
```

```
    ptBase(0) = 0: ptBase(1) = 0: ptBase(2) = 0

    '添加块定义
    Dim objBlkDef As AcadBlock
    Set objBlkDef = ThisDrawing.Blocks.Add(ptBase, "属性块")

    '向块定义中添加图形对象
    Dim pt1(0 To 2) As Double
    Dim pt2(0 To 2) As Double
    pt1(0) = -10: pt1(1) = 0: pt1(2) = 0
    pt2(0) = 10: pt2(1) = 0: pt2(2) = 0
    objBlkDef.AddLine pt1, pt2

    pt1(0) = 0: pt1(1) = -10: pt1(2) = 0
    pt2(0) = 0: pt2(1) = 10: pt2(2) = 0
    objBlkDef.AddLine pt1, pt2

    objBlkDef.AddCircle ptBase, 6

    objBlkDef.AddAttribute 5, 0, "输入插入点坐标:", ptBase, "插入点坐标", CStr("(100,
100)")
End Sub
```

与创建普通块相比,代码中惟一不同的一句是:

```
objBlkDef.AddAttribute 5, 0, "输入插入点坐标:", ptBase, "插入点坐标", CStr("(100, 100)")
```

该语句的作用是向块定义中添加一个属性定义。

(2) 创建一个宏 InsertAttBlk,向图形窗口中插入"属性块"的块参照,其实现代码为:

```
Public Sub InsertAttBlk( )
    '插入块
    Dim objBlkRef As AcadBlockReference
    Dim ptInsert(0 To 2) As Double
    ptInsert(0) = 100: ptInsert(1) = 100: ptInsert(2) = 0
    Set objBlkRef = ThisDrawing.ModelSpace.InsertBlock(ptInsert, "属性块", 1, 1.1, 0)
    ZoomAll

    '获得块参照的属性
    Dim varAttributes As Variant
    varAttributes = objBlkRef.GetAttributes
    varAttributes(0).TextString = "(200, 200)"
End Sub
```

插入块参照之后,使用 GetAttributes 方法获得块参照中的属性,并且对属性值进行修改。如果块参照中包含多个属性,可以使用下面的语句修改属性值。

```
varAttributes(n).TextString = "新的属性值"
```

其中，n 是所要修改的属性的索引值。

7.4.4 实例效果

在 VBA 集成开发环境中，首先执行 CreateAttBlk 宏，创建带有属性的块定义“属性块”，然后执行 InsertAttBlk 宏，向图形中插入一个块参照，得到如图 7.8 所示的结果。

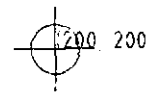


图 7.8 插入块参照的结果

7.4.5 小结

本节只是简单介绍了属性块的创建和插入，下一节将要结合提取属性的知识，创建一个自动标题栏的程序。

7.5 自动标题栏

7.5.1 实例说明

本节的程序从《AutoCAD VBA 精彩实例教程》的 4.3 节发展而来，将块和属性结合使用，创建可被用户自由编辑的标题栏，如图 7.9 所示。

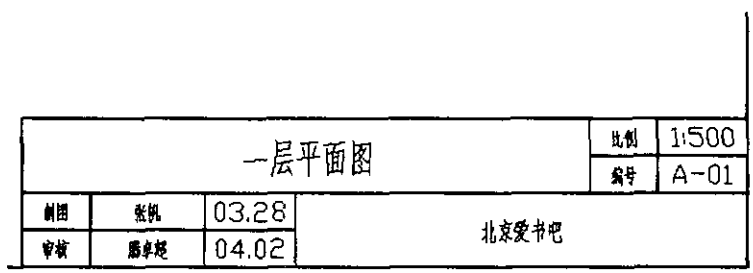


图 7.9 创建标题栏和图框的效果

7.5.2 编程思路

本节的程序允许用户创建和修改标题栏，整个标题栏被定义成一个块，其中可被修改的文字都是块属性，例如图形名称、设计单位、设计者等信息。

1. 创建标题栏的块定义

在标题栏的块定义中，包含了标题栏和图框的直线、文字和属性。

创建图框和标题栏的分隔线时，可采用相对定位的方法分别确定各个关键点的位置（如图 7.10 所示），为此编写了一个函数 GetPoint，该函数能够获得相对已知点偏移一定距离的点，其定义为：

```
Public Function GetPoint(pt As Variant, x As Double, y As Double) As Variant
```

其中，pt 是已知点；x 为所要计算的点在 X 轴方向上的偏移量；y 是所要计算的点在 Y

轴方向上的偏移量。要获得点 pt(1)右边 20 个单位、上边 10 个单位的点，就可以用下面的语句：

```
pt(2) = GetPoint(pt(1), 20, 10)
```

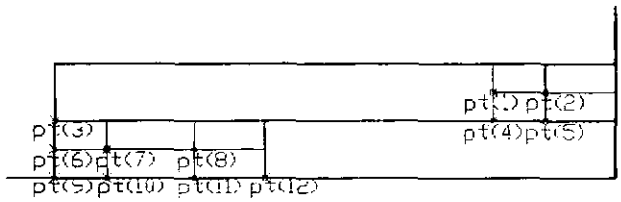


图 7.10　关键点的位置(关键点均为文字框的左下角点)

为什么要用相对定位来确定关键点的位置？因为各个关键点之间的相对坐标容易确定，而绝对坐标必须经过计算才能确定。

确定关键点位置之后，就可以使用“相对绘图法”来绘制标题栏内部的直线，为此定义了一个函数 AddLWPlineSegRela，根据一个基点和另外一点的相对位置绘制多段线，其定义为：

```
Public Function AddLWPlineSegRela(ByVal ptSt As Variant, ByVal x As Double, ByVal y As Double,
    Optional width As Double = 0) As AcadLWPolyline
```

其中，ptSt 是起点；x 是终点相对于起点在 x 轴方向的偏移；y 是终点相对于起点在 y 轴方向的偏移；width 是多段线的线宽。例如，要创建以关键点 pt(3)为起点的水平线，就可以使用下面的语句：

```
AddLWPlineSegRela pt(3), 160, 0, 0.2
```

为了使文字和属性正好分布在标题栏内矩形的中心，创建了两个函数 DrawTextRec 和 DrawAttributeRec，根据矩形左下角点、文字(属性)内容、文字(属性)高度、矩形的高度和宽度来创建文字(属性)，确保文字放置在矩形的中心。

2. 属性的提取和编辑

要提取和编辑标题栏块参照的属性，都必须首先获得标题栏块参照，然后使用 GetAttributes 方法获得块参照的属性集合，通过该集合中的元素获得或编辑属性值。

需要注意的一点是，属性集合中各元素的索引值与创建块定义时添加属性的顺序是相同的。例如，在创建块定义时，添加的第一个属性是图形比例，那么属性集合中第一个元素 varAttributes(0)代表的就是图形比例属性。

3. 创建模式和修改模式

创建标题栏和修改标题栏都用一个对话框来完成。判断当前的操作是创建还是修改标题栏的依据，就是图形中是否已经存在“标题栏”块参照，如果已经存在，那么当前就是在编辑模式。编辑模式和创建模式用一个布尔类型的全局变量 createMode 来标识，每次执行程序都要判断一下工作模式，根据工作模式来确定不同的操作。

在创建模式中，对话框的标题是“创建标题栏”，要完成创建“标题栏”块定义的创建和对应的块参照的插入；在修改模式中，对话框的标题是“修改标题栏”，要获得当前标题栏中属性值并显示在对话框中，并能根据用户修改的结果更新块参照的属性值。

判断工作模式的方法是，检查当前图形中是否存在名称为“标题栏”的块参照，如果存在，表示当前是修改模式，否则是创建模式。

7.5.3 步骤

(1) 在 VBA 集成开发环境中，创建一个新工程，保存在适当的位置。选择“插入”|“用户窗体”菜单项，向当前的工程中添加一个对话框，在其中放置如图 7.11 所示的控件。

其中，各个文本框的名称分别修改为 txtDwgName、txtDwgScale、txtDwgSerial、txtDesigner、txtDesignDate、txtAudit、txtAuditDate 和 txtDesignCorp，两个命令按钮的名称分别修改为 cmdOk 和 cmdCancel。

(2) 选择“插入”|“模块”菜单项，向工程中添加一个标准模块，在属性窗口中将其名称修改为 mdlMain，在其中添加启动宏 AutoTitleBar 的定义和全局变量 createMode 的声明：

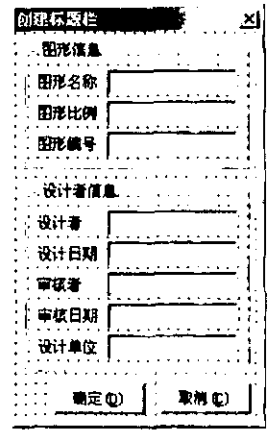


图 7.11 在窗体中添加控件

```
Public createMode As Boolean                '创建模式或修改模式…

Public Sub AutoTitleBar( )
    '判断当前的工作模式
    createMode = True
    Dim objBlkRef As AcadBlockReference
    For Each objBlkRef In ThisDrawing.ModelSpace
        If StrComp(objBlkRef.name, "标题栏") = 0 Then
            createMode = False
            Exit For
        End If
    Next objBlkRef

    frmMain.Show
End Sub
```

从上面的代码中可以看出，判断工作模式的依据就是，在当前图形中是否存在名称为“标题栏”的块参照。

(3) 向工程中添加一个新的标准模块，命名为 mdlCalculate，在其中添加一个函数，用于获得相对已知点偏移一定距离的点，其实现代码为：

```
'参数说明。pt：已知点；x：所要计算的点在 X 轴方向上的
'偏移量；y：所要计算的点在 Y 轴方向上的偏移量
Public Function GetPoint(pt As Variant, x As Double, y As Double) As Variant
```

```
    Dim ptTarget(0 To 2) As Double

    ptTarget(0) = pt(0) + x
    ptTarget(1) = pt(1) + y
    ptTarget(2) = 0

    GetPoint = ptTarget
End Function
```

(4) 在 mdlCalculate 模块中添加两个函数 MinDouble 和 MaxDouble，用于判断两个 Double 类型变量的大小，并返回较大(小)值，其相关代码为：

```
'返回两个 Double 类型变量的最小值
Public Function MinDouble(ByVal A As Double, ByVal b As Double) As Double
    If A > b Then
        MinDouble = b
    Else
        MinDouble = A
    End If
End Function

'返回两个 Double 类型变量的最大值
Public Function MaxDouble(ByVal A As Double, ByVal b As Double) As Double
    If A > b Then
        MaxDouble = A
    Else
        MaxDouble = b
    End If
End Function
```

(5) 向工程中添加一个新的标准模块，命名为 mdlEntity，在其中放置创建基本对象的函数。在其中添加一个函数 AddLWPlineSeg，用于创建仅包含一段直线的多段线，其实现代码为：

```
'参数说明。ptSt：起点；ptEn：终点；width：线宽
Public Function AddLWPlineSeg(ByVal ptSt As Variant, ByVal ptEn As Variant, Optional width As
Double = 0) As AcadLWPolyline
    Dim objPline As AcadLWPolyline
    Dim ptArr(0 To 3) As Double

    ptArr(0) = ptSt(0)
    ptArr(1) = ptSt(1)
    ptArr(2) = ptEn(0)
    ptArr(3) = ptEn(1)

    Set objPline = ThisDrawing.Blocks.Item("标题栏").AddLightWeightPolyline(ptArr)
    objPline.ConstantWidth = width
```

```
    objPline.Update
    Set AddLWPlineSeg = objPline
End Function
```

(6) 在 mdlEntity 模块中添加一个函数 AddLWPlineSegRela，根据一个基点和另外一点的相对位置绘制多段线，其实现代码为：

```
'参数说明。ptSt：起点；x：终点相对于起点在 x 轴方向的偏移；y：终点相对于起点在 y 轴方向
'的偏移；width：线宽
Public Function AddLWPlineSegRela(ByVal ptSt As Variant, ByVal x As Double, ByVal y As Double, _
    Optional width As Double = 0) As AcadLWPolyline
    Dim ptEn As Variant
    ptEn = GetPoint(ptSt, x, y)

    Set AddLWPlineSegRela = AddLWPlineSeg(ptSt, ptEn, width)
End Function
```

(7) 在 mdlEntity 模块中添加一个函数 AddRectangle，根据指定的两个角点和线宽创建矩形，其实现代码为：

```
'参数说明。pt1：一个角点；pt2：另一个角点；width：线宽
Public Function AddRectangle(ByVal pt1 As Variant, ByVal pt2 As Variant, Optional width As
    Double = 0) As AcadLWPolyline
    Dim ptArr(7) As Double
    Dim objPline As AcadLWPolyline

    '错误处理
    If Abs(pt1(0) - pt2(0)) < 0.000001 Or Abs(pt1(1) - pt2(1)) < 0.000001 Then
        MsgBox "创建矩形失败!"
        Exit Function
    End If

    ptArr(0) = MinDouble(pt1(0), pt2(0)): ptArr(1) = MaxDouble(pt1(1), pt2(1))
    ptArr(2) = MinDouble(pt1(0), pt2(0)): ptArr(3) = MinDouble(pt1(1), pt2(1))
    ptArr(4) = MaxDouble(pt1(0), pt2(0)): ptArr(5) = MinDouble(pt1(1), pt2(1))
    ptArr(6) = MaxDouble(pt1(0), pt2(0)): ptArr(7) = MaxDouble(pt1(1), pt2(1))

    Set objPline = ThisDrawing.Blocks.Item("标题栏").AddLightWeightPolyline(ptArr)
    objPline.ConstantWidth = width
    objPline.Closed = True
    Set AddRectangle = objPline
End Function
```

(8) 在窗体初始化时(Initialize 事件)，根据当前工作模式来设置窗体的标题栏，或者进行其他的初始化工作，其相关代码为：

```
Private Sub UserForm_Initialize( )
    '根据工作模式修改对话框的标题
```

```
    If mdlMain.createMode = True Then
        frmMain.Caption = "创建标题栏"
    Else
        frmMain.Caption = "修改标题栏"
    End If

    '将图形中的属性值显示在对应的文本框中
    If mdlMain.createMode = False Then
        '获得图形中的标题栏块参照
        Dim objBlkRef As AcadBlockReference
        Set objBlkRef = GetTitleBar

        '显示属性值
        Dim varAttributes As Variant
        varAttributes = objBlkRef.GetAttributes
        txtDwgScale.text = varAttributes(0).TextString
        txtDwgName.text = varAttributes(1).TextString
        txtDwgSerial.text = varAttributes(2).TextString
        txtDesigner.text = varAttributes(3).TextString
        txtDesignDate.text = varAttributes(4).TextString
        txtAudit.text = varAttributes(5).TextString
        txtAuditDate.text = varAttributes(6).TextString
        txtDesignCorp.text = varAttributes(7).TextString
    End If
End Sub
```

其中，GetTitleBar 是一个自定义函数，用于获得图形中的标题栏块参照，其实现代码为：

```
Private Function GetTitleBar( ) As AcadBlockReference
    Dim blkRef As AcadBlockReference

    For Each blkRef In ThisDrawing.ModelSpace
        If StrComp(blkRef.name, "标题栏") = 0 Then
            Set GetTitleBar = blkRef
            Exit Function
        End If
    Next blkRef
End Function
```

从代码中可以看出，该函数会返回图形窗口中找到的第一个“标题栏”块参照，找到块参照之后，直接使用 Exit Function 语句来结束函数的执行。

(9) 在“确定”按钮的单击事件中，根据当前的工作模式来指定适当的处理方式。如果当前处在创建模式，就创建标题栏；如果处在修改模式，就根据当前对话框中的参数值对标题栏的属性进行修改，其实现代码为：

```
Private Sub cmdOk_Click( )
```

```
    On Error Resume Next
    '确保文本框不为空
    Dim txt As Control
    For Each txt In frmMain.Controls
        If TypeOf txt Is TextBox Then
            If txt.text = "" Then
                MsgBox "参数不能为空!", vbCritical
                Exit Sub
            End If
        End If
    Next txt

    If createMode = True Then
        '创建标题栏
        Call CreateTitleBar
    Else
        '修改标题栏
        Call ModifyTitleBar
    End If

    End
End Sub
```

CreateTitleBar 和 ModifyTitleBar 是两个自定义过程，分别用于创建和修改标题栏，它们的定义将在下面的步骤中详细介绍。

(10) CreateTitleBar 过程用于根据用户在文本框中输入的参数值，向图形中添加标题栏，其实现代码为：

```
Private Sub CreateTitleBar( )
    '如果图形中已经存在标题栏图块的定义
    If BlkIsExist("标题栏") Then
        GoTo InsertBlock
    End If

    '定义标题栏图块
    Dim objBlkDef As AcadBlock
    Dim ptInsert(0 To 2) As Double
    ptInsert(0) = 0: ptInsert(1) = 0: ptInsert(2) = 0
    Set objBlkDef = ThisDrawing.Blocks.Add(ptInsert, "标题栏")

    '外图框
    Dim ptVer1(0 To 2) As Double, ptVer2 As Variant
    ptVer1(0) = 0: ptVer1(1) = 0: ptVer1(2) = 0
    ptVer2 = GetPoint(ptVer1, 420, 297)
    AddRectangle ptVer1, ptVer2, 0.5
```

```
'计算所有关键点的位置(关键点均为文字框的左下角点)
Dim pt(1 To 12) As Variant
pt(1) = GetPoint(ptVer1, 385, 24)
pt(2) = GetPoint(pt(1), 15, 0)
pt(3) = GetPoint(pt(1), -125, -8)
pt(4) = GetPoint(pt(3), 125, 0)
pt(5) = GetPoint(pt(4), 15, 0)
pt(6) = GetPoint(pt(3), 0, -8)
pt(7) = GetPoint(pt(6), 15, 0)
pt(8) = GetPoint(pt(7), 25, 0)
pt(9) = GetPoint(pt(6), 0, -8)
pt(10) = GetPoint(pt(9), 15, 0)
pt(11) = GetPoint(pt(10), 25, 0)
pt(12) = GetPoint(pt(11), 20, 0)

'绘制内部分隔线
Dim ptTemp1 As Variant
ptTemp1 = GetPoint(pt(9), 0, 32)
'宽度为 0.3
AddLWPlineSegRela ptTemp1, 0, -32, 0.3
AddLWPlineSegRela ptTemp1, 160, 0, 0.3
'宽度为 0.2
'水平
AddLWPlineSegRela pt(1), 35, 0, 0.2
AddLWPlineSegRela pt(3), 160, 0, 0.2
AddLWPlineSegRela pt(6), 60, 0, 0.2
'垂直
AddLWPlineSegRela pt(10), 0, 16, 0.2
AddLWPlineSegRela pt(11), 0, 16, 0.2
AddLWPlineSegRela pt(12), 0, 16, 0.2
AddLWPlineSegRela pt(4), 0, 16, 0.2
AddLWPlineSegRela pt(5), 0, 16, 0.2

'添加图框文字和属性
Dim objText As AcadText
Dim objAttribute As AcadAttribute
Set objText = DrawTextRec(pt(1), "比例", 4, 15, 8)
Set objAttribute = DrawAttributeRec(pt(2), "比例", 4, 20, 8)
Set objAttribute = DrawAttributeRec(pt(3), "图形名称", 8, 125, 16)
Set objText = DrawTextRec(pt(4), "编号", 4, 15, 8)
Set objAttribute = DrawAttributeRec(pt(5), "编号", 4, 20, 8)
Set objText = DrawTextRec(pt(6), "制图", 4, 15, 8)
Set objAttribute = DrawAttributeRec(pt(7), "设计者", 4, 25, 8)
Set objAttribute = DrawAttributeRec(pt(8), "设计日期", 4, 20, 8)
```

```
    Set objText = DrawTextRec(pt(9), "审核", 4, 15, 8)
    Set objAttribute = DrawAttributeRec(pt(10), "审核者", 4, 25, 8)
    Set objAttribute = DrawAttributeRec(pt(11), "审核日期", 4, 20, 8)
    Set objAttribute = DrawAttributeRec(pt(12), "设计单位", 5, 100, 16)

InsertBlock:
    '在图形中插入标题栏
    Dim objBlkRef As AcadBlockReference
    Set objBlkRef = ThisDrawing.ModelSpace.InsertBlock(ptInsert, "标题栏", 1, 1, 1, 0)

    '根据文本框的值修改标题栏的属性
    Call EditAttributes(objBlkRef)
End Sub
```

创建标题栏可以分成几个小步骤来说明：如果当前图形中不存在“标题栏”块定义，就向图形中添加这个块定义(防止块定义中实体的重复添加)；计算关键点的位置；向块定义中添加图框、标题栏分隔线和标题栏的文字(一部分用文字，另一部分用属性)；在图形中插入“标题栏”块参照；根据用户在文本框中输入的参数值修改标题栏的属性值。

上面的代码中包含三个自定义函数 BlkIsExist、DrawTextRec、DrawAttributeRec 和一个自定义过程 EditAttributes，下面的步骤将详细介绍其含义。

(11) 自定义函数 BlkIsExist 用于判断图形中是否存在“标题栏”块定义，其实现代码为：

```
Private Function BlkIsExist(ByVal strBlk As String) As Boolean
    Dim objBlkDef As AcadBlock

    For Each objBlkDef In ThisDrawing.Blocks
        If StrComp(objBlkDef.name, "标题栏") = 0 Then
            BlkIsExist = True
            Exit Function
        End If
    Next objBlkDef

    BlkIsExist = False
End Function
```

(12) 自定义函数 DrawTextRec 在给定的矩形区域内创建文字，并且文字被放置在矩形的中心，其实现代码为：

```
Private Function DrawTextRec(ByVal ptInsert As Variant, ByVal strText As String, ByVal height
    As Double, ByVal widthRec As Double, ByVal heightRec As Double) As AcadText
    Dim objText As AcadText
    Dim ptTemp(0 To 2) As Double
    ptTemp(0) = ptInsert(0) + widthRec/2
    ptTemp(1) = ptInsert(1) + heightRec/2
    ptTemp(2) = ptInsert(2)
```

```
    Set objText = ThisDrawing.Blocks.Item("标题栏").AddText(strText, ptTemp, height)
    '调整文字的对齐方式
    objText.Alignment = acAlignmentMiddleCenter
    objText.TextAlignmentPoint = ptTemp
    objText.Update

    Set DrawTextRec = objText
End Function
```

(13) 自定义函数 DrawAttributeRec 在给定的矩形区域内创建属性定义，并且属性文字被放置在矩形的中心，其实现代码为：

```
Private Function DrawAttributeRec(ByVal ptInsert As Variant, ByVal strText As String, ByVal
    height As Double, ByVal widthRec As Double, ByVal heightRec As Double) As AcadAttribute
    Dim objAttribute As AcadAttribute
    Dim ptTemp(0 To 2) As Double
    ptTemp(0) = ptInsert(0) + widthRec/2
    ptTemp(1) = ptInsert(1) + heightRec/2
    ptTemp(2) = ptInsert(2)

    Set objAttribute = ThisDrawing.Blocks.Item("标题栏").AddAttribute(height, 0, "请输入" &
        strText & ": ", ptTemp, strText, strText)
    '调整文字的对齐方式
    objAttribute.Alignment = acAlignmentMiddleCenter
    objAttribute.TextAlignmentPoint = ptTemp
    objAttribute.Update

    Set DrawAttributeRec = objAttribute
End Function
```

函数 DrawAttributeRec 和 DrawTextRec 的实现基本一致，但是创建属性定义的 AddAttribute 方法需要更多的参数，注意上面的代码中输入的参数情况，尤其是对属性的标记、提示和值的处理。如果输入的 strText 内容为“图形比例”，那么创建的属性定义标记为“图形比例”，提示为“请输入图形比例：”，值为“图形比例”。

如果对属性的标记、提示和值的具体含义有什么疑问，可参考 AutoCAD 帮助系统或者 AutoCAD 应用方面的图书。

(14) 自定义过程 EditAttributes 能够根据用户在对话框中输入的参数值，对标题栏中相应的属性值进行修改，其实现代码为：

```
Private Sub EditAttributes(ByVal objBlkRef As AcadBlockReference)
    Dim varAttributes As Variant
    varAttributes = objBlkRef.GetAttributes
    varAttributes(0).TextString = txtDwgScale.text
    varAttributes(1).TextString = txtDwgName.text
    varAttributes(2).TextString = txtDwgSerial.text
```

```
    varAttributes(3).TextString = txtDesigner.text
    varAttributes(4).TextString = txtDesignDate.text
    varAttributes(5).TextString = txtAudit.text
    varAttributes(6).TextString = txtAuditDate.text
    varAttributes(7).TextString = txtDesignCorp.text
End Sub
```

(15) 在“确定”按钮的单击事件中，调用了另外一个自定义函数 ModifyTitleBar，用于修改标题栏，其实现代码为：

```
Private Sub ModifyTitleBar( )
    '获得标题栏的块参照
    Dim objBlkRef As AcadBlockReference
    Set objBlkRef = GetTitleBar

    '修改标题栏的属性值
    Call EditAttributes(objBlkRef)
End Sub
```

(16) 在“取消”按钮的单击事件中，添加一个 End 语句，用于结束程序的执行：

```
Private Sub cmdCancel_Click( )
    End
End Sub
```

到此为止，程序的编码工作完成。

7.5.4 实例效果

(1) 执行 AutoTitleBar 宏，系统会弹出如图 7.12 所示的对话框。

(2) 在“创建标题栏”对话框中输入标题栏的各种内容，单击“确定”按钮，就能在图形窗口中插入如图 7.13 所示的标题栏。

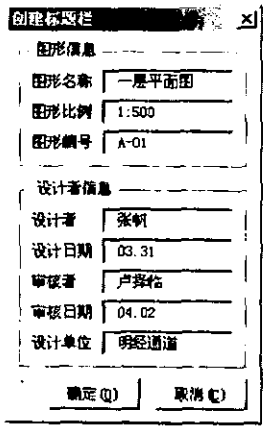


图 7.12 输入标题栏的内容

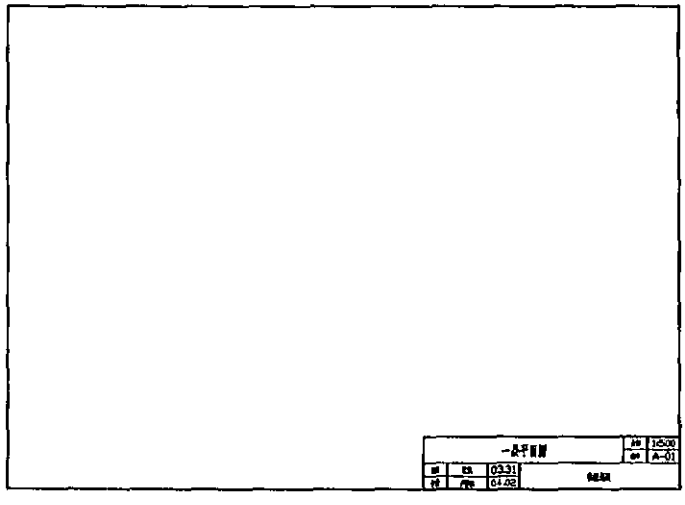


图 7.13 插入标题栏的结果

(3) 再次执行 AutoTitleBar 宏，系统会弹出如图 7.14 所示的“修改标题栏”对话框，并在其中显示标题栏的内容。

(4) 修改某个文本框中的值，单击“确定”按钮，图形中的标题栏就会自动得到更新。

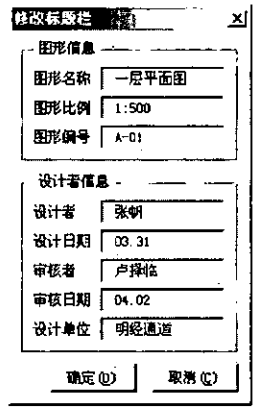


图 7.14　显示标题栏的内容

7.5.5　小结

本节的程序介绍了一个自动标题栏程序的雏形，还有很多地方不够完善，可以在此基础上继续完成如下所示的一些工作。

- 本节的程序仅能创建适合 A3 图纸使用的标题栏，可以扩充使其支持更多的图纸尺寸。
- 动态拾取标题栏的插入点。
- 当用户试图分解标题栏时，给出警告(可以使用对象层事件来实现)。

7.6　AutoCAD 中“斜切”功能的实现

7.6.1　实例说明

在 AutoCAD 中，对图形对象可以进行一定的变形操作，但是并未提供“斜切”的功能，本节的实例模拟这种功能，如图 7.15 所示。

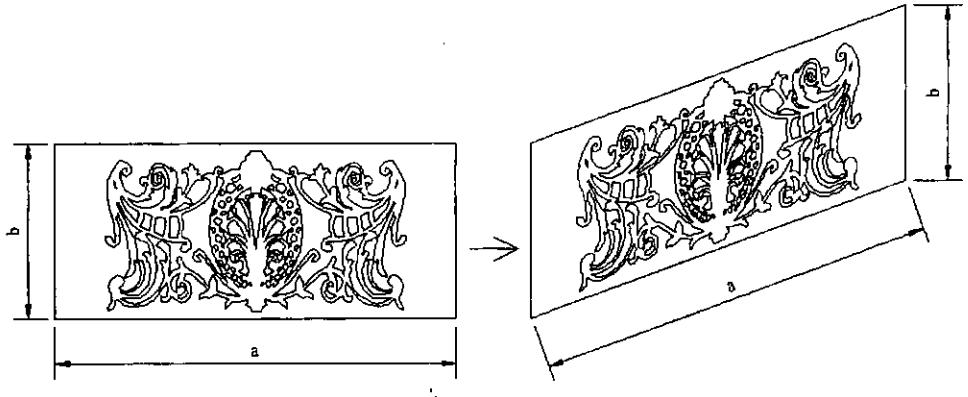


图 7.15　“斜切”的效果

7.6.2　编程思路

为说明问题的方便，在下面的描述中，用长方形代替复杂图形进行分析。

先将图 7.16 中的长方形 ABCD 顺时针旋转 45°，创建成一个块定义，进行不等比的块插入，再对其进行旋转，即可完成“斜切”的效果，如图 7.17 所示。在此过程中，关键是进行相关的参数计算。

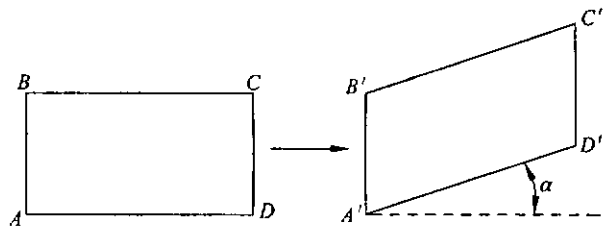


图 7.16 “斜切”效果的分析

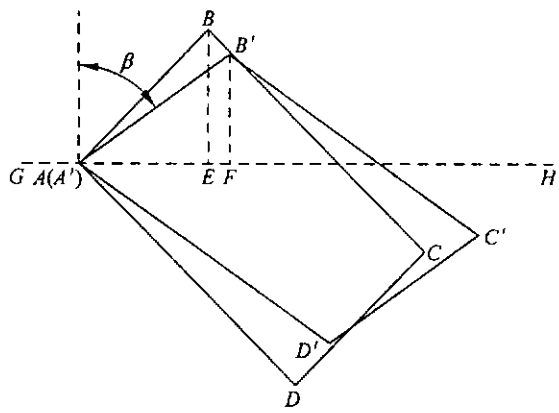


图 7.17 不等比插入块参照实现“斜切”功能

从图中可以看出$\overline{A'B'}=\overline{AB}$,$\overline{A'D'}=\overline{AD}$,$\angle B'A'D'=90°-\alpha$,

X 向缩放比例为：$x_{SC}=\dfrac{\overline{A'F}}{\overline{AE}}=\dfrac{\overline{A'B'}\cdot\cos\dfrac{90°-\alpha}{2}}{\overline{AB}\cdot\cos45°}=\dfrac{\cos(45°-0.5\alpha)}{\cos45°}$

Y 向缩放比例为：$y_{SC}=\dfrac{\overline{B'F}}{\overline{BE}}=\dfrac{\overline{A'B'}\cdot\sin\dfrac{90°-\alpha}{2}}{\overline{AB}\cdot\sin45°}=\dfrac{\sin(45°-0.5\alpha)}{\sin45°}$

旋转角度为：$\beta=90°-\dfrac{90°-\alpha}{2}=45°+0.5\alpha$

最后一步旋转实体可在插入块参照时一并完成，不需要单独进行。

7.6.3 步骤

(1) 在 VBA 集成开发环境中，创建一个新工程，保存在适当的位置。在“工程资源管理器”窗口中，双击 ThisDrawing 图标，打开该模块的代码窗口，在模块起始处添加常量 PI 的定义：

```
Const PI As Double = 3.1415926535897
```

(2) 在代码窗口中添加一个函数 DegreeToRadian，用于将角度转化为弧度，其实现代码为：

```
Private Function DegreeToRadian(angle As Double) As Double
    DegreeToRadian = angle * PI/180
```

```
End Function
```

(3) 创建 Chamfering 宏，实现斜切的功能，其实现代码为：

```
Public Sub Chamfering( )
    On Error Resume Next
    '构造选择集
    Dim SSet As AcadSelectionSet
    If Not IsNull(ThisDrawing.SelectionSets.Item("chamfering")) Then
        Set SSet = ThisDrawing.SelectionSets.Item("chamfering")
        SSet.Delete
    End If
    Set SSet = ThisDrawing.SelectionSets.Add("chamfering")
    ThisDrawing.Utility.Prompt ("选择要斜切的对象…")
    SSet.SelectOnScreen

    '提示用户输入拾取基点
    Dim ptBase As Variant
    ptBase = ThisDrawing.Utility.GetPoint(, "请输入斜切对象的基点：")

    '拾取块参照的插入点
    Dim pt2 As Variant
    pt2 = ThisDrawing.Utility.GetPoint(ptBase, "请输入斜切对象的插入点：")

    Dim angle As Double
    angle = ThisDrawing.Utility.GetAngle(ptBase, "请输入倾斜角度：")

    '如果输入的角度接近 90°的整数倍，将无法完成操作
    If Abs((angle/(0.5 * PI)) - Int(angle/(0.5 * PI))) < 0.001 Then
        MsgBox ("您输入的角度不合适，无法完成操作!")
        Exit Sub
    End If

    '将选择集中的每一个对象旋转-45°
    Dim ent( ) As Object, i As Integer
    ReDim ent(SSet.Count - 1) As Object
    For i = 0 To SSet.Count - 1
        Set ent(i) = SSet.Item(i)
        ent(i).Rotate ptBase, DegreeToRadian( - 45)
    Next

    '创建块
    Dim strBlkName As String, n As Integer
    strBlkName = "CAD 倾斜"
    n = 1
```

```
    Dim blockObj As AcadBlock
BLOCK:
    For Each blockObj In ThisDrawing.Blocks
        If blockObj.Name = strBlkName Then
            strBlkName = "CAD 倾斜" & "_" & CStr(n)
            n = n + 1
            GoTo BLOCK
        End If
    Next blockObj

    Set blockObj = ThisDrawing.Blocks.Add(ptBase, strBlkName)

    '向块添加对象
    Dim objCollection( ) As Object
    ReDim objCollection(SSet.Count - 1) As Object
    For i = 0 To SSet.Count - 1
        Set objCollection(i) = SSet.Item(i)
    Next

    Dim retObjects As Variant
    retObjects = ThisDrawing.CopyObjects(objCollection, blockObj)

    '将选择集中的每一个对象旋转 45°
    For i = 0 To SSet.Count - 1
        ent(i).Rotate ptBase, DegreeToRadian(45)
    Next

    '计算插入块参照的 X、Y、Z 方向比例因子和旋转角度
    Dim xScale As Double, yScale As Double, zScale As Double, ang As Double
    xScale = Cos(PI/4 - angle/2)/Cos(DegreeToRadian(45))
    yScale = Sin(PI/4 - angle/2)/Sin(DegreeToRadian(45))
    zScale = 1
    ang = PI/4 + angle/2

    '插入块参照
    ThisDrawing.ModelSpace.InsertBlock pt2, strBlkName, xScale, yScale, zScale, ang
End Sub
```

首先，采用安全的机制创建了一个名称为 chamfering 的选择集，用于提示用户选择所要进行斜切操作的实体。用户输入斜切的角度之后，判断输入角度是否是 90 的整数倍，如果是就退出程序的执行，程序的目的是进行斜切操作，因此输入 90 整数倍的角度是没有意义的，这不难理解。需要注意的是判断角度是否有效的方法：

```
If Abs((angle/(0.5 * PI)) - Int(angle/(0.5 * PI))) < 0.001 Then
    MsgBox ("您输入的角度不合适，无法完成操作!")
    Exit Sub
```

```
End If
```

Int 函数返回指定参数值的整数部分，如果 angle 是 90（亦即 PI/2）的整数倍，那么 angle/(0.5 * PI)和 Int(angle/(0.5 * PI))的值必然相等，If 语句的条件就成立，程序退出。angle 是前一步 GetAngle 方法的返回值，是一个弧度值（GetAngle 方法返回的只能是弧度值），因此要将 angle 与 PI/2 相除，而不是直接与 90 相除。

下面的这段代码可能会令人疑惑：

```
Dim blockObj As AcadBlock
BLOCK:
For Each blockObj In ThisDrawing.Blocks
    If blockObj.Name = strBlkName Then
        strBlkName = "CAD 倾斜" & "_" & CStr(n)
        n = n + 1
        GoTo BLOCK
    End If
Next blockObj

Set blockObj = ThisDrawing.Blocks.Add(ptBase, strBlkName)
```

仔细观察这段代码，其目的实际上是防止在同一幅图形中多次向同一块定义中添加实体而导致错误。如果当前图形中已经存在“CAD 倾斜”、“CAD 倾斜_1”等块定义，程序会自动选择一个合适的名称来创建新的块定义。

7.6.4　实例效果

在 AutoCAD 2005 中创建一个矩形，执行 Chamfering 宏，选择矩形作为斜切的对象，然后选择矩形的左下角点作为基点，输入插入点和角度，能够得到如图 7.18 所示的结果。

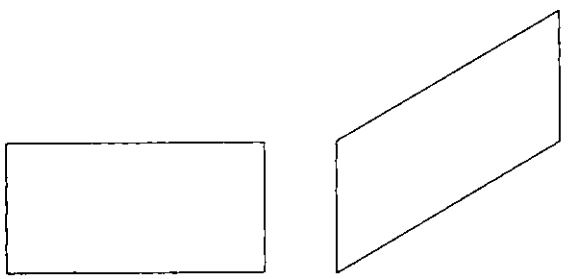


图 7.18　执行斜切的结果

7.6.5　小结

学习本节之后，需要掌握下面的几个知识点：

- 安全创建选择集。
- 对选择集中的对象进行编辑。
- 用 CopyObjects 方法向块定义中添加对象。
- 不等比插入块参照。

7.7 块属性生成表格

7.7.1 实例说明

属性块与普通块的不同点是，普通块一成不变，每一插入的块参照都完全是块定义的副本，而属性块则在文字上可以有变化，这些文字就是属性块中的属性。属性块中的属性一般用于显性或隐性地标记某功能图元组的编号或说明性内容。例如：在教室中需要对每一桌子进行编号及标明姓名，可用属性块来实现。属性块用于生成明细表时最实用的。

属性块中属性的输出问题一直是一个热门话题，在 AutoCAD 的旧版本中，就已经提供了将属性块输出成某格式的文本文件的功能。而属性块与 Excel 之间的链接互导也一直是许多二次开发者关注的内容。

自 AutoCAD 2005 开始，AutoCAD 就增加了表格对象，虽然表格对象还有许多不尽人意的地方，但毕竟是个开始。以前使用属性块做明细表，在 AutoCAD 2005 版本后就可以直接使用表格来做明细表了，这样更方便用户的编辑。

注意，块属性生成表格这样的功能在 AutoCAD 2006 版本中已经提供，但做为一种知识，使用户更加了解属性块和表格这两种对象的使用方法，本节就是很好的一个教学实例。

7.7.2 编程思路

属性块中的属性内容是可以通过程序提取出来的。

本程序通过选择单个属性块，再以属性块名称过滤出所有同名的属性块，读取这些属性块中的属性并写入表格中。表格的行数以属性块数量多少来计算，表格的列数则以属性块中属性的多少来计算，表格的表头内容需要读取属性定义的内容。

7.7.3 步骤

(1) 在 AutoCAD2005 中，打开 VBA 管理器，新建一个工程，保存在适当的位置，进入 VBA 集成开发环境(注意，该程序只能在 AutoCAD 2005 或 AutoCAD 2006 中运行)。

(2) 在“工程资源管理器”窗口中双击 ThisDrawing 图标，打开其代码窗口，在其中添加一个函数 AddBlkTable，用于按行列数添加表格，其实现代码为：

```
Function AddBlkTable(InsertionPoint As Variant, TableColCount As Double, TableRowCount As
    Double)
    Dim Table As AcadTable
    Dim RowHeight As Double, Colwidth As Double
    RowHeight = 6: Colwidth = 40 '行高及列宽
    Set Table = ThisDrawing.ModelSpace.AddTable _
                (InsertionPoint, TableRowCount + 1, TableColCount, RowHeight, Colwidth)
    Table.HeaderSuppressed = True
    '取消原先表格格式中的首行合并
```

```
    Table.UnmergeCells 0, 0, 0, TableColCount - 1 '按顺序为合并的起始行号、结束行号、起始列
                                                  '号、结束列号
    Table.SetTextHeight 5, 5
    'Table.SetAlignment 3, 5
    Set AddBlkTable = Table
    'Debug.Print Table.Rows
End Function
```

(3) 添加一个函数 CreatSSet,用于在图形中创建一个空白选择集,其实现代码为:

```
Function CreatSSet( ) As AcadSelectionSet
    On Error Resume Next
    ThisDrawing.SelectionSets("mccad").Delete
    Set CreatSSet = ThisDrawing.SelectionSets.Add("mccad")
End Function
```

(4) 在 ThisDrawing 模块的代码窗口中添加宏的实现代码:

```
Sub Att2Table( )
    On Error Resume Next
    Dim Ent As AcadEntity
    Dim Pnt As Variant

    '提示选择属性块,并判断块中是否含有属性
    Do
        ThisDrawing.Utility.GetEntity Ent, Pnt, vbCrLf & "请选择要提取属性的块: "
        If Err.Number <> 0 Then Exit Sub
        If Ent.ObjectName = "AcDbBlockReference" Then
            If Ent.HasAttributes = True Then
                Exit Do
            End If
        End If
    Loop
    Dim BlkRef As AcadBlockReference
    Set BlkRef = Ent
    Dim BlkName As String
    BlkName = BlkRef.Name

    '创建空白选择集
    Dim SS As AcadSelectionSet
    Set SS = CreatSSet

    '设置过滤条件,将所有同名的带属性块过滤出来
    Dim FilterType As Variant
    Dim FilterData As Variant
    Dim FType(2) As Integer
    Dim FData(2) As Variant
```

```
FType(0) = 0
FData(0) = "INSERT"      '图元名
FType(1) = 66
FData(1) = 1     '带属性
FType(2) = 2
FData(2) = BlkName     '图块名
FilterType = FType
FilterData = FData
SS.Select acSelectionSetAll, , , FilterType, FilterData

Dim i As Integer
Dim j As Integer
Dim Blk As AcadBlock
Dim Att As AcadAttribute
Dim AttRef As AcadAttributeReference
Dim AttRefs As Variant
Dim Rows As Double
Dim Cols As Double
Dim Table As AcadTable

'遍历选择集中的属性块
For i = 0 To SS.Count - 1
    Set BlkRef = SS(i)
    AttRefs = BlkRef.GetAttributes

    '添加表格,并设置表头
    If i = 0 Then
        Cols = UBound(AttRefs) + 1
        Rows = SS.Count
        Dim InsertionPoint As Variant
        InsertionPoint = ThisDrawing.Utility.GetPoint(, vbCrLf & "请选择表格插入点: ")
        Set Table = AddBlkTable(InsertionPoint, Cols, Rows)
        Set Blk = ThisDrawing.Blocks(BlkRef.Name)

        '提取属性定义内容中的提示作为表头
        For Each Ent In Blk
            If Ent.ObjectName = "AcDbAttributeDefinition" Then
                Set Att = Ent
                Table.SetText 0, j, Att.PromptString
                j = j + 1
            End If
        Next
    End If

    '遍历属性块中的每一属性内容,将属性内容填入表格相应的行中
```

```
            For j = 0 To UBound(AttRefs)
                Set AttRef = AttRefs(j)
                Table.SetText i + 1, j, AttRef.TextString
            Next
        Next
    End Sub
```

7.7.4　实例效果

(1) 在 VBA 集成开发环境中，按下 F5 键执行程序。

(2) 回到 AutoCAD 界面，此时可看到命令行中提示“请选择要提取属性的块”，可以选择图形中带有属性的块。如果用户没有准备好属性块，可以打开配套光盘中提供的示例图形(Att2TableSample.dwg 文件)，该图形中的属性块如图 7.19 所示。

6	A-06	按钮B	ABS	1	0.018	0.018	
5	A-05	按钮A	ABS	1	0.013	0.013	
4	A-04	镜片	透明ABS	1	0.008	0.008	
3	A-03	面盖	ABS	1	0.038	0.038	
2	A-02	底壳	ABS	1	0.032	0.032	
1	A-01	面壳	ABS	1	0.025	0.025	
序号	代　　号	名称与规格	材　　料	数量	单重	总重	备　　注

图 7.19　属性块样例

(3) 选择了带属性的块后，命令行中会提示“请选择表格插入点：”，此时可以在图形中任意选取一个点来作为即将插入的表格位置。

(4) 完成以上操作后，程序自动将属性内容提取到表格中，得到如图 7.20 所示的结果。

序号	代号	名称与规格	材料	数量	单重	总重	备注
6	A-06	按钮B	ABS	1	0.018	0.018	
5	A-05	按钮A	ABS	1	0.013	0.013	
4	A-04	镜片	透明ABS	1	0.008	0.008	
3	A-03	面盖	ABS	1	0.038	0.038	
2	A-02	底壳	ABS	1	0.032	0.032	
1	A-01	面壳	ABS	1	0.025	0.025	

图 7.20　提取后生成的表格

7.7.5　小结

属性块分为属性定义和属性参照两种类型。未插入的块定义中包含的是属性定义，而插入后的块参照中包含的是属性参照。

属性定义和属性参照的访问方式不同。属性定义是块中的对象，通过遍历块中的对象可以得到，而属性参照不是通过遍历的方式取得，它必须通过 GetAttributes 的方式形成属性数组进行访问。

属性定义和属性参照中保存的数据不同，属性参照中保存的是属性的内容，如 TagString（标签）、TextString（文字），而属性中除了保存以上内容外，还保存了 PromptString（提示字符串）。在本节程序中，为了设置表格的表头内容，必须读取相应属性块对象的属性定义中的提示字符串内容。

属性参照除了可以修改属性内容（文字内容）外，还可以修改属性的样式，如对齐方式、字体、文字大小、图层等。

第 8 章　使用 AutoCAD 中的事件

事件是 Windows 编程的基本概念之一，Windows 程序运行的过程中充满着对各种事件的处理，例如窗体加载时初始化控件、单击按钮时弹出信息对话框、用户选择一个列表项之后更新其他控件的属性等。

在 AutoCAD 中，为了方便程序员对事件进行处理，将 AutoCAD 中的事件分成三种类型：

- 应用程序层事件。顾名思义，这类事件在 AutoCAD 应用程序及其环境更改时被触发，例如打开图形、创建新图形、打印、AutoCAD 命令执行、系统变量的更改和应用程序窗口的更改等。这类事件的特点是，与具体的对象和文档无关。
- 文档层事件。这类事件与特定的文档有关，可以响应文档内容的更改。这些事件可以响应对象的添加、删除或修改，快捷菜单的激活，优先选择集的更改，“图形”窗口的更改以及图形的重生成。还有一些文档层事件会响应图形的打开、关闭和打印，从图形加载或卸载 ARX 和 LISP 应用程序。
- 对象层事件。这类事件仅包含一种，就是特定对象的修改。

AutoCAD 捕获某个事件之后，会执行事件处理程序，事件中包含的信息（例如 BeginOpen 事件中的图形名称）会通过参数传递给事件处理程序。

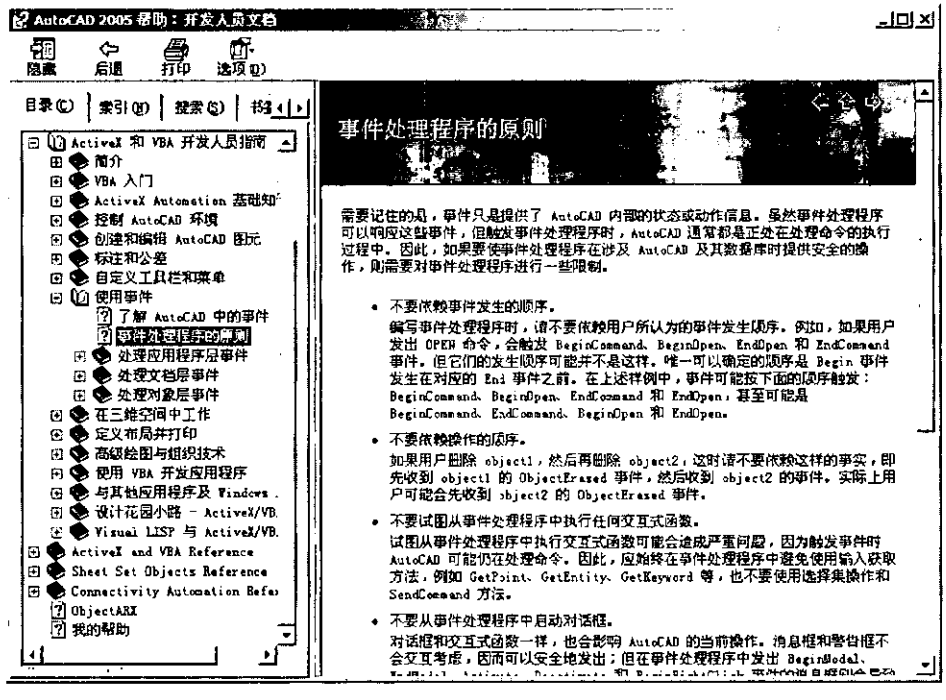


图 8.1　处理 AutoCAD 事件的原则

在进行处理时，要遵循一定的原则：

- 不要依赖事件发生的顺序。
- 不要依赖操作的顺序。
- 不要试图从事件处理程序中执行任何交互式函数。
- 不要从事件处理程序中启动对话框。
- 可以向数据库中的任何对象写入数据，但发出事件的对象除外。
- 不要从事件处理程序中执行任何触发相同事件的操作。
- 请记住，当 AutoCAD 显示模式对话框时，不会触发任何事件。

关于这些原则的详细介绍，可以参看《AutoCAD 2005 帮助：开发人员文档》中的相关内容，如图 8.1 所示。

8.1 处理应用程序层事件

8.1.1 实例说明

在 AutoCAD 中，有很多应用层事件可以使用，例如加载和卸载 ObjectARX 应用程序的事件、命令执行前后的事件、打开和关闭文件的事件等。本节将以几个典型的应用为例，介绍在 VBA 中处理应用程序层事件的一般方法。

本节的实例将实现三个功能：

- 在 AutoCAD 中“禁用”某些命令，或者替换其执行方式。
- 打开特定的图形时执行一些任务。
- 实现按照实体类型的分层，也就是在创建实体时将不同类型的实体放置在不同的图层中。

8.1.2 编程思路

1. 截获 AutoCAD 中的命令

BeginCommand 事件紧接着命令发出之后，但在完成之前触发，并且在事件处理过程中传递当前所执行命令的名称，因此可以在处理过程中获取当前执行命令的名称。如果当前执行的命令是需要禁用的命令，可以执行下面的代码：

```
SendKeys "{ESC}" & "{ESC}", True
```

SendKeys 函数将一个或多个按键消息发送到活动窗口，就如同在键盘上进行输入一样。上面的代码相当于在命令执行过程中，用户在键盘上两次按下 Esc 键，也就是中断当前命令的执行。如果要替换该命令执行的方式，就可以在中断当前命令之后执行自己需要的功能。

除此之外，实际上还有一种方法，就是使用 AutoCAD 中的 UNDEFINE 命令来禁用指定的命令，例如，可以通过下面的代码禁用 MOVE 命令：

```
ThisDrawing.SendCommand "UNDEFINE" & vbCr & "MOVE" & vbCr
```

禁用 MOVE 命令之后，如果在命令行执行该命令，系统会提示“未知命令“MOVE”。按 F1 查看帮助。”实际上，被禁用的命令还可以在命令行执行，不过要输入“圆点(.)＋命令名称”的形式，例如. MOVE。

禁用的命令可以通过 REDEFINE 命令来重新启用，例如，可以通过下面的代码重新启用 MOVE 命令：

```
ThisDrawing.SendCommand "REDEFINE" & vbCr & "MOVE" & vbCr
```

2. 打开特定的图形时执行一些任务

BeginOpen 事件在 AutoCAD 收到打开现有图形的请求之后立即触发，在事件处理过程中传递当前所打开的图形文件的名称，可以在处理过程中执行一些特定的任务。

3. 按照实体类型进行分层

仍然可以使用 BeginCommand 事件，在事件处理过程中判断当前的命令名称，在创建实体的命令执行之前，可将要放置此类实体的图层设置为当前图层。

8.1.3 步骤

(1) 在 AutoCAD 2005 中，打开 VBA 管理器，创建一个新工程，保存在适当的位置。进入 VBA 集成开发环境，选择“插入”|“类模块”菜单项，向工程中添加一个类模块，重新命名为 EventClassModule。

(2) 在 EventClassModule 类模块中添加下面的代码，这样才可以在类模块中启用应用程序层事件：

```
Public WithEvents App As AcadApplication
```

(3) 在代码窗口左上角的列表中选择 App 选项，右上角的列表中选择 BeginOpen 选项，系统会自动在代码窗口中添加 BeginOpen 事件的处理过程，即在其中添加下面的代码：

```
Private Sub App_BeginCommand(ByVal CommandName As String)
    If StrComp(CommandName, "move", vbTextCompare) = 0 Then
        SendKeys "{ESC}" & "{ESC}", True
    ElseIf StrComp(CommandName, "line", vbTextCompare) = 0 Then
        ThisDrawing.ActiveLayer = ThisDrawing.Layers.Item("直线")
    ElseIf StrComp(CommandName, "arc", vbTextCompare) = 0 Then
        ThisDrawing.ActiveLayer = ThisDrawing.Layers.Item("圆弧")
    ElseIf StrComp(CommandName, "pline", vbTextCompare) = 0 Then
        ThisDrawing.ActiveLayer = ThisDrawing.Layers.Item("多段线")
    ElseIf StrComp(CommandName, "circle", vbTextCompare) = 0 Then
        ThisDrawing.ActiveLayer = ThisDrawing.Layers.Item("圆")
    End If
End Sub
```

上面的代码禁用了 MOVE 命令，同时在执行 LINE、ARC、PLINE 和 CIRCLE 命令时自动将相应实体的图层设置为当前图层。

(4) 按照同样的方法对 BeginOpen 事件进行处理,其相关代码为:

```
Private Sub App_BeginOpen(FileName As String)
    If StrComp(FileName, "C: \AutoCAD 图形\points.dwg", vbTextCompare) = 0 Then
        ThisDrawing.Application.LoadDVB "C: \points.dvb"
    ElseIf StrComp(FileName, "C: \AuotCAD 图形\floor1.dwg", vbTextCompare) = 0 Then
        ThisDrawing.Application.LoadDVB "C: \floor1.dvb"
    ElseIf StrComp(FileName, "C: \AuotCAD 图形\group.dwg", vbTextCompare) = 0 Then
        ThisDrawing.Application.LoadDVB "C: \group.dvb"
    End If
End Sub
```

上面的代码根据当前打开的图形加载处理该图形所需要的工程。还可以在加载一个 DVB 文件之前判断该文件是否存在,也就是使用下面的代码:

```
If Len(Dir("C: \points.dvb")) > 0 Then
    ThisDrawing.Application.LoadDVB "C: \points.dvb"
End If
```

(5) 在"工程资源管理器"窗口中双击 ThisDrawing 图标,打开其代码窗口,在其中添加 EventClassModule 类的一个实例:

```
Dim X As New EventClassModule
```

(6) 添加一个宏 InitializeEvents,用于启用 AutoCAD 的应用程序层事件,其实现代码为:

```
Public Sub InitializeEvents( )
    Set X.App = ThisDrawing.Application

    '创建若干新图层
    Call NewLayer("直线")
    Call NewLayer("圆")
    Call NewLayer("圆弧")
    Call NewLayer("多段线")
End Sub
```

其中,NewLayer 过程能够安全(如果已经存在指定名称的图层,就不再创建该图层)地向当前图形中添加一个图层,其实现代码为:

```
Public Sub NewLayer(ByVal strLayer As String)
    On Error Resume Next

    If Not IsNull(ThisDrawing.Layers.Item(strLayer)) Then
        ThisDrawing.Layers.Add strLayer
    End If
End Sub
```

8.1.4 实例效果

(1) 在 VBA 集成开发环境中,执行 InitializeEvents 宏。

(2) 在 AutoCAD 命令窗口中执行 MOVE 命令,系统会自动停止该命令的执行。

(3) 在 AutoCAD 中，打开一个已经存在的图形，如果该图形是 BeginOpen 事件中所指定的一个，系统会自动加载与其相对应的 DVB 文件。

(4) 执行 LINE、ARC、PLINE 和 CIRCLE 命令，系统会自动切换到相应的图层，非常方便。

8.1.5　小结

本节介绍了处理 AutoCAD 应用程序层事件的一般方法，并且实现了在 AutoCAD 中截获并取消命令的功能。需要指出的是，使用 SendKeys 函数取消命令，对于在执行过程中会弹出对话框的命令无效，例如 OPEN、ARRAY 等。这个问题目前没有更好的处理方法。

8.2　处理文档层事件

8.2.1　实例说明

AutoCAD 提供了多种文档层事件可供用户处理，其中比较常用的是 BeginDoubleClick 和 BeginShortcutMenuEdit 事件。BeginDoubleClick 事件在用户双击图形中的对象之后触发，并在事件处理的过程中传递双击时光标所在点的坐标；BeginShortcutMenuEdit 事件则是用户在图形窗口中右击之后、“编辑”模式中出现快捷菜单之前触发。

在 AutoCAD 中，根据右击时所处的状态不同，可以弹出以下的几种快捷菜单。

- “命令”模式：在执行命令的过程中右击，弹出“命令”模式的快捷菜单。
- “默认”模式：在系统未执行任何操作时右击，弹出“默认”模式的快捷菜单。
- “编辑”模式：在系统中选择实体之后右击，弹出“编辑”模式的快捷菜单。
- “夹点”模式：在系统中激活某个夹点之后右击，弹出“夹点”模式的快捷菜单。
- “对象捕捉”模式：在系统中按下 Ctrl 键或 Shift 键的同时右击，弹出“对象捕捉”模式的快捷菜单。

在 AutoCAD 的文档层事件中，分别为各种模式提供了不同的事件处理方式，本节以 BeginDoubleClick 和 BeginShortcutMenuEdit 事件为例，介绍处理 AutoCAD 文档层事件的方法。

执行本节的实例之后，在图形窗口中双击一个包含属性的块参照，系统会弹出自定义的编辑属性对话框，用户可在此对话框中编辑块参照的属性值；双击图形中的一个文字，系统会弹出一个信息框，显示当前所双击文字的内容；在图形窗口中选择若干实体之后，在图形窗口中右击，在弹出的快捷菜单中会增加一个菜单项“修改颜色”，选择此菜单项，选择的实体会被修改为红色。

8.2.2　编程思路

1. 处理 BeginDoubleClick 事件

处理 BeginDoubleClick 事件的过程仅传递了双击的位置，要判断在双击的时候是否已经选择了实体，可使用 PickfirstSelectionSet 对象。实际上在 Windows 内部，双击会触发两

个单击事件和一个双击事件，在第一个单击事件触发的时候，如果光标下方有实体就会被选中，并将其添加到 PickfirstSelectionSet 对象所代表的选择集中。

要对双击时选择的实体进行处理，实际上就是处理 PickfirstSelectionSet 选择集中的对象。

2. 处理 BeginShortcutMenuEdit 事件

处理 BeginShortcutMenuEdit 事件的过程会传递一个 ShortcutMenu 参数，表示此时所弹出的快捷菜单，可使用 PopupMenu 对象的 AddMenuItem 方法向快捷菜单中添加一个菜单项。

为了不让程序在系统中留下“垃圾”，可以在快捷菜单弹出之后将添加的菜单项删除，可以在 EndShortcutMenu 事件中进行处理。

3. 获得和修改块参照的属性

使用块参照的 GetAttributes 方法可从块参照中获得所有的属性参照，该方法返回一个 Variant 变量，保存了块参照中所有属性参照的数组。

要修改块参照中的某个属性值，可以使用属性参照对象的 TextString 方法：

```
varAttributes(i).TextString = newValue
```

8.2.3 步骤

(1) 在 AutoCAD 2005 中，打开 VBA 管理器，创建一个新工程，保存在适当的位置。进入 VBA 集成开发环境，双击 ThisDrawing 图标打开其代码窗口，从左上角的列表中选择 AcadDocument 选项，从右上角的列表中选择 BeginDoubleClick 选项，添加对 BeginDoubleClick 事件处理的代码：

```
Private Sub AcadDocument_BeginDoubleClick(ByVal PickPoint As Variant)
    Dim ss As AcadSelectionSet
    Dim ent As AcadEntity
    Set ss = ThisDrawing.PickfirstSelectionSet

    '如果未选中任何的对象，则不做处理
    If ss.Count = 0 Then Exit Sub

    Set ent = ss.Item(0)

    '处理块参照
    If ss.Count = 1 And TypeOf ent Is AcadBlockReference Then
        Set objBlkRef = ent          '保存在全局对象中，已在 frmEditBlkRef 对话框中使用

        If objBlkRef.HasAttributes Then
            frmEditBlkRef.Show
        End If
    End If
```

```
    '处理文字
    If ss.Count = 1 And TypeOf ent Is AcadText Then
        MsgBox ent.TextString
        SendKeys "{ESC}", True
    End If
End Sub
```

首先使用 PickfirstSelectionSet 对象获得在处理事件之前已经选择的实体，如果该选择集中未包含任何对象，则不执行任何操作。

如果选择的实体是块参照，并且块参照包含属性，将块参照对象保存在一个全局对象 objBlkRef 中，并打开 frmEditBlkRef 对话框，在其中对块参照的属性进行编辑。

如果选择的实体是文字，就弹出一个对话框，在对话框中显示字符串的内容。

(2) 选择“插入”|“模块”菜单项，向工程中添加一个标准模块，在其中添加一个全局变量的定义：

```
Public objBlkRef As AcadBlockReference     '双击时被选择的块参照
```

由于需要在 ThisDrawing 模块和 frmEditBlkRef 窗体中共享该变量的值，必须将其定义成全局变量。

(3) 选择“插入”|“用户窗体”菜单项，向工程中添加一个对话框，将其名称修改为 frmEditBlkRef，在其中添加如图 8.2 所示的控件。

其中，列表控件(ListView)的名称为 lvwAttributes，文本框的名称为 txtValue，两个命令按钮的名称为 cmdOk 和 cmdCancel。

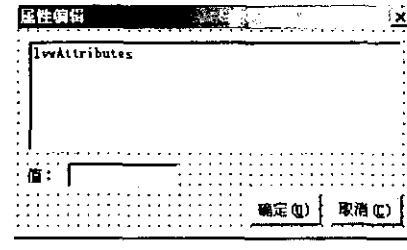


图 8.2　设计用户窗体

(4) 在窗体初始化的事件中，对 ListView 控件进行初始化，获得当前选择的块参照的属性，将其属性值添加到 ListView 控件中，其相关代码为：

```
Private Sub UserForm_Initialize( )
    '初始化 ListView
    lvwAttributes.View = lvwReport
    lvwAttributes.LabelEdit = lvwManual
    lvwAttributes.FullRowSelect = True

    '为 ListView 添加表头
    Dim colH As ColumnHeader
    Set colH = lvwAttributes.ColumnHeaders.Add(, , "标记", lvwAttributes.Width/4)
    Set colH = lvwAttributes.ColumnHeaders.Add(, , "提示", lvwAttributes.Width * (1 - 1/3 - 1/4))
    Set colH = lvwAttributes.ColumnHeaders.Add(, , "值", lvwAttributes.Width/3)

    '获得块参照的属性值
    Dim varAttributes As Variant
    varAttributes = objBlkRef.GetAttributes
```

```
    '添加属性值
    Dim objItem As ListItem
    Dim i As Integer
    For i = 0 To UBound(varAttributes)
        Set objItem = lvwAttributes.ListItems.Add(, , varAttributes(i).TagString)
        objItem.SubItems(1) = GetPromptString(objBlkRef, varAttributes(i))
        objItem.SubItems(2) = varAttributes(i).TextString
    Next i
End Sub
```

其中 GetPromptString 是一个自定义函数，用于获得一个属性参照对象的提示字符串，其实现代码为：

```
Private Function GetPromptString(ByVal objBlkRef As AcadBlockReference, _
                                 ByVal objAttRef As AcadAttributeReference) _
                                 As String
    '获得块参照对应的块定义
    Dim objBlkDef As AcadBlock
    Set objBlkDef = ThisDrawing.Blocks.Item(objBlkRef.Name)

    '遍历块定义，获得与指定属性参照对应的属性定义
    Dim objAttDef As AcadAttribute
    Dim ent As AcadEntity
    For Each ent In objBlkDef
        If TypeOf ent Is AcadAttribute Then
            Set objAttDef = ent
            If StrComp(objAttDef.TagString, objAttRef.TagString) = 0 Then
                GetPromptString = objAttDef.PromptString
                Exit Function
            End If
        End If
    Next ent
End Function
```

严格地说，这种方法有一个缺陷，因为块定义中可能会包含两个标记相同的属性定义，但是除此之外，确实没有其他的方法访问到属性的 PromptString 属性。

(5) 用户单击 ListView 控件中的项目时，需要在文本框中显示属性值，这样用户可以通过修改文本框的值来对属性值进行编辑，其相关代码为：

```
Private Sub lvwAttributes_ItemClick(ByVal Item As MSComctlLib.ListItem)
    txtValue.Text = Item.SubItems(2)
End Sub
```

(6) 在对话框中“确定”按钮的单击事件中，使用文本框中的值对块参照中的属性参照进行更新，其相关代码为：

```
Private Sub cmdOk_Click( )
    If Len(txtValue) = 0 Then
```

```
        MsgBox "值不能为空!"
        Exit Sub
    End If

    '获得当前选择项的标记和值
    Dim strTag As String, strValue As String
    strTag = lvwAttributes.SelectedItem.Text
    strValue = lvwAttributes.SelectedItem.SubItems(2)

    '获得当前要修改的属性参照
    Dim varAttributes As Variant
    varAttributes = objBlkRef.GetAttributes

    Dim i As Integer
    For i = 0 To UBound(varAttributes)
        If StrComp(strTag, varAttributes(i).TagString) = 0 And _
            StrComp(strValue, varAttributes(i).TextString) = 0 Then
            varAttributes(i).TextString = txtValue.Text
        End If
    Next i

    Unload Me

    '取消其他的操作
    SendKeys "{ESC}", True
End Sub
```

(7) 在对话框中“取消”按钮的单击事件中，卸载对话框，并且向 AutoCAD 发送 Esc 字符，防止 AutoCAD 执行夹点编辑，其相关代码为：

```
Private Sub cmdCancel_Click( )
    Unload Me

    '防止执行夹点编辑
    SendKeys "{ESC}", True
End Sub
```

(8) 切换到 ThisDrawing 模块的代码窗口，从窗口左上角的列表中选择 AcadDocument 选项，然后从右上角的列表中选择 BeginShortcutMenuEdit 选项，用户在“图形”窗口中右击之后、“编辑”模式中出现快捷菜单之前触发这个事件，在事件发生时向当前的快捷菜单添加一个菜单项“修改颜色”，其相关代码为：

```
Private Sub AcadDocument_BeginShortcutMenuEdit(ShortcutMenu As IAcadPopupMenu, SelectionSet
    As IAcadSelectionSet)
    On Error Resume Next

    '向快捷菜单中添加菜单项
```

```
    Dim newMenuItem As AcadPopupMenuItem
    Dim strMacro As String
    strMacro = Chr(vbKeyEscape) + Chr(vbKeyEscape) + _
            " - vbarun" + Chr(vbKeySpace) + "ThisDrawing.ChangeColor" + Chr(vbKeySpace)
    Set newMenuItem = ShortcutMenu.AddMenuItem(0, Chr(Asc("&")) + "修改颜色", strMacro)

    '获得传递的选择集参数
    Set SSet = SelectionSet
End Sub
```

其中，ChangeColor 是一个自定义函数，能够修改选择集中所有实体的颜色为红色，其实现代码为：

```
Public Sub ChangeColor( )
    Dim ent As AcadEntity
    For Each ent In SSet
        ent.color = acRed
    Next ent
End Sub
```

(9) 在 ThisDrawing 模块的代码窗口中，从窗口左上角的列表中选择 AcadDocument 选项，然后从右上角的列表中选择 EndShortcutMenu 选项，在快捷菜单隐藏时将“修改颜色”菜单项从中删除，其相关代码为：

```
Private Sub AcadDocument_EndShortcutMenu(ShortcutMenu As IAcadPopupMenu)
    On Error Resume Next

    ShortcutMenu.Item("修改颜色").Delete
End Sub
```

8.2.4 实例效果

(1) 在 AutoCAD 2005 中，创建一个包含属性的块定义，在图形窗口中插入一个块参照，然后在图形窗口中首先选择块参照，再在其上面双击，系统会弹出如图 8.3 所示的对话框。

(2) 在“值”文本框中输入新的属性值，单击“确定”按钮，图形中的块参照属性值就得到更新。

(3) 在图形窗口中任意选择一个实体，然后在图形窗口中右击，系统会弹出如图 8.4 所示的快捷菜单，快捷菜单中已经增加了一个名为“修改颜色”的菜单项。如果在快捷菜单中选择了“修改颜色”菜单项，前面选择的实体颜色就会被修改成红色。

8.2.5 小结

有一个重要的文档事件 ObjectAdded 未在本节介绍，是因为在第 15 章“基本图库的实现”一节将会详细剖析该事件的使用，并且提供了在程序中启动和停止文档事件的方法。

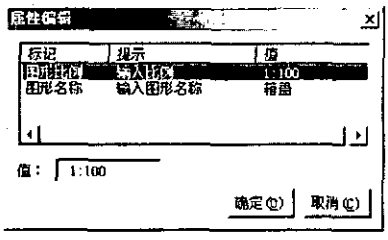


图 8.3　属性编辑的对话框

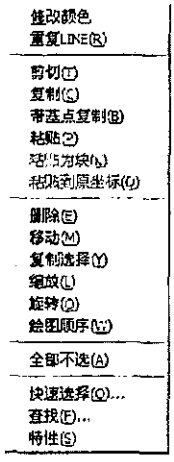


图 8.4　添加了自定义菜单项的快捷菜单

8.3　处理对象层事件

8.3.1　实例说明

AutoCAD 2005 中有一个很有意思的现象，创建一个圆时，选择“标注”|“半径”菜单项，在图形窗口中对圆创建半径标注。此后，使用 MOVE 命令移动圆，会发现标注对象随之移动；使用 SCALE 命令对圆进行放大或缩小，尺寸标注的文本随之改变。

在 6.1 节创建的圆弧长度标注却没有这么聪明，移动圆弧，标注对象还停留在原来的位置。本节将要通过处理对象层事件，使标注对象能够随着圆弧的移动而移动。

8.3.2　编程思路

在 AutoCAD VBA 中，用户可以通过一些编码来启用对象层事件，对当前图形中已经存在的一些对象进行监督，一旦这些对象被修改，就会触发 Modified 事件，并且在事件中传递发生代表修改的实体的变量。

在处理 Modified 事件的过程中，可以监督被标注了长度的圆弧位置的变化，如果其位置发生变化，对标注对象使用 Move 方法，使之与圆弧对象保持固定的相对位置。

8.3.3　步骤

(1) 在 AutoCAD 2005 中，打开 VBA 管理器，创建一个新工程，保存在适当的位置。进入 VBA 集成开发环境，选择“插入”|“模块”菜单项，向工程中添加一个新的标准模块，并在 VBA 管理器中加载 6.1 节的 DVB 文件，将 DimArcLeng 宏的实现代码复制到当前工程的标准模块中。

(2) 卸载 6.1 节的工程，在 VBA 集成开发环境中选择“插入”|“类模块”菜单项，向当前工程中添加一个类模块，并在“属性”窗口中将其名称修改为 EventClassModule。在 EventClassModule 类中声明包含事件的对象，其相关代码为：

```
Public WithEvents Object As AcadArc
```

(3) 切换到标准模块中,在其中添加下面的声明:

```
Dim X As New EventClassModule          '创建类模块的一个实例
Public DimAng As AcadDimAngular        '圆弧长度标注对象
Public ptCenter As Variant             '圆弧的圆心
```

由于标注对象 DimAng 需要在类模块中访问,因此作为全局变量来声明。ptCenter 保存了圆弧被修改之前的圆心坐标。

(4) 对 DimArcLeng 宏的部分代码进行修改,并且增加了若干代码,下面是修改后的代码(修改的部分已用黑体和阴影标示出来):

```
Public Sub DimArcLeng( )
    Dim Arc As AcadArc
    Dim Pnt As Variant
    Err.Clear
    On Error Resume Next

    '选择圆弧
    ThisDrawing.Utility.GetEntity Arc, Pnt, "请选择圆弧:"
    If Err.Number <> 13 And Err.Number <> 0 Then Exit Sub

    Do Until Arc.ObjectName = "AcDbArc"
        Err.Clear
        ThisDrawing.Utility.GetEntity Arc, Pnt, "你所选的不是圆弧,请重新选择圆弧:"
        If Err.Number <> 13 And Err.Number <> 0 Then Exit Sub
    Loop

    '获取圆弧各属性
    Dim Leng As Double
    Dim SPnt As Variant
    Dim EPnt As Variant
    Dim CPnt As Variant
    Leng = Arc.ArcLength
    SPnt = Arc.StartPoint
    EPnt = Arc.EndPoint
    CPnt = Arc.Center
    ptCenter = Arc.Center

    '选择标注点
    Dim PntforDim As Variant
    PntforDim = ThisDrawing.Utility.GetPoint(, "选择标注点位置:")

    '对圆弧进行角度标注
    Set DimAng = ThisDrawing.ModelSpace.AddDimAngular(CPnt, SPnt, EPnt, PntforDim)
```

```
    '获取角度标注的精度控制(小数点位)
    Dim FormatDot As Integer
    Dim FormatTxt As String
    FormatDot = DimAng.TextPrecision

    '转换为精度控制格式
    FormatTxt = "0"
    Dim i As Integer
    For i = 1 To FormatDot
        If i > 1 Then
            FormatTxt = FormatTxt & "0"
        Else    'i = 1
            FormatTxt = FormatTxt & ".0"
        End If
    Next

    '更改角度标注的文字内容
    DimAng.TextOverride = "{\Fgdt.shx;^}\P" & Format(Leng, FormatTxt)

    '为标注的圆弧启用对象层事件
    Set X.Object = Arc
End Sub
```

(5) 在 EventClassModule 类模块中，添加对象事件处理的代码：

```
Private Sub Object_Modified(ByVal pObject As IAcadObject)
    '始终保证标注对象的标注点与圆弧的起点和终点重合
    Dim objArc As AcadArc
    Set objArc = pObject

    '根据圆弧移动的情况，随之移动标注
    DimAng.Move ptCenter, objArc.Center
    DimAng.Update

    '更新圆弧的圆心
    ptCenter = objArc.Center
End Sub
```

圆弧被修改前后，圆心的位置可以确定一个矢量，该矢量反映了圆弧被移动的情况，但是要注意必须在圆弧被修改之后更新 ptCenter，使 ptCenter 始终能存储当前的圆心坐标。

8.3.4 实例效果

在 VBA 集成开发环境中，执行 DimArcLeng 宏，对图形窗口中的某个圆弧进行标注，然后使用 MOVE 命令移动其位置，就会发现标注能够随着圆弧一起移动。

8.3.5 小结

使用对象层事件的方法非常简单，但是能够实现的功能很多，本节的实例仅实现了标注随着圆弧一起移动，对于圆弧的旋转、放缩等变换则不起作用。读者可以根据本节提供的思路，编写更加完善的标注圆弧长度的函数。

8.4 序号球(永久反应器)

8.4.1 实例说明

Visual LISP 和 ObjectARX 都可以实现反应器，而 VBA 里只能用事件处理来解决，如果用文档级事件 AcadDocument_ObjectModified 处理的话，由于该事件在 AutoCad 的绘图过程中频繁触发，会严重影响 AutoCAD 的运行速度，而 VBA 里是不能做类似 VB 的控件数组的事件处理的，因此该实例提供一个 ActiveX 接口解决该问题。

8.4.2 编程思路

要做永久反应器，必须处理好反应器数据的存储，本实例使用 SetXRecord 方法将该数据存放在数据字典里，在初始化模块中读取该数据、将相关实体放在反应器的实体集合中。

使用事件处理，必须引用相应版本的类型库，以下实例以 AutoCad2005 版本为例进行说明。

8.4.3 步骤

(1) 在 VB 中新建一个 ActiveX DLL 工程，将工程名改为 TlsEvent2005，引用 AutoCAD 2005 类型库。

(2) 在工程中加入 4 个类：TlsReactorManager、TlsReactor、CDocument 和 CEntity，将类 CDocument、CEntity 的 Instancing 属性设为 1－Private。

(3) 在 CEntity 类模块中加入下列代码：

```
Private WithEvents m_oEntity As AcadObject
'事件源
Private m_Value As Variant
'附加数据
Private m_Owner As TlsReactor

Public Property Let Owner(ByVal vNewValue As TlsReactor)
    Set m_Owner = vNewValue
End Property

Public Property Let Entity(ByVal vNewValue As Object)
    Set m_oEntity = vNewValue
End Property
```

```
Public Property Let Value(ByVal vNewValue As Variant)
    m_Value = vNewValue
End Property

Private Sub m_oEntity_Modified(ByVal pObject As AutoCAD.IAcadObject)
    m_Owner.Change m_oEntity, m_Value
End Sub

Public Property Get EntityInfo( ) As Variant
On Error GoTo ErrHandle
    Dim Count As Integer
    If IsArray(m_Value) Then Count = UBound(m_Value) Else Count = -1
    EntityInfo = m_oEntity.Handle
    For i = 0 To Count
        EntityInfo = EntityInfo & "," & m_Value(i)
    Next i
ErrHandle:
End Property

Public Property Let EntityInfo(ByVal vNewValue As Variant)
On Error Resume Next
    Dim Values( )
    Dim Count As Integer
    If InStr(vNewValue, ",") Then
        vNewValue = Split(vNewValue, ",")
    Else
        vNewValue = Array(vNewValue)
    End If
    Count = UBound(vNewValue) - 1
    Set m_oEntity = m_Owner.Document.HandleToObject(vNewValue(0))

    If Count > -1 Then
        ReDim Values(Count)
        For i = 0 To Count
            Values(i) = vNewValue(i + 1)
        Next i
        m_Value = Values
    End If
End Property
```

CEntity 类是处理 Modified 事件的事件源类，m_oEntity_Modified 的事件处理调用该事件源所在集合(m_Owner)的 Change 函数来触发反应器的 Modified 事件，m_oEntity 是事件的触发者，m_Value 则是事件的附加数据。

(4) 在 CDocument 类模块中加入下列代码：

```
Private m_Entitys As New Collection
'当前文档实体集合
Private WithEvents m_oDoc As AcadDocument
'当前文档
Private m_sName As String
'反应器名
Private m_Owner As TlsReactor


Private Sub m_oDoc_BeginClose( )
'当前文档关闭时,移除本文档反应器集合
    Set m_Entitys = Nothing
    m_Owner.RemoveDoc m_oDoc.hWnd
End Sub

Public Property Let Owner(ByVal vNewValue As TlsReactor)
    Set m_Owner = vNewValue
End Property

Public Property Let Application(ByVal vNewValue As AcadApplication)
'获取 AutoCad 对象实例时对当前文档初始化,读取存储的数据
On Error Resume Next
    Dim ent As AcadObject
    Dim Infos
    Dim oEnt As CEntity
    Set m_oDoc = vNewValue.ActiveDocument

    Infos = GetXRecord("TlsEvent", m_sName)
    For Each i In Infos
        Err.Clear
        Set ent = m_oDoc.HandleToObject(LeftStr(i, ","))
        If Not Err And i <> "" Then
            Set oEnt = New CEntity
            oEnt.Owner = m_Owner
            oEnt.EntityInfo = i
            m_Entitys.Add oEnt
        End If
    Next i
End Property

Public Property Let Name(ByVal vNewValue As String)
    m_sName = vNewValue
End Property

Public Sub Add(ByVal Entity As CEntity)
```

```
'向当前文档实体集合加入实体
On Error Resume Next
    Dim Infos As New Collection
    Dim Info As String

    m_Entitys.Add Entity
    For Each i In m_Entitys
        Info = i.EntityInfo
        If Info <> "" Then Infos.Add Info
    Next i
    SetXRecord "TlsEvent", m_sName, Col2Arr(Infos)

End Sub

Private Function Col2Arr(ByVal Values As Collection)
'转换集合为数组
    Dim arr( )
    ReDim arr(Values.Count - 1)
    For i = 0 To Values.Count - 1
        arr(i) = Values(i + 1)
    Next i
    Col2Arr = arr
End Function

Private Function SetXRecord(ByVal DictName As String, ByVal Keyword As String, ByVal XRecordData)
'存储数据
On Error Resume Next
    Dim pDict As AcadDictionary
    Dim pXRecord As AcadXRecord
    Dim XRecordType( ) As Integer
    Dim pLen As Integer

    Set pDict = m_oDoc.Dictionaries.Add(DictName)
    Set pXRecord = pDict.AddXRecord(Keyword)

    pLen = UBound(XRecordData)
    ReDim XRecordType(pLen) As Integer
    For i = 0 To pLen
        Select Case VarType(XRecordData(i))
            Case vbInteger, vbLong
                XRecordType(i) = 70
            Case vbSingle, vbDouble
                XRecordType(i) = 40
            Case vbString
                XRecordType(i) = 1
```

```
            End Select
        Next i

        pXRecord.SetXRecordData XRecordType, XRecordData

    End Function

    Private Function GetXRecord(ByVal DictName As String, ByVal Keyword As String)
    '读取数据
    On Error Resume Next
        Dim pDict As AcadDictionary
        Dim pXRecord As AcadXRecord
        Dim xt
        Set pDict = m_oDoc.Dictionaries(DictName)
        Set pXRecord = pDict.GetObject(Keyword)
        pXRecord.GetXRecordData xt, GetXRecord
    End Function

    Private Function LeftStr(ByVal String1 As Variant, ByVal String2 As Variant)
    On Error Resume Next
        LeftStr = Left(String1, InStr(String1, String2) - 1)
        If InStr(String1, String2) = 0 Then LeftStr = String1
    End Function
```

由于 ThisDrawing 中的变量是在整个应用程序中的全局变量，如果简单地将数据加入反应器集合就没有办法存储对应文档的数据，使用 CDocument 类将对应文档的数据实现分离。

(5) 在 TlsReactor 类模块中加入下列代码：

```
    Private m_oApp As AcadApplication
    '当前 AutoCad 实例
    Private m_Entitys As New Collection
    '实体集合
    Private m_sName As String
    '反应器名
    Private m_bChanged As Boolean

    Public Event Modified(ByVal pObject As AcadObject, ByVal Value)

    Public Property Let Application(ByVal vNewValue As AcadApplication)
    On Error Resume Next
        Set m_oApp = vNewValue
    End Property

    Public Property Get Document( ) As AcadDocument
        Set Document = m_oApp.ActiveDocument
```

```
End Property

Public Sub InitDoc( )
'初始化当前文档
On Error Resume Next
    ThisDrawing
End Sub

Public Property Let Name(ByVal vNewValue As String)
    m_sName = vNewValue
End Property

Public Sub Add(ByVal Entity As AcadObject, Optional ByVal Value As Variant)
'向反应器加入实体
On Error Resume Next
    Dim oEnt As New CEntity
    oEnt.Entity = Entity
    oEnt.Value = Value
    oEnt.Owner = Me
    ThisDrawing.Add oEnt
    Change Entity, Value
End Sub

Public Sub RemoveDoc(ByVal hWndStr As String)
    m_Entitys.Remove "TlsReactor_" & hWndStr
End Sub

Public Sub Change(ByVal Entity As AcadObject, ByVal Value As Variant)
'事件源调用该方法触发 Modified 事件,使用 m_bChanged 变量的目的是防止循环调用
If Not m_bChanged Then
    m_bChanged = True
    RaiseEvent Modified(Entity, Value)
    m_bChanged = False
End If
End Sub

Private Function ThisDrawing( ) As CDocument
'获取当前文档实体集合,如果没有就创建之
On Error Resume Next
    Dim sName As String
    sName = "TlsReactor_" & Document.hWnd
    Set ThisDrawing = m_Entitys(sName)
    If Err Then
        Set ThisDrawing = New CDocument
        ThisDrawing.Name = m_sName
```

```
            ThisDrawing.Owner = Me
            ThisDrawing.Application = m_oApp
            m_Entitys.Add ThisDrawing, sName
        End If
    End Function

    Private Sub Class_Initialize( )
        m_bChanged = False
    End Sub
```

TlsReactor是永久反应器类，是本实例的接口部分，该接口提供 Modified 事件给VBA，可以在 VBA 中向该事件加入代码控制反应器的行为。

(6) 在 TlsReactorManager 类模块中加入下列代码：

```
Private m_oApp As AcadApplication
Private m_Events As New Collection
'反应器集合

Public Property Let Application(ByVal vNewValue As AcadApplication)
On Error Resume Next
    Set m_oApp = vNewValue
End Property

Public Property Get Item(ByVal Name As String) As TlsReactor
'获取名称为 Name 的反应器，如果集合中没有，就创建之
On Error Resume Next
    Set Item = m_Events(Name)
    If Err Then
        Set Item = New TlsReactor
        Item.Name = Name
        Item.Application = m_oApp
        m_Events.Add Item, Name
    End If
    Item.InitDoc
End Property
```

TlsReactor 是反应器的管理类，最初的代码中是没有该类的，后来发现反应器的初始化代码太复杂，才加入该类。

(7) 编译工程，生成 ActiveX DLL 文件。

(8) 打开 AutoCAD2005，运行 VBAIDE 命令，打开 VBA 编辑器，引用生成的 ActiveX DLL，然后在 ThisDrawing 模块中加入下列代码：

```
Private m_Reactors As New TlsReactorManager
Private WithEvents m_XhqReactor As TlsReactor

Public Sub EventsInit( )
```

```
    m_Reactors.Application = ThisDrawing.Application
    Set m_XhqReactor = m_Reactors("TlsXHQ")
End Sub

Private Sub m_XhqReactor_Modified(ByVal pObject As IAcadObject, ByVal Value As Variant)
On Error GoTo ErrHandle

    Dim pObj As AcadLine
    Dim pStart, pEnd

    Set pObj = ThisDrawing.HandleToObject(Value(0))

    pStart = pObj.StartPoint
    pEnd = pObject.Center
    pAngle = ThisDrawing.Utility.AngleFromXAxis(pStart, pEnd)
    pDis = ((pStart(0) - pEnd(0)) ^ 2 + (pStart(1) - pEnd(1)) ^ 2) ^ 0.5 - pObject.Radius
    pObj.EndPoint = ThisDrawing.Utility.PolarPoint(pStart, pAngle, pDis)

ErrHandle:
End Sub

Public Sub TlsXHQ( )
On Error GoTo ErrHandle
    Dim pLine As AcadLine
    Dim pCircle As AcadCircle

    p1 = ThisDrawing.Utility.getpoint(, "输入第一点：")
    p2 = ThisDrawing.Utility.getpoint(p1, "输入第二点：")
    Set pLine = ThisDrawing.ModelSpace.AddLine(p1, p2)
    pDis = ThisDrawing.Utility.GetDistance(p2, "输入圆半径：")
    Set pCircle = ThisDrawing.ModelSpace.AddCircle(p2, pDis)
    m_XhqReactor.Add pCircle, Array(pLine.Handle)
ErrHandle:
End Sub
```

(9) 在 AutoCAD2005Doc.lsp 文件中加入下列代码：

```
(vla-runmacro (vlax-get-acad-object) "EventsInit")
```

8.4.4 实例效果

(1) 保存 VBA 文档，并将生成的 DVB 文件加入启动组，关闭 AutoCAD，再重新打开。

(2) 在 AutoCAD 命令行键入 VBARUN 命令，运行 TlsXHQ 宏。

(3) 在图形窗口中拖动生成的圆，直线始终跟随圆的位置，如图 8.5 所示。

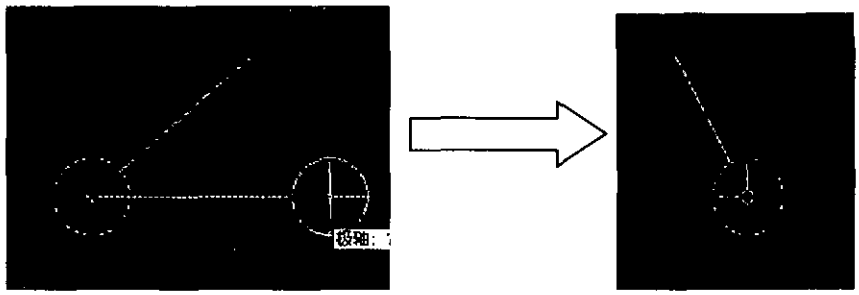


图 8.5 永久反应器的作用效果

8.4.5 小结

这里简述一下事件的触发过程，以帮助理解。

TlsXHQ 宏向反应器里添加实体，当该实体被修改，实体本身的 Modified 事件被触发，而该事件又触发反应器的 Modified 事件，执行了 Private Sub m_XhqReactor_Modified (ByVal pObject As IAcadObject, ByVal Value As Variant)事件处理过程，从而更改了直线的结束点，使直线始终指向圆。

永久反应器与普通的对象层事件处理不同，只要加载了对应的 DVB 工程文件，直线就会永远“跟着”圆，不会因为再次关闭图形、再次打开导致反应器的失效。

第9章　在三维空间工作

三维操作中UCS是一个非常重要的工具，但是VBA中仅能在WCS中创建实体，这就意味着程序员无法享受UCS在三维建模过程中带来的快捷，本章提供了在UCS中创建基本图元的方法，彻底解决这个问题。

三维空间中的操作总是和数学紧密结合，无论是三点法创建UCS、在任意平面上创建图案填充，还是创建正多面体和偏心圆台都是如此。在本章的程序实例中演示了绝大部分的三维建模方法，希望能给读者一些启示。

9.1　在UCS中创建基本图元

9.1.1　实例说明

ActiveX模型中提供了AddBox、AddCone、AddCylinder、AddEllipticalCone、AddEllipticalCylinder、AddSphere、AddTorus和AddWedge八种方法用于创建三维基本实体(长方体、圆锥体、圆柱体、椭圆柱体、椭圆锥体、球体、圆环体和楔体)。但是这些方法仅接受WCS中的坐标，而在三维图形的处理中，UCS是必不可少的，因此，提供在UCS中创建三维基本实体的方法很有意义。

二维对象的创建同样只能使用WCS的点坐标，而在三维操作过程中经常需要创建直线、圆、三维多段线、网格、多面网格等对象，同样需要提供在UCS创建二维基本图元的方法。

本节的实例对系统提供的方法进行封装，创建了在UCS中绘制三维和二维基本图元的底层函数。

9.1.2　编程思路

一下子想找出使用UCS坐标创建基本图元的方法似乎不太容易，先从逆向思维开始：对UCS和其中已经创建的一个实体进行几何变换，使UCS和WCS重合，此时该实体在此UCS和WCS中的坐标是相同的。为什么相同？原因很简单，UCS和WCS已经重合了。

那么，如果首先在一个和WCS重合的UCS中，使用WCS坐标(默认情况下创建图元的方法都是基于WCS的)来创建一个图元，然后对图元和UCS进行相同的几何变换，此后图元在UCS中的坐标和其创建时在WCS中的坐标是一致的。无论进行多么复杂的变换，只要图元和UCS进行的是相同的几何变换，这一点就会成立。

从上面的分析中，我们已经找到了使用UCS坐标创建基本图元的方法：

(1) 在WCS(或者和WCS重合的UCS)中，使用UCS下的坐标创建该图元。

(2) 获得从WCS到当前UCS的变换矩阵。

(3) 使用变换矩阵对创建的图元进行几何变换。

9.1.3 步骤

(1) 在 AutoCAD 2005 中,打开 VBA 管理器,创建一个新工程,保存在适当的位置。进入 VBA 集成开发环境,选择"插入"|"模块"菜单项,向工程中添加一个标准模块,在其中添加两个模块级变量 objCircle 和 curUcs:

```
Dim objCircle As AcadCircle     '在 WCS 中创建,并进行几何变换的圆
Dim curUcs As AcadUCS           '当前的 UCS
```

(2) 假设要在当前 UCS 中创建一个圆心为(0,0),半径为 30 的圆,就可以首先在 WCS 中创建一个相应尺寸的圆。添加一个宏 AddCircle_WCS,用于在 WCS 中创建圆心为(0,0),半径为 30 的圆,其实现代码为:

```
Public Sub AddCircle_WCS( )
    '保存当前 UCS
    Set curUcs = GetActiveUCS

    Dim ptCenter(0 To 2) As Double
    ptCenter(0) = 0: ptCenter(1) = 0: ptCenter(2) = 0

    '确保是在 WCS(与 WCS 重合的 UCS)中
    Dim objUcs As AcadUCS
    Set objUcs = GetWCS
    ThisDrawing.ActiveUCS = objUcs

    Set objCircle = ThisDrawing.ModelSpace.AddCircle(ptCenter, 30)
    ThisDrawing.Regen acActiveViewport
End Sub
```

其中,GetActiveUCS 和 GetWCS 是两个自定义函数。GetActiveUCS 函数用于获得当前的 UCS,其实现代码为:

```
Public Function GetActiveUCS( ) As AcadUCS
    '如果 UCS 没有命名,需要创建这个 UCS
    If ThisDrawing.GetVariable("UCSNAME") = "" Then
        Dim ptOrigin(2) As Double '要创建的 UCS 的原点
        Dim ptXAxis(2) As Double 'UCS 的 X 轴正半轴上一点
        Dim ptYAxis(2) As Double 'UCS 的 Y 轴正半轴上一点
        Dim xDir, yDir, org As Variant '当前 UCS 的参数

        '获得当前 UCS 的参数
        xDir = ThisDrawing.GetVariable("UCSXDIR")
        yDir = ThisDrawing.GetVariable("UCSYDIR")
        org = ThisDrawing.GetVariable("UCSORG")

        'UCS 的原点
```

```
        ptOrigin(0) = org(0)
        ptOrigin(1) = org(1)
        ptOrigin(2) = org(2)

        '获得 UCS 的 X 轴正半轴上的一点
        ptXAxis(0) = org(0) + xDir(0)
        ptXAxis(1) = org(1) + xDir(1)
        ptXAxis(2) = org(2) + xDir(2)

        '获得 UCS 的 Y 轴正半轴上的一点
        ptYAxis(0) = org(0) + yDir(0)
        ptYAxis(1) = org(1) + yDir(1)
        ptYAxis(2) = org(2) + yDir(2)

        '创建和当前 UCS 重合的 UCS
          Set GetActiveUCS = ThisDrawing.UserCoordinateSystems.Add(ptOrigin, ptXAxis,
              ptYAxis, "MyUCS")

        ThisDrawing.ActiveUCS = GetActiveUCS
    Else
        Set GetActiveUCS = ThisDrawing.ActiveUCS
    End If
End Function
```

如果当前的 UCS 已经保存，那么直接可以使用 ThisDrawing.ActiveUCS 获得当前的 UCS。如果当前的 UCS 尚未保存(在 UCS 命令中显示为“没有名称”)，使用 ThisDrawing.ActiveUCS 来获得当前的 UCS 会出现错误，如图 9.1 所示。

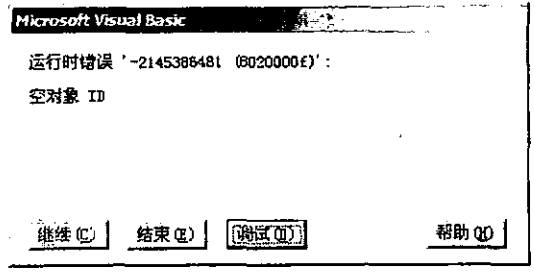


图 9.1　无法使用 ThisDrawing.ActiveUCS 获得未保存的 UCS

这种情况下，只能从系统变量中获得当前 UCS 的信息。其中，UCSXDIR 系统变量存储当前空间当前视口中当前 UCS 的 X 方向；UCSYDIR 系统变量存储当前空间当前视口中当前 UCS 的 Y 方向；UCSORG 系统变量存储当前空间当前视口的当前坐标系原点，该值总是以世界坐标形式保存。利用这些信息，完全可以获得当前 UCS 的原点、X 轴正半轴上一点和 Y 轴正半轴上一点，自然就可以创建并保存与当前 UCS 重合的 UCS 对象。

GetWCS 函数用于获得与 WCS 重合的一个 UCS(因为 VBA 中无法实现返回 WCS，故出此下策)，其实现代码为：

```
Public Function GetWCS( ) As AcadUCS
    '定义创建 UCS 的三个点
    Dim ptOrigin(2) As Double, ptXAxis(2) As Double, ptYAxis(2) As Double
    ptOrigin(0) = 0: ptOrigin(1) = 0: ptOrigin(2) = 0
    ptXAxis(0) = 1: ptXAxis(1) = 0: ptXAxis(2) = 0
    ptYAxis(0) = 0: ptYAxis(1) = 1: ptYAxis(2) = 0

    Set GetWCS = ThisDrawing.UserCoordinateSystems.Add(ptOrigin, ptXAxis, ptYAxis, "WCS")
End Function
```

创建与 WCS 重合的 UCS 比较简单，直接取 WCS 的原点、WCS 中 X 正半轴上一点和 Y 正半轴上一点即可构建一个与 WCS 重合的 UCS。

(3) 添加一个宏 ModifyCircle，用于获得 WCS 到当前 UCS 的变换矩阵，并使用该矩阵对步骤(2)生成的圆进行几何变换，其相关代码为：

```
Public Sub ModifyCircle( )
    '恢复到 UCS
    ThisDrawing.ActiveUCS = curUcs

    '获得从 WCS 到 UCS 的转换矩阵
    Dim transMatrix As Variant
    transMatrix = curUcs.GetUCSMatrix

    '对圆进行同样的变换
    objCircle.TransformBy transMatrix
    objCircle.Update
End Sub
```

(4) 可以在 AutoCAD 窗口中任意创建一个 UCS，然后依次执行 AddCircle_WCS 和 ModifyCircle 宏，必然能够在 UCS 中得到圆心为(0,0)、半径为 30 的圆。

测试证明这个思路是正确的，下面就利用这个思路对创建基本图元的函数进行封装，以实现使用 UCS 坐标来创建基本图元。

(5) 双击“工程资源管理器”窗口中的 ThisDrawing 图标，打开其代码窗口，后面的代码都放在该模块中。首先封装一下使用 UCS 坐标创建圆的函数，可以写成下面的形式：

```
Private Function AddCircle_UCS(ptCenter As Variant, radius As Double) As AcadCircle
    '保存当前的 UCS
    Dim curUcs As AcadUCS
    Set curUcs = GetActiveUCS

    '返回到 WCS
    ThisDrawing.ActiveUCS = GetWCS

    '在 WCS 中创建轻量多段线
    Dim objCircle As AcadCircle
    Set objCircle = ThisDrawing.ModelSpace.AddCircle(ptCenter, radius)
```

```
    '恢复保存的 UCS
    ThisDrawing.ActiveUCS = curUcs

    '对长方体进行变换
    Dim transMatrix As Variant
    transMatrix = curUcs.GetUCSMatrix( )
    objCircle.TransformBy transMatrix
    objCircle.Update

    Set AddCircle_UCS = objCircle
End Function
```

(6) 再添加一个函数 AddLWPoly_UCS,用于使用 UCS 坐标创建轻量多段线,其实现代码为:

```
Private Function AddLWPoly_UCS(points As Variant) As AcadLWPolyline
    '保存当前的 UCS
    Dim curUcs As AcadUCS
    Set curUcs = GetActiveUCS

    '返回到 WCS
    ThisDrawing.ActiveUCS = GetWCS

    '在 WCS 中创建轻量多段线
    Dim objLWPoly As AcadLWPolyline
    Set objLWPoly = ThisDrawing.ModelSpace.AddLightWeightPolyline(points)

    '恢复保存的 UCS
    ThisDrawing.ActiveUCS = curUcs

    '对长方体进行变换
    Dim transMatrix As Variant
    transMatrix = curUcs.GetUCSMatrix( )
    objLWPoly.TransformBy transMatrix
    objLWPoly.Update

    Set AddLWPoly_UCS = objLWPoly
End Function
```

上面代码的原理和使用 UCS 坐标创建圆完全相同,不再详细分析。之所以选择轻量多段线作为例子,是因为创建轻量多段线的方法 AddLightweightPolyline 比较特殊,它接受的是 OCS 中的坐标,这与其他的图元不太一样。

(7) 添加一个函数 AddBox_UCS,用于使用 UCS 坐标创建一个长方体,其实现代码为:

```
Private Function AddBox_UCS(ptCenter As Variant, length As Double, _
                            width As Double, height As Double) As Acad3DSolid
```

```
    '保存当前的 UCS
    Dim curUcs As AcadUCS
    Set curUcs = GetActiveUCS

    '返回到 WCS
    ThisDrawing.ActiveUCS = GetWCS

    '在 WCS 中创建长方体
    Dim objBox As Acad3DSolid
    Set objBox = ThisDrawing.ModelSpace.AddBox(ptCenter, length, width, height)

    '恢复保存的 UCS
    ThisDrawing.ActiveUCS = curUcs

    '对长方体进行变换
    Dim transMatrix As Variant
    transMatrix = curUcs.GetUCSMatrix( )
    objBox.TransformBy transMatrix
    objBox.Update

    Set AddBox_UCS = objBox
End Function
```

之所以选择长方体作为例子，是为了说明本节讲解的原理同样适合封装使用 UCS 坐标创建三维实体。

(8) 这时，可以发现一个严重的问题，下面的这段代码在 WCS 中运行得很好，但是在 UCS 中却不然：

```
Public Sub CreateLine( )
    On Error GoTo ERRHANDLE
    Dim ptStart As Variant, ptEnd As Variant

    ptStart = ThisDrawing.Utility.GetPoint(, "拾取起点：")
    ptEnd = ThisDrawing.Utility.GetPoint(ptStart, "拾取终点：")

    ThisDrawing.ModelSpace.Addline ptStart, ptEnd
    Exit Sub

ERRHANDLE:
    ThisDrawing.Utility.Prompt vbCrLf & "指定点的过程中遇到错误，未创建任何直线！" & vbCrLf
End Sub
```

其原因在于 GetPoint 方法返回的是 WCS 中的坐标，而 GetPoint 方法接受的相对点坐标又是 UCS 中的坐标，因此在拾取直线的终点时必然会发现相对点位置不对。

(9) 为了能让 CreateLine 宏正常工作，首先需要创建一个使用 UCS 坐标创建直线的函数：

```
Private Function AddLine_UCS(ptStart As Variant, ptEnd As Variant) As AcadLine
    '保存当前的 UCS
    Dim curUcs As AcadUCS
    Set curUcs = GetActiveUCS

    '返回到 WCS
    ThisDrawing.ActiveUCS = GetWCS

    '在 WCS 中创建轻量多段线
    Dim objLine As AcadLine
    Set objLine = ThisDrawing.ModelSpace.Addline(ptStart, ptEnd)

    '恢复保存的 UCS
    ThisDrawing.ActiveUCS = curUcs

    '对长方体进行变换
    Dim transMatrix As Variant
    transMatrix = curUcs.GetUCSMatrix( )
    objLine.TransformBy transMatrix
    objLine.Update

    Set AddLine_UCS = objLine
End Function
```

(10) 添加一个函数 GetPoint_UCS,用于返回用户选择点的 UCS 坐标,其实现代码为:

```
Private Function GetPoint_UCS(Optional pt As Variant = Empty) As Variant
    Dim ptWcs As Variant 'WCS 中的点坐标

    '获得用户拾取点的 WCS 坐标
    If VarType(pt) = vbEmpty Then
        ptWcs = ThisDrawing.Utility.GetPoint( )
    Else
        ptWcs = ThisDrawing.Utility.GetPoint(pt)
    End If

    '返回该点的 UCS 坐标
    GetPoint_UCS = ThisDrawing.Utility.TranslateCoordinates(ptWcs, acWorld, acUCS, False)
End Function
```

该函数的实现并不复杂,在系统的 GetPoint 方法返回点的 WCS 坐标之后,使用 TranslateCoordinates 方法将其转化成 UCS 坐标。

(11) 此时可以修改 CreateLine 宏,使其能够在 UCS 中正常工作,其实现代码为:

```
Public Sub CreateLine( )
    On Error GoTo ERRHANDLE
    Dim ptStart As Variant, ptEnd As Variant
```

```
    ptStart = GetPoint_UCS( )
    ptEnd = GetPoint_UCS(ptStart)

    AddLine_UCS ptStart, ptEnd
    Exit Sub

ERRHANDLE:
    ThisDrawing.Utility.Prompt vbCrLf & "指定点的过程中遇到错误,未创建任何直线!" & vbCrLf
End Sub
```

其中,修改的代码已经加粗。

9.1.4 实例效果

(1) 在 VBA 集成开发环境中,打开 ThisDrawing 模块的代码窗口,添加一个宏 testCircle,用于测试创建圆的函数,其实现代码为:

```
Public Sub testCircle( )
    Dim ptCenter(0 To 2) As Double
    ptCenter(0) = 100: ptCenter(1) = 100: ptCenter(2) = 0
    Dim radius As Double

    AddCircle_UCS ptCenter, 30
End Sub
```

(2) 添加一个宏 testLWPoly,用于测试创建轻量多段线的函数,其实现代码为:

```
Public Sub testLWPoly( )
    Dim utilObj As Object '必须使用后期绑定
    Set utilObj = ThisDrawing.Utility

    Dim points As Variant
    utilObj.CreateTypedArray points, vbDouble, 0, 0, 30, 30, 30, 100, 0, 100

    AddLWPoly_UCS points
End Sub
```

(3) 执行上面创建的两个宏,在 WCS 和 UCS 均能得到正确的结果。

9.1.5 小结

学习本节的内容之后,读者可以按照这个思路,创建自己的函数库,封装使用 UCS 创建各种基本图元的函数。并且,还应掌握下面的知识点:

- 三点法创建 UCS。
- TranslateCoordinates 方法进行坐标的转换。
- 使用 GetVariable 方法获得系统变量的值。

9.2　创建 UCS

9.2.1　实例说明

在 AutoCAD 中，用户可以使用 UCS 命令创建新的 UCS，其中包含很多种方法，例如三点法、移动原点法、根据对象创建 UCS、根据面创建 UCS，以及绕坐标轴旋转创建新的 UCS 等。

在 ActiveX 模型中，仅提供了三点法创建 UCS，这在很多情况下远不能满足要求，本节提供了几个函数，分别用于移动原点创建 UCS 和旋转坐标轴创建 UCS。关于三点法创建 UCS(不同于系统提供的方法，与 AutoCAD 操作中创建 UCS 的三点法相同)，将在 9.3 节(在任意的平面上创建图案填充)一节中介绍。

9.2.2　编程思路

在 ActiveX 模型中，要创建新的 UCS，可以使用 UserCoordinateSystems 集合的 Add 方法，该方法被定义为：

```
RetVal = object.Add(Origin, XAxisPoint, YAxisPoint, Name)
```

其中，Origin 为新的 UCS 原点在 WCS 中的坐标；XAxisPoint 为 UCS 的 X 正半轴上任意点在 WCS 中的坐标；YAxisPoint 为 UCS 的 Y 正半轴上任意点在 WCS 中的坐标；Name 为新的 UCS 的名称。

使用各种方法创建新的 UCS，归根结底都要调用 Add 方法，因此编程的关键在于根据已知条件确定 Origin、XAxisPoint 和 YAxisPoint 三个参数的值。

9.2.3　步骤

(1) 在 AutoCAD 2005 中，打开 VBA 管理器，创建一个新工程，保存在适当的位置。进入 VBA 集成开发环境，双击“工程资源管理器”窗口中的 ThisDrawing 图标，打开其代码窗口，首先添加两个函数 PtUcs2Wcs 和 PtWcs2Ucs，用于在 UCS 和 WCS 坐标中转换，简化书写代码的工作量，其实现代码为：

```
'将点的 UCS 坐标转化到 WCS 坐标
Private Function PtUcs2Wcs(ptUcs As Variant) As Variant
    PtUcs2Wcs = ThisDrawing.Utility.TranslateCoordinates(ptUcs, acUCS, acWorld, False)
End Function

'将点的 WCS 坐标转化到 UCS 坐标
Private Function PtWcs2Ucs(ptWcs As Variant) As Variant
    PtWcs2Ucs = ThisDrawing.Utility.TranslateCoordinates(ptWcs, acWorld, acUCS, False)
End Function
```

(2) 添加一个函数 AddOrgUCS，用于移动原点创建新的 UCS，其中 ptOriginWcs 是新的 UCS 原点在 WCS 中的坐标，其实现代码为：

```
Public Function AddOrgUCS(ptOriginWcs As Variant, strUcsName As String) As AcadUCS
    '获得新 UCS 原点在当前 UCS 中的坐标
    Dim ptOriginUcs As Variant
    ptOriginUcs = PtWcs2Ucs(ptOriginWcs)
    'Debug.Print ptOriginWcs(0)
    '获得 X、Y 正半轴上任一点的 UCS 坐标
    Dim ptXUcs(0 To 2) As Double, ptYUcs(0 To 2) As Double
    ptXUcs(0) = ptOriginUcs(0) + 1
    ptXUcs(1) = ptOriginUcs(1)
    ptXUcs(2) = ptOriginUcs(2)
    ptYUcs(0) = ptOriginUcs(0)
    ptYUcs(1) = ptOriginUcs(1) + 1
    ptYUcs(2) = ptOriginUcs(2)

    '获得 X、Y 正半轴上任一点的 WCS 坐标
    Dim ptXWcs As Variant, ptYWcs As Variant
    ptOriginWcs = PtUcs2Wcs(ptOriginUcs)
    ptXWcs = PtUcs2Wcs(ptXUcs)
    ptYWcs = PtUcs2Wcs(ptYUcs)
    'Debug.Print ptOriginWcs(0)
    '创建 UCS
     Set AddOrgUCS = ThisDrawing.UserCoordinateSystems.Add(ptOriginWcs, ptXWcs, ptYWcs,
strUcsName)
End Function
```

如果在当前 UCS 的坐标系统中移动原点创建新的 UCS 会简单很多，只需指定新的 UCS 原点（当前 UCS 下的坐标），然后就可以计算出其 X 轴正半轴上的一点（只需将原点的 X 坐标加 1，其他两个坐标不变）和 Y 轴正半轴上的一点（只需将原点的 Y 坐标加 1，其他两个坐标不变）。当然这计算出来的都是当前 UCS 下的坐标，再将其转化成 WCS 中的坐标就可以了。

编写代码的过程中有一点很让人疑惑，函数参数中输入的 ptOriginWcs 是新 UCS 原点在 WCS 中的坐标，但是并没有直接在 Add 方法中作为参数使用，而是经过 ptOriginWcs＝PtUcs2Wcs(ptOriginUcs)进行了转化才作为 Add 方法的参数使用。实际上，直接将参数中输入的 ptOriginWcs 作为 Add 方法的参数会导致一个错误，创建 UCS 时 X 轴和 Y 轴不垂直。

为了找出问题的原因，可以使用两个测试语句（代码中阴影部分的两个语句）来打印出 ptOriginWcs 变量在转换前后的 X 坐标值，发现转换前结果是 107.174182527426，转换后结果是 107.174182527422，两者之间还是有一个微小的差别。这说明在 TranslateCoordinates 方法执行的过程中不可避免地存在一些精度的损失，因此只能再对 ptOriginWcs 进行坐标变换，保证创建 UCS 时 X 轴和 Y 轴的垂直。

(3) 添加一个函数 AddXAngUCS，用于绕 X 轴旋转创建新的 UCS，angle 表示绕当前坐标系（WCS 或 UCS 均可）X 轴旋转的弧度值，其方向由右手定则决定，相关代码为：

```
Public Function AddXAngUCS(angle As Double, strUcsName As String) As AcadUCS
```

```
    '定义新 UCS 原点的三维 WCS 坐标
    Dim ptOriginUcs(0 To 2) As Double
    ptOriginUcs(0) = 0: ptOriginUcs(1) = 0: ptOriginUcs(2) = 0
    Dim ptOriginWcs As Variant
    ptOriginWcs = PtUcs2Wcs(ptOriginUcs)

    '定义新 UCS 在 X 轴正方向上一个点的三维 WCS 坐标
    Dim ptXUcs(0 To 2) As Double
    ptXUcs(0) = 1: ptXUcs(1) = 0: ptXUcs(2) = 0
    Dim ptXWcs As Variant
    ptXWcs = PtUcs2Wcs(ptXUcs)

    '定义新 UCS 在 Y 轴正方向上一个点的三维 WCS 坐标
    Dim ptYUcs(0 To 2) As Double
    ptYUcs(0) = 0: ptYUcs(1) = Cos(angle): ptYUcs(2) = Sin(angle)
    Dim ptYWcs As Variant
    ptYWcs = PtUcs2Wcs(ptYUcs)

    Set AddXAngUCS = ThisDrawing.UserCoordinateSystems.Add(ptOriginWcs, ptXWcs, ptYWcs,
        strUcsName)
End Function
```

注意当前坐标系的两个点(1,0,0)和(0,1,0),在坐标系绕 X 轴旋转一定角度之后,(1,0,0)的坐标不会发生变化,但是(0,1,0)的坐标变成(0,cos(angle),sin(angle))。如果觉得难以理解,可以在纸上画草图辅助理解,也可以在 AutoCAD 图形窗口中绘制三条直线模拟三个坐标轴,并对其进行旋转,观察旋转前后两点坐标的变化。

(4) 添加一个函数 AddYAngUCS,用于将当前坐标系绕 Y 轴旋转一定角度,从而创建新的 UCS,其实现代码为:

```
Public Function AddYAngUCS(angle As Double, strUcsName As String) As AcadUCS
    '定义新 UCS 原点的三维 WCS 坐标
    Dim ptOriginUcs(0 To 2) As Double
    ptOriginUcs(0) = 0: ptOriginUcs(1) = 0: ptOriginUcs(2) = 0
    Dim ptOriginWcs As Variant
    ptOriginWcs = PtUcs2Wcs(ptOriginUcs)

    '定义新 UCS 在 X 轴正方向上一个点的三维 WCS 坐标
    Dim ptXUcs(0 To 2) As Double
    ptXUcs(0) = Cos(angle): ptXUcs(1) = 0: ptXUcs(2) = - Sin(angle)
    Dim ptXWcs As Variant
    ptXWcs = PtUcs2Wcs(ptXUcs)

    '定义新 UCS 在 Y 轴正方向上一个点的三维 WCS 坐标
    Dim ptYUcs(0 To 2) As Double
    ptYUcs(0) = 0: ptYUcs(1) = 1: ptYUcs(2) = 0
    Dim ptYWcs As Variant
```

```
    ptYWcs = PtUcs2Wcs(ptYUcs)

    Set AddYAngUCS = ThisDrawing.UserCoordinateSystems.Add(ptOriginWcs, ptXWcs, ptYWcs,
        strUcsName)
End Function
```

(5) 创建一个新的函数 AddZAngUCS,用于将当前坐标系绕 Z 轴旋转一定角度,从而创建新的 UCS,其实现代码为:

```
Public Function AddZAngUCS(angle As Double, strUcsName As String) As AcadUCS
    '定义新 UCS 原点的三维 WCS 坐标
    Dim ptOriginUcs(0 To 2) As Double
    ptOriginUcs(0) = 0: ptOriginUcs(1) = 0: ptOriginUcs(2) = 0
    Dim ptOriginWcs As Variant
    ptOriginWcs = PtUcs2Wcs(ptOriginUcs)

    '定义新 UCS 在 X 轴正方向上一个点的三维 WCS 坐标
    Dim ptXUcs(0 To 2) As Double
    ptXUcs(0) = Cos(angle): ptXUcs(1) = Sin(angle): ptXUcs(2) = 0
    Dim ptXWcs As Variant
    ptXWcs = PtUcs2Wcs(ptXUcs)

    '定义新 UCS 在 Y 轴正方向上一个点的三维 WCS 坐标
    Dim ptYUcs(0 To 2) As Double
    ptYUcs(0) = - Sin(angle): ptYUcs(1) = Cos(angle): ptYUcs(2) = 0
    Dim ptYWcs As Variant
    ptYWcs = PtUcs2Wcs(ptYUcs)

    Set AddZAngUCS = ThisDrawing.UserCoordinateSystems.Add(ptOriginWcs, ptXWcs, ptYWcs,
        strUcsName)
End Function
```

9.2.4 实例效果

(1) 在 VBA 集成开发环境中,创建一个宏 test_AddOrgUCS,用于测试移动原点创建 UCS 的函数:

```
Sub test_AddOrgUCS( )
    '原点 UCS 调用示例
    Dim myUCS As AcadUCS, NewOrgPt As Variant
    NewOrgPt = ThisDrawing.Utility.GetPoint(, vbCrLf & "请输入新原点:")
    Set myUCS = AddOrgUCS(NewOrgPt, "abc")
    ThisDrawing.ActiveUCS = myUCS
End Sub
```

(2) 创建一个宏 test_AddXAngUCS,用于测试 AddXAngUCS 函数,该函数通过绕 X 轴旋转一定角度创建新的 UCS:

```
Sub test_AddXAngUCS( )
    'X 轴旋转 UCS 调用示例
    Dim myUCS As AcadUCS, Ang As Double
    Ang = ThisDrawing.Utility.GetAngle(, vbCrLf & "请指定绕 X 轴的旋转角度: ")
    Set myUCS = AddXAngUCS(Ang, "abc")
    ThisDrawing.ActiveUCS = myUCS
End Sub
```

(3) 创建一个宏 test_AddYAngUCS,用于测试 AddYAngUCS 函数,该函数通过绕 Y 轴旋转一定角度创建新的 UCS:

```
Sub test_AddYAngUCS( )
    'Y 轴旋转 UCS 调用示例
    Dim myUCS As AcadUCS, Ang As Double
    Ang = ThisDrawing.Utility.GetAngle(, vbCrLf & "请指定绕 Y 轴的旋转角度: ")
    Set myUCS = AddYAngUCS(Ang, "abc")
    ThisDrawing.ActiveUCS = myUCS
End Sub
```

(4) 创建一个宏 test_AddZAngUCS,用于测试 AddZAngUCS 函数,该函数通过绕 Z 轴旋转一定角度创建新的 UCS:

```
Sub test_AddZAngUCS( )
    'Z 轴旋转 UCS 调用示例
    Dim myUCS As AcadUCS, Ang As Double
    Ang = ThisDrawing.Utility.GetAngle(, vbCrLf & "请指定绕 Z 轴的旋转角度: ")
    Set myUCS = AddZAngUCS(Ang, "abc")
    ThisDrawing.ActiveUCS = myUCS
End Sub
```

9.2.5　小结

AutoCAD 使用中,UCS 命令可以根据视图、Z 轴移动距离、对象等创建新的 UCS,读者可以根据本节所介绍的方法,封装新的创建 UCS 的方法。

9.3　在任意的平面上创建图案填充

9.3.1　实例说明

在 AutoCAD 中能够创建填充,一般来说都是在世界坐标系的 XOY 平面上创建填充,但是在某些行业,却需要在三维空间中的某个平面上创建填充来表示一定的含义。本节的实例介绍了在三维空间中任意平面(由三个点来确定的平面)上创建图案填充的方法。

9.3.2　编程思路

1. 创建填充

在 VBA 中,创建三维空间中图案填充的一般步骤是:

(1) 使用 AddHatch 方法创建填充对象。

(2) 使用 AppendOuterLoop 方法为填充对象定义填充的外部边界(之所以说外部边界,是相对于内部边界而言,创建填充时确实可以同时指定外部边界和内部边界,来创建"环形"结构的填充)。

(3) 使用 Elevation 属性来设置填充对象的标高。所谓填充对象的标高,是指填充对象所在的平面到世界坐标系(WCS)原点的距离。在实际应用中,填充对象的标高应该取世界坐标系原点在填充所用的用户坐标系中的 Z 坐标值的负数。

(4) 使用 Evaluate 方法显示填充对象。

2. 根据给定的三点创建 UCS

在创建图案填充时,AutoCAD 总会将填充创建在当前坐标系(无论是 WCS 还是 UCS)的 XOY 平面上,因此,在创建图案填充之前,必须将要创建填充的平面设置为当前坐标系的 XOY 平面。也就是说,要根据指定的平面来创建一个 UCS。

在 AutoCAD 软件中,用户可以通过指定三点来创建一个新的 UCS,这里同样需要实现这个功能。由于是要根据三个点(三个点的 WCS 坐标已知)来确定 UCS 的 XOY 平面,也就是要根据这三点创建一个新的 UCS。

创建一个新的 UCS,可以使用 UserCoordinateSystems 集合的 Add 方法,其定义为:

```
RetVal = object.Add(Origin, XAxisPoint, YAxisPoint, Name)
```

Origin 给出要创建的 UCS 的原点坐标; XaxisPoint 指定要创建的 UCS 的 X 轴正半轴上一点; YaxisPoint 指定要创建的 UCS 的 Y 轴正半轴上一点; Name 指定 UCS 的名称。

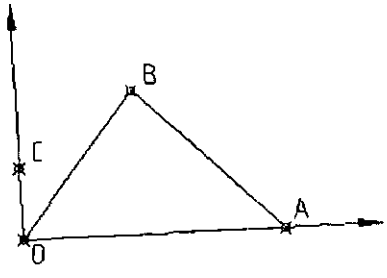


图 9.2 三点创建 UCS 的已知条件

在目前的已知条件下,可以将已知点 O 作为 UCS 的原点,A 作为 UCS 的 X 轴正半轴上一点,缺少的条件是 UCS 的 Y 轴正半轴上一点(目前集中力量要在 UCS 的 Y 轴正半轴上找出一点 C)。可以画出一个草图来描述当前的已知条件,如图 9.2 所示。

将已知条件转化成具体的信息:已知原点 O 的坐标是(x_0, y_0, z_0),X 轴正方向上一点 A 的坐标是(x_1, y_1, z_1),XOY 平面上一点 B(Y 坐标为正)的坐标是(x_2, y_2, z_2),要找出 Y 轴正方向上一点 C 的坐标(x, y, z)。

假设将点 O 移动到 WCS 的原点,则 A、B 两点的坐标变为$(x_1-x_0, y_1-y_0, z_1-z_0)$、$(x_2-x_0, y_2-y_0, z_2-z_0)$。根据立体解析几何的知识,由于 $OA \perp OC$,故

$$(x_1-x_0)\cdot x+(y_1-y_0)\cdot y+(z_1-z_0)\cdot = 0 \tag{1}$$

又因为 O、A、B 和 C 四点共面,故

$$\begin{vmatrix} 0 & 0 & 0 & 1 \\ x_1-x_0 & y_1-y_0 & z_1-z_0 & 1 \\ x_2-x_0 & y_2-y_0 & z_2-z_0 & 1 \\ x-x_0 & y-y_0 & z-z_0 & 1 \end{vmatrix} = 0 \tag{2}$$

由于 C 点是 Y 轴正半轴上任一点,假设其坐标满足

$$y=\frac{1}{2}(y_1+y_2) \tag{3}$$

如果将方程(1)、(2)和(3)结合起来，就可以解出 x、y 和 z 的值，具体的计算过程可以参考 CreateUCS3Pt 函数的实现代码。

需要注意的一点是，这里计算出来 x、y 和 z 的值并不是 C 点的坐标，因为这个计算结果有一个前提，那就是将点 O 移动到 WCS 的原点。因此，C 点在 WCS 下的坐标是($x+x_0$，$y+y_0$，$z+z_0$)。

9.3.3 步骤

(1) 在 VBA 集成开发环境中，创建一个新工程，保存在适当的位置。打开 ThisDrawing 窗口，添加用于创建真彩色填充的函数 AddHatchTC，其实现代码为：

```
Public Function AddHatchTC(ByVal pt1 As Variant, ByVal pt2 As Variant, ByVal pt3 As Variant,
ByVal associativity As Boolean, ByVal color As AcadAcCmColor) As AcadHatch
    On Error GoTo errHandle

    Dim objHatch As AcadHatch
    Dim objPline As Acad3DPolyline        '填充边界
    Dim ptArr(0 To 8) As Double           '构成填充边界的点数组
    Dim objList(0) As AcadEntity          '填充边界实体数组

    '创建一条三维多段线，作为填充边界
    ptArr(0) = pt1(0): ptArr(1) = pt1(1): ptArr(2) = pt1(2)
    ptArr(3) = pt2(0): ptArr(4) = pt2(1): ptArr(5) = pt2(2)
    ptArr(6) = pt3(0): ptArr(7) = pt3(1): ptArr(8) = pt3(2)
    Set objPline = ThisDrawing.ModelSpace.Add3DPoly(ptArr)
    objPline.Closed = True

    '创建真彩色填充
    Set objHatch = ThisDrawing.ModelSpace.AddHatch(acPreDefinedGradient, "LINEAR", True,
        acGradientObject)

    objHatch.GradientColor1 = color
    objHatch.GradientColor2 = color

    Set objList(0) = objPline
    objHatch.AppendOuterLoop (objList)
    Dim ptTemp(0 To 2) As Double, ptOrigin As Variant
    ptTemp(0) = 0: ptTemp(1) = 0: ptTemp(2) = 0
    ptOrigin = ThisDrawing.Utility.TranslateCoordinates(ptTemp, acWorld, acUCS, False)
    '设置填充对象的标高是正确确定填充位置的关键
    objHatch.Elevation = - ptOrigin(2)
    objHatch.Evaluate
    ThisDrawing.Regen True

    objPline.Delete                       '删除填充边界
```

```
    Set AddHatchTC = objHatch
    Exit Function

    '处理填充边界未闭合的错误
errHandle:
    If Err.Number = -2145386493 Then
        MsgBox "填充定义边界未闭合!", vbCritical
    End If
    Err.Clear
End Function
```

该函数包括 5 个输入参数，pt1、pt2 和 pt3 是三个点，用于确定要填充的三角形区域；associativity 指定填充的关联性；color 指定真彩色填充的颜色。

必须明确指出，这个函数与在 WCS 中创建填充的函数不太一样，它可能并不适合在 WCS 中创建填充对象。与在 WCS 中创建填充相比，增加了下面的代码：

```
Dim ptTemp(0 To 2) As Double, ptOrigin As Variant
ptTemp(0) = 0: ptTemp(1) = 0: ptTemp(2) = 0
ptOrigin = ThisDrawing.Utility.TranslateCoordinates(ptTemp, acWorld, acUCS, False)
objHatch.Elevation = -ptOrigin(2)
```

这几句代码的作用是设置填充对象的标高，正如前面所说，标高的取值应该是世界坐标系原点在填充所用的用户坐标系中的 Z 坐标值的负数。

(2) 创建一个函数 CreateUCS3Pt，用于根据指定的三点创建一个 UCS，保证指定的三点都在 UCS 的 XOY 平面上，其实现代码为：

```
Private Function CreateUCS3Pt(pt1 As Variant, pt2 As Variant, pt3 As Variant, strUCS As String,
    ByRef objUCS As AcadUCS) As Boolean
    '如果 pt1 和 pt2 重合
    If pt1(0) = pt2(0) And pt1(1) = pt2(1) And pt1(2) = pt2(2) Then
        CreateUCS3Pt = False
    End If

    Dim a1 As Double, b1 As Double, c1 As Double
    Dim a2 As Double, b2 As Double, c2 As Double
    Dim x As Double, y As Double, z As Double
    y = (pt3(1) + pt2(1))/2

    a1 = pt2(0) - pt1(0)
    b1 = pt2(2) - pt1(2)
    c1 = (pt1(1) - pt2(1)) * y

    a2 = ((pt2(1) - pt1(1)) * (pt3(2) - pt1(2))) - ((pt3(1) - pt1(1)) * (pt2(2) - pt1(2)))
    b2 = (a1 * (pt3(1) - pt1(1))) - ((pt2(1) - pt1(1)) * (pt3(0) - pt1(0)))
    c2 = (a1 * (pt3(2) - pt1(2)) - b1 * (pt3(0) - pt1(0))) * y
```

```
    '如果三点共线或重合
    If (a1 * b2 - a2 * b1) = 0 Then
        CreateUCS3Pt = False
    End If

    x = (c1 * b2 - c2 * b1)/(a1 * b2 - a2 * b1)
    z = (c2 * a1 - c1 * a2)/(a1 * b2 - a2 * b1)

    '将假定的偏移量加在坐标上
    Dim ptY(0 To 2) As Double
    ptY(0) = x + pt1(0): ptY(1) = y + pt1(1): ptY(2) = z + pt1(2)

    Set objUCS = ThisDrawing.UserCoordinateSystems.Add(pt1, pt2, ptY, strUCS)
    CreateUCS3Pt = True
End Function
```

该函数的原理已在编程思路中介绍，这里就不再详细分析。

(3) 创建一个宏 CreateHatch，用于提示用户输入三个点，并且填充这三个点所组成的三角形区域，其实现代码为：

```
Public Sub CreateHatch( )
    '提示用户输入要确定填充边界的三点
    Dim pt1 As Variant, pt2 As Variant, pt3 As Variant
    pt1 = ThisDrawing.Utility.GetPoint(, "拾取第一点：")
    pt2 = ThisDrawing.Utility.GetPoint(, "拾取第二点：")
    pt3 = ThisDrawing.Utility.GetPoint(, "拾取第三点：")

    '在填充之前，创建一个 UCS，其 XOY 平面在三点确定的平面上
    Dim objUCS As AcadUCS
    If Not CreateUCS3Pt(pt1, pt2, pt3, "填充", objUCS) Then
        Exit Sub
    End If
    ThisDrawing.ActiveUCS = objUCS        '将新建的 UCS 设置为当前 UCS

    '创建真彩色的填充
    Dim color As New AcadAcCmColor
    color.ColorIndex = acMagenta
    AddHatchTC pt1, pt2, pt3, True, color

    '设置当前的坐标系为世界坐标系
    ThisDrawing.SendCommand "UCS "
End Sub
```

9.3.4　实例效果

(1) 在 AutoCAD 中，选择“绘图”|“实体”|“长方体”菜单项，在图形窗口中创建一个长方体。

(2) 在 VBA 集成开发环境中,运行 CreateHatch 宏,按照命令行的提示,任意选择长方体的三个顶点,能够得到如图 9.3 所示的结果。

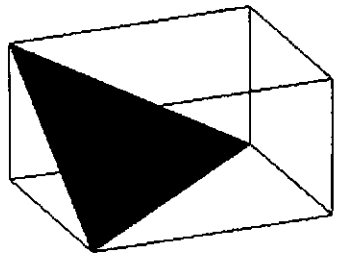


图 9.3 填充给定的三角形区域

9.3.5 小结

学习本节内容之后,需要掌握下面的知识点:

- 创建真彩色填充的方法。
- 根据三点创建 UCS。

9.4 创建正多面体

9.4.1 实例说明

本节的实例能够根据用户输入的棱长,创建几种常用的正多面体,包括正四面体、正六面体、正八面体、正十二面体和正二十面体。

9.4.2 编程思路

创建正四面体,只须以一个等边三角形为拉伸截面,沿其法线方向进行拉伸,并设置一个合适的拉伸角度即可,其关键问题在于计算拉伸角度。

创建正六面体(正方体),可以一个正方形作拉伸截面,沿其法线方向进行拉伸,将正方形的边长作为拉伸高度即可。

创建正八面体,可以将其分成对称的两个部分来考虑,每一部分都是一个四面体,同样可以以一个正方形为拉伸截面,沿法线方向进行拉伸,设置一个适当的拉伸角度。获得一个四面体之后,进行镜像操作得到另一半,然后对两部分实体进行并集。

创建正十二面体,可以分成对称的两个部分来考虑,每个部分都可以由一个正五边形作拉伸截面,设置一定的拉伸角度,进行拉伸得到一个实体,然后对其进行三维镜像、旋转、移动等操作,最终对两个部分进行交集。

创建正二十面体,可以首先对一个正五边形面域进行拉伸,获得一个棱锥(正二十面体的上部)和一个棱台;对棱台进行三维镜像,生成另一个棱台,再将原棱台删除;将上一步生成的棱台再进行一次三维镜像,将生成的棱台旋转 36°,再和源棱台进行交集运算,生成正二十面体的中部;将棱锥进行三维镜像,生成一个新棱锥,并将其旋转 36°,生成正二十面体的下部;最后将三部分进行并集,完成正二十面体。

9.4.3 步骤

(1) 在 AutoCAD 2005 中,打开 VBA 管理器,创建一个新工程,保存在适当的位置。进入 VBA 集成开发环境,选择"插入"|"用户窗体"菜单项,向工程中添加一个窗体,将其名称修改为 frmMain,并在其中添加如图 9.4 所示的控件。

其中,上部是一个多页控件,每一页都包含一个 Image 控件,Image 控件的 Picture 属性

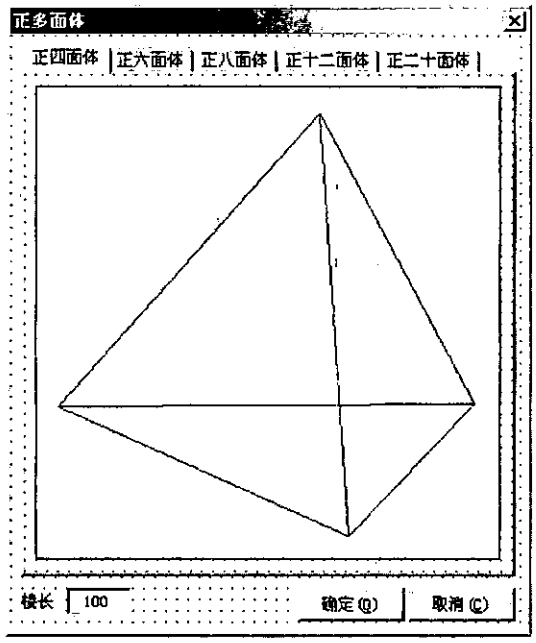


图 9.4　设计窗体

设置为与本页标题对应的正多面体图形(这些图形都在配套光盘中相关章节的文件夹中),例如,“正四面体”页面内显示的图像是“正四面体. wmf”。文本框的名称修改为 txtRidgeLength,两个命令按钮的名称修改为 cmdOk 和 cmdCancel。

(2) 在窗体的代码窗口中,添加下面几个模块级变量:

```
Dim ridgeLength As Double      '多面体的棱长
Dim pt1(2) As Double, pt2(2) As Double, pt3(2) As Double
Dim objRegion As Variant       '创建多面体时的临时面域

Const PI As Double = 3.1415926535897
```

(3) 添加一个函数 Rad2angle,用于将弧度转化为角度,其实现代码为:

```
Private Function Rad2angle(angle As Double) As Double
    Rad2angle = angle * PI/180
End Function
```

(4) 添加一个函数 PtUcs2Wcs,用于将点的坐标从 UCS 转化到 WCS,其实现代码为:

```
Private Function PtUcs2Wcs(ptUcs As Variant) As Variant
    PtUcs2Wcs = ThisDrawing.Utility.TranslateCoordinates(ptUcs, acUCS, acWorld, False)
End Function
```

(5) 添加一个函数 AddLWPolyline,用于在 UCS 中使用 OCS 方法创建优化多段线,其实现代码为:

```
Private Function AddLWPolyline(pointsUcs As Variant) As AcadLWPolyline
    Dim xDir, yDir As Variant
```

```
    xDir = ThisDrawing.GetVariable("UCSXDIR")
    yDir = ThisDrawing.GetVariable("UCSYDIR")

    '得到 Z 轴向量
    Dim zDir(2) As Double
    zDir(0) = xDir(1) * yDir(2) - yDir(1) * xDir(2)
    zDir(1) = xDir(2) * yDir(0) - yDir(2) * xDir(0)
    zDir(2) = xDir(0) * yDir(1) - yDir(0) * xDir(1)

    '用 OCS 方法创建优化多段线
    Dim n As Integer '顶点数
    n = (UBound(pointsUcs) + 1)/2

    '获得 OCS 下的多段线坐标数组
    Dim pointsOcs( ) As Double
    ReDim pointsOcs(UBound(pointsUcs))
    Dim ptUcs(2) As Double, ptOcs As Variant        '临时点的 UCS 和 OCS 坐标
    Dim i As Integer

    For i = 0 To n - 1
        ptUcs(0) = pointsUcs(2 * i)
        ptUcs(1) = pointsUcs(2 * i + 1)
        ptUcs(2) = 0
        ptOcs = ThisDrawing.Utility.TranslateCoordinates(ptUcs, acUCS, acOCS, False, zDir)
        pointsOcs(2 * i) = ptOcs(0)
        pointsOcs(2 * i + 1) = ptOcs(1)
    Next

    Set AddLWPolyline = ThisDrawing.ModelSpace.AddLightWeightPolyline(pointsOcs)
    AddLWPolyline.Elevation = ptOcs(2)              '修改标高
End Function
```

(6) 在窗体中“确定”按钮的单击事件中,根据当前显示的多页控件的页面,创建与其对应的正多面体,其实现代码为:

```
Private Sub cmdOk_Click( )
    '用户输入非数值时,发出提示信息
    If Not (IsNumeric(txtRidgeLength.Text)) Then
        MsgBox ("输入的数据有误,请重新输入!")
        Exit Sub
    Else
        ridgeLength = txtRidgeLength.Text
    End If

    '用户输入负数或零时,发出提示信息
    If ridgeLength <= 0 Then
```

```
        MsgBox ("输入的数据必须是正数,请重新输入!")
        Exit Sub
    End If

    Dim intCount As Integer     '当前模型空间的实体数量
    intCount = ThisDrawing.ModelSpace.count

    frmMain.Hide     '开始绘图之前隐藏窗体,将焦点交给 AutoCAD
    Select Case MultiPage1.Value
        Case 0 '正四面体
            Call AddTetrahedron
        Case 1 '正六面体
            Call AddHexahedron
        Case 2 '正八面体
            Call AddOctahedron
        Case 3 '正十二面体
            Call AddDodecahedron
        Case 4 '正二十面体
            Call AddIcosahedron
    End Select

    '删除已无用的面域
    Dim ent As AcadEntity
    If ThisDrawing.ModelSpace.count > intCount Then
        Set ent = ThisDrawing.ModelSpace.Item(ThisDrawing.ModelSpace.count - 2)
        ent.Delete
    End If

    ThisDrawing.Regen acActiveViewport
    ZoomExtents

    End
End Sub
```

(7) 添加一个过程 AddTetrahedron,用于根据棱长创建正四面体,其实现代码为:

```
Private Sub AddTetrahedron( )
    '以原点为中心创建等边三角形
    Dim points(5) As Double
    points(0) = - 0.5 * ridgeLength
    points(1) = - 0.5 * ridgeLength/Sqr(3)
    points(2) = 0.5 * ridgeLength
    points(3) = - 0.5 * ridgeLength/Sqr(3)
    points(4) = 0
    points(5) = ridgeLength/Sqr(3)
```

```
    Dim objPoly(0) As AcadLWPolyline
    Set objPoly(0) = AddLWPolyline(points)
    objPoly(0).Closed = True

    '创建面域
    objRegion = ThisDrawing.ModelSpace.AddRegion(objPoly)
    objPoly(0).Delete

    '拉伸面域
    Din angle As Double        '拉伸角度
    angle = Atn(1/Sqr(8))

    Dim height As Double       '拉伸高度
    height = ridgeLength * Sqr(2/3)

    Dim objSolid As Acad3DSolid
    Set objSolid = ThisDrawing.ModelSpace.AddExtrudedSolid(objRegion(0), height, angle)
End Sub
```

(8) 添加一个过程 AddHexahedron,用于根据棱长创建正六面体,其实现代码为:

```
Private Sub AddHexahedron( )
    '以原点为中心创建正方形
    Dim points(7) As Double
    points(0) = -0.5 * ridgeLength
    points(1) = -0.5 * ridgeLength
    points(2) = 0.5 * ridgeLength
    points(3) = -0.5 * ridgeLength
    points(4) = 0.5 * ridgeLength
    points(5) = 0.5 * ridgeLength
    points(6) = -0.5 * ridgeLength
    points(7) = 0.5 * ridgeLength

    Dim objPoly(0) As AcadLWPolyline
    Set objPoly(0) = AddLWPolyline(points)
    objPoly(0).Closed = True

    '创建面域
    objRegion = ThisDrawing.ModelSpace.AddRegion(objPoly)
    objPoly(0).Delete

    '拉伸面域
    Dim angle As Double     '拉伸角度
    angle = 0

    Dim height As Double    '拉伸高度
```

```
    height = ridgeLength

    Dim objSolid As Acad3DSolid
    Set objSolid = ThisDrawing.ModelSpace.AddExtrudedSolid(objRegion(0), height, angle)
End Sub
```

(9) 添加一个过程 AddOctahedron,用于根据棱长创建正八面体,其实现代码为:

```
Private Sub AddOctahedron( )
    '以原点为中心创建正方形
    Dim points(7) As Double
    points(0) = -0.5 * ridgeLength
    points(1) = -0.5 * ridgeLength
    points(2) = 0.5 * ridgeLength
    points(3) = -0.5 * ridgeLength
    points(4) = 0.5 * ridgeLength
    points(5) = 0.5 * ridgeLength
    points(6) = -0.5 * ridgeLength
    points(7) = 0.5 * ridgeLength

    Dim objPoly(0) As AcadLWPolyline
    Set objPoly(0) = AddLWPolyline(points)
    objPoly(0).Closed = True

    '创建面域
    objRegion = ThisDrawing.ModelSpace.AddRegion(objPoly)
    objPoly(0).Delete

    '拉伸面域
    Dim angle As Double    '拉伸角度
    angle = Atn(Sqr(2)/2)

    Dim height As Double    '拉伸高度
    height = ridgeLength * Sqr(2)/2

    Dim objSolid1 As Acad3DSolid
    Set objSolid1 = ThisDrawing.ModelSpace.AddExtrudedSolid(objRegion(0), height, angle)

    '定义三维镜像的轴线
    pt1(0) = 0: pt1(1) = 0: pt1(2) = 0
    pt2(0) = 0: pt2(1) = 1: pt2(2) = 0
    pt3(0) = 1: pt3(1) = 0: pt3(2) = 0

    Dim objSolid2 As Acad3DSolid
    Set objSolid2 = objSolid1.Mirror3D(PtUcs2Wcs(pt1), PtUcs2Wcs(pt2), PtUcs2Wcs(pt3))
```

```
    '并集
    objSolid1.Boolean acUnion, objSolid2
End Sub
```

(10) 添加一个过程 AddOctahedron,用于根据棱长创建正十二面体,其实现代码为:

```
Private Sub AddDodecahedron( )
    '以原点为中心画一正五边形
    Dim points(9) As Double
    points(0) = - 0.5 * ridgeLength
    points(1) = - 0.5 * ridgeLength * Tan(Rad2angle(54))
    points(2) = 0.5 * ridgeLength
    points(3) = points(1)
    points(4) = ridgeLength * (0.5 + Cos(Rad2angle(72)))
    points(5) = ridgeLength * (Sin(Rad2angle(72)) - 0.5 * Tan(Rad2angle(54)))
    points(6) = 0
    points(7) = 0.5 * ridgeLength/Cos(Rad2angle(54))
    points(8) = - points(4)
    points(9) = points(5)

    Dim objPolygon(0) As AcadLWPolyline
    Set objPolygon(0) = AddLWPolyline(points)
    objPolygon(0).Closed = True

    '创建面域
    objRegion = ThisDrawing.ModelSpace.AddRegion(objPolygon)
    objPolygon(0).Delete

    '拉伸面域
    Dim tmp As Double
    tmp = Tan(Rad2angle(54))/Tan(Rad2angle(72))

    Dim angle As Double     '拉伸角度
    angle = Atn( - tmp/Sqr( - tmp * tmp + 1)) + 2 * Atn(1) - 0.5 * PI

    Dim height As Double    '拉伸高度
    height = ridgeLength * Tan(Rad2angle(54)) * Tan( - 0.5 * angle + 0.25 * PI)

    Dim objSolid1 As Acad3DSolid
    Set objSolid1 = ThisDrawing.ModelSpace.AddExtrudedSolid(objRegion(0), height, angle)

    '三维镜像、旋转、移动、交集
    pt1(0) = 0: pt1(1) = 0: pt1(2) = 0
    pt2(0) = 0: pt2(1) = 1: pt2(2) = 0
    pt3(0) = 1: pt3(1) = 0: pt3(2) = 0
```

```
    Dim objSolid2 As Acad3DSolid
    Set objSolid2 = objSolid1.Mirror3D(PtUcs2Wcs(pt1), PtUcs2Wcs(pt2), PtUcs2Wcs(pt3))
    objSolid2.Rotate PtUcs2Wcs(pt1), Rad2angle(36)

    pt2(0) = 0: pt2(1) = 0: pt2(2) = height
    objSolid2.Move PtUcs2Wcs(pt1), PtUcs2Wcs(pt2)
    objSolid1.Boolean acIntersection, objSolid2
End Sub
```

(11) 添加一个过程 AddIcosahedron，用于根据棱长创建正二十面体，其实现代码为：

```
Private Sub AddIcosahedron( )
    '以原点为中心画一正五边形
    Dim points(9) As Double
    points(0) = -0.5 * ridgeLength
    points(1) = -0.5 * ridgeLength * Tan(Rad2angle(54))
    points(2) = 0.5 * ridgeLength
    points(3) = points(1)
    points(4) = ridgeLength * (0.5 + Cos(Rad2angle(72)))
    points(5) = ridgeLength * (Sin(Rad2angle(72)) - 0.5 * Tan(Rad2angle(54)))
    points(6) = 0
    points(7) = 0.5 * ridgeLength/Cos(Rad2angle(54))
    points(8) = -points(4)
    points(9) = points(5)

    Dim objPolygon(0) As AcadLWPolyline
    Set objPolygon(0) = AddLWPolyline(points)
    objPolygon(0).Closed = True

    '创建面域
    objRegion = ThisDrawing.ModelSpace.AddRegion(objPolygon)
    objPolygon(0).Delete

    '拉伸面域，生成一个棱锥(正二十面体的上部)和一个棱台
    Dim angle1 As Double, height1 As Double     '拉伸角度 1 和拉伸高度 1
    Dim tmp As Double
    tmp = 0.5 * Tan(Rad2angle(54))/Sin(Rad2angle(60))
    angle1 = Atn(tmp/Sqr(-tmp * tmp + 1))
    height1 = 0.5 * ridgeLength * Tan(Rad2angle(54))/Tan(angle1)

    Dim Angle2 As Double, height2 As Double     '拉伸角度 2 和拉伸高度 2
    tmp = (1 - Sin(Rad2angle(54)))/(Cos(Rad2angle(54)) * Tan(Rad2angle(60)))
    Angle2 = -Atn(tmp/Sqr(-tmp * tmp + 1))
    height2 = ridgeLength * Sin(Rad2angle(60)) * Cos(Angle2)

    Dim objSolid1 As Acad3DSolid
    Set objSolid1 = ThisDrawing.ModelSpace.AddExtrudedSolid(objRegion(0), height1, angle1)
```

```
    Dim SolidtmpObj As Acad3DSolid
    Set SolidtmpObj = ThisDrawing.ModelSpace.AddExtrudedSolid(objRegion(0), height2, Angle2)

    '对棱台进行三维镜像,生成另一个棱台,再将原棱台删除
    pt1(0) = 0: pt1(1) = 0: pt1(2) = 0
    pt2(0) = 0: pt2(1) = 1: pt2(2) = 0
    pt3(0) = 1: pt3(1) = 0: pt3(2) = 0

    Dim objSolid2 As Acad3DSolid
    Set objSolid2 = SolidtmpObj.Mirror3D(PtUcs2Wcs(pt1), PtUcs2Wcs(pt2), PtUcs2Wcs(pt3))
    '删除三维镜像的源对象
    SolidtmpObj.Delete

    '将上一步生成的棱台再进行一次三维镜像
    '将生成的棱台旋转 36°,再和原棱台进行交集运算,生成正二十面体的中部
    pt1(0) = 0: pt1(1) = 0: pt1(2) = -0.5 * height2
    pt2(0) = 1: pt2(1) = 0: pt2(2) = -0.5 * height2
    pt3(0) = 0: pt3(1) = 1: pt3(2) = -0.5 * height2
    Set SolidtmpObj = objSolid2.Mirror3D(PtUcs2Wcs(pt1), PtUcs2Wcs(pt2), PtUcs2Wcs(pt3))

    Din pt4(2) As Double
    pt4(0) = 0: pt4(1) = 0: pt4(2) = - height2
    SolidtmpObj.Rotate PtUcs2Wcs(pt4), Rad2angle(36)
    objSolid2.Boolean acIntersection, SolidtmpObj

    '将棱锥进行三维镜像,生成一个新棱锥,并将其旋转 36°,生成正二十面体的下部
    Dim objSolid3 As Acad3DSolid
    Set objSolid3 = objSolid1.Mirror3D(PtUcs2Wcs(pt1), PtUcs2Wcs(pt2), PtUcs2Wcs(pt3))
    objSolid3.Rotate PtUcs2Wcs(pt4), Rad2angle(36)

    '将三部分进行并集,完成正二十面体
    objSolid1.Boolean acUnion, objSolid2
    objSolid1.Boolean acUnion, objSolid3
End Sub
```

(12) 在窗体中"取消"按钮的单击事件中,添加退出程序执行的代码,其相关代码为:

```
Private Sub cmdExit_Click( )
    End
End Sub
```

(13) 双击"工程资源管理器"窗口中的 ThisDrawing 图标,打开其代码窗口,在其中添加用于启动应用程序的宏:

```
Public Sub Main( )
    frmMain.Show
End Sub
```

9.4.4　实例效果

(1) 在 VBA 集成开发环境中，执行 Main 宏，系统会弹出如图 9.5 所示的对话框。

(2) 在“棱长”文本框中输入适当的数值，单击“确定”按钮，能够在图形窗口得到如图 9.6 所示的结果。

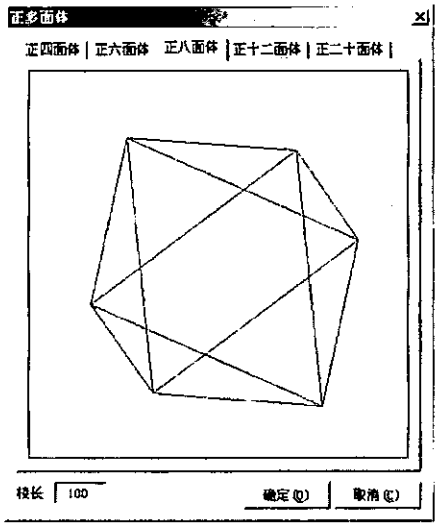


图 9.5　程序运行主界面

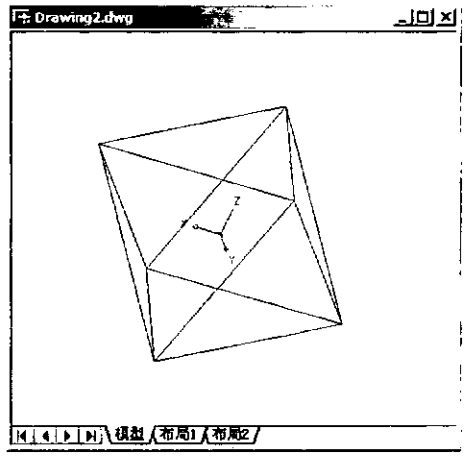


图 9.6　创建正八面体的结果

其他正多面体的创建读者可自行验证，在此不过多赘述。

9.4.5　小结

相信这个程序对提高空间想象能力和数学运算能力大有帮助，学习该程序之后，应该掌握下面的知识点：

- 窗体输入参数时的错误处理。
- 在 UCS 中拉伸、移动、旋转、三维镜像、并集、交集等三维建模及编辑方法的使用。
- 删除图形中多余的面域的方法。

9.5　创建偏心圆台、圆柱和圆锥

9.5.1　实例说明

在 AutoCAD 中，有一些形状比较特殊的实体无法直接创建出来，有必要通过编程来解决，并且简化手工操作的繁琐步骤。

本节要介绍的实例能够根据用户输入的参数，创建偏心的圆台、圆柱或圆锥，如图 9.7 所示。

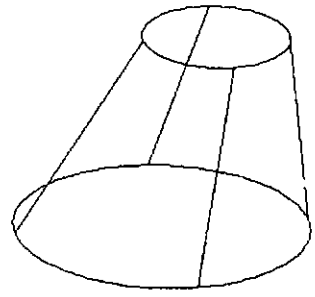


图 9.7 偏心圆台的形状

9.5.2 编程思路

要想完成创建三维实体的程序，必须首先能在 AutoCAD 中手工将其创建出来。这里以斜圆锥的创建为例，展示本节程序最核心的算法。

已知条件：顶点 P(300,0,400)，底面椭圆长轴的两个端点 A(−100,0,0)和 B(100,0,0)，其位置如图 9.8 所示。

下面介绍创建斜圆锥的手工操作步骤：

(1) 使用 LINE 命令，创建连接 PA、PB 的直线，如图 9.9 所示。

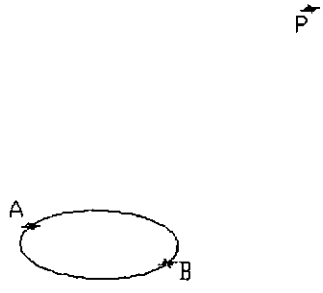


图 9.8 创建斜圆锥的已知条件

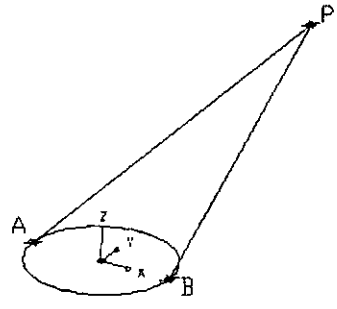


图 9.9 连接 PA 和 PB

(2) 将 PA 和 PB 延长，使其长度变为原来的两倍。选择“修改”|“拉长”菜单项，按照命令提示进行操作：

```
命令：_lengthen
选择对象或 [增量(DE)/百分数(P)/全部(T)/动态(DY)]：p
输入长度百分数 <100.0000>：200
选择要修改的对象或 [放弃(U)]：选择 PA 直线上靠近 A 的一点
选择要修改的对象或 [放弃(U)]：选择 PB 直线上靠近 B 的一点
选择要修改的对象或 [放弃(U)]：按下 Enter 键结束命令
```

完成操作后，得到如图 9.10 所示的结果。

(3) 使用 LINE 命令，创建连接 CD 的直线。使用 CIRCLE 命令，以 CD 为直径，创建一个圆，得到如图 9.11 所示的结果。

(4) 做∠CPD 的角平分线。单击“绘图”工具栏中的“构造线”按钮，按照命令提示进行操作：

```
命令：_xline
指定点或 [水平(H)/垂直(V)/角度(A)/二等分(B)/偏移(O)]：b
指定角的顶点：拾取 P 点
指定角的起点：在直线 PC 上任意拾取一点
指定角的端点：在直线 PD 上任意拾取一点
指定角的端点：按下 Enter 键结束命令
```

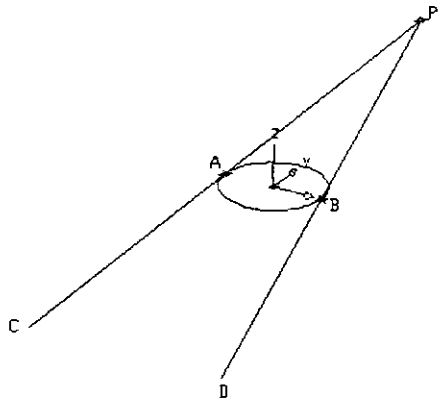


图 9.10　拉长直线

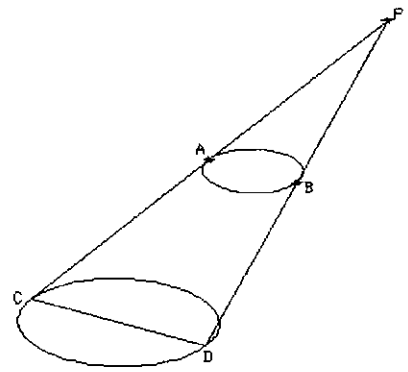


图 9.11　连接直线 CD 并创建圆

(5) 过步骤(4)创建直线和直线 CD 的交点，做直线 CD 的垂线。既可以使用构造线的“角度”选项，也可以直接使用极轴追踪创建该垂线。完成操作后，得到如图 9.12 所示的结果。

(6) 创建新的 UCS。执行 UCS 命令，按照命令提示进行操作：

命令：ucs
当前 UCS 名称：*没有名称*
输入选项 [新建(N)/移动(M)/正交(G)/上一个(P)/恢复(R)/保存(S)/删除(D)/应用(A)/?/世界(W)] <世界>：n
指定新 UCS 的原点或 [Z 轴(ZA)/三点(3)/对象(OB)/面(F)/视图(V)/X/Y/Z] <0,0,0>：3
指定新原点 <0,0,0>：拾取 O 点
在正 X 轴范围上指定点 <-245.9644,-299.2111,-293.2455>：拾取直线 PO 上任意点
在 UCS XY 平面的正 Y 轴范围上指定点 <-247.7200,-298.5560,-293.2455>：拾取直线 OC 上任意点
忽略倾斜、不等比例的对象。

(7) 使用 TRIM 命令，剪去直线 OP 两点之外的部分，由构造线对象获得一个直线对象。使用构造线的“角度”选项，过点 O 做直线 OP 的垂线，交直线 PC 于点 G，如图 9.13 所示。

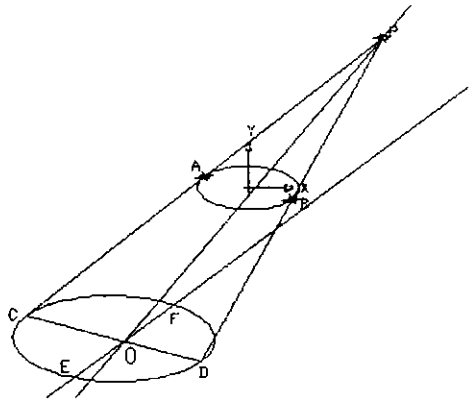


图 9.12　创建直线 CD 的垂线

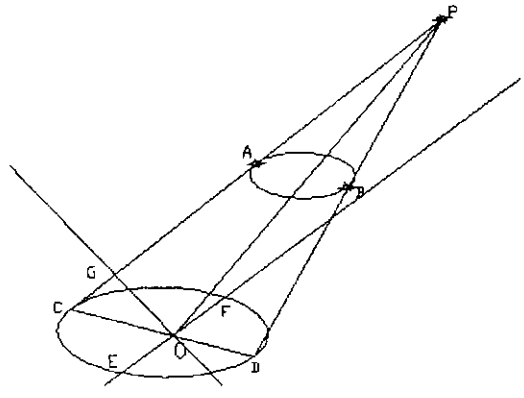


图 9.13　创建直线 PO 的垂线

(8) 使用 UCS 命令的“三点”法，以 O 为原点、OF 为 X 轴、OG 为 Y 轴，创建一个新的 UCS。选择“绘图”|“实体”|“圆锥体”菜单项，按照命令提示进行操作：

```
命令：_cone
当前线框密度：ISOLINES = 4
指定圆锥体底面的中心点或 [椭圆(E)] <0,0,0>：e
指定圆锥体底面椭圆的轴端点或 [中心点(C)]：c
指定圆锥体底面椭圆的中心点 <0,0,0>：【捕捉 O 点】
指定圆锥体底面椭圆的轴端点：【捕捉 F 点】
指定圆锥体底面的另一个轴的长度：【捕捉 G 点】
指定圆锥体高度或 [顶点(A)]：a
指定顶点：【捕捉 P 点】
```

完成操作后，得到如图 9.14 所示的结果。

(9) 选择“绘图”|“实体”|“剖切”菜单项，按照命令提示进行操作：

```
命令：SLICE
选择对象：找到 1 个
选择对象：
指定切面上的第一个点，依照 [对象(O)/Z 轴(Z)/视图(V)/XY 平面(XY)/YZ 平面(YZ)/ZX 平面(ZX)/
    三点(3)] <三点>：o
选择圆、椭圆、圆弧、二维样条曲线或二维多段线：选择以 AB 为直径的圆
在要保留的一侧指定点或 [保留两侧(B)]：b
```

完成操作后，删除辅助线，得到如图 9.15 所示的结果。

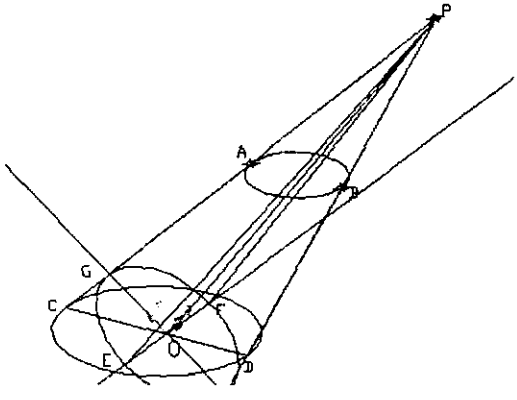


图 9.14 创建圆锥

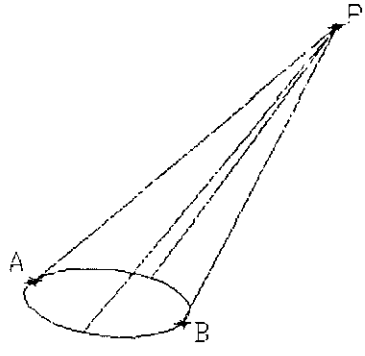


图 9.15 获得斜圆锥

程序设计的思路和上面的绘图方法有一个区别：手工绘制，是在一个新的 UCS 中画椭圆锥，再进行剖切来生成偏心圆锥。程序则是在同一个 UCS 中，生成椭圆锥，再进行旋转，最后用剖切来得到偏心圆锥。

9.5.3 步骤

(1) 在 AutoCAD 2005 中，打开 VBA 管理器，创建一个新工程，保存在适当的位置。进入 VBA 集成开发环境，选择“插入”|“用户窗体”菜单项，向工程中添加一个对话框，并在其中添加如图 9.16 所示的控件。

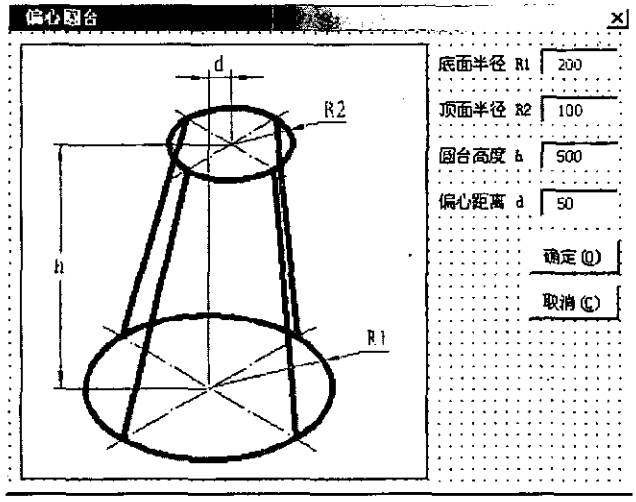


图 9.16　设计窗体

其中，左侧是一个 Image 控件，在属性窗口中将其 Picture 属性设置为配套光盘中的“偏心圆台. wmf”文件；4 个文本框的名称分别为 txtRadiusBottom、txtRadiusTop、txtHeight 和 txtOffset，其 Text 属性分别设置为 200、100、500 和 50；两个命令按钮的名称分别为 cmdOk 和 cmdCancel。

(2) 打开窗体的代码窗口，在其中添加几个公有变量的定义：

```
Dim ptCenter(2) As Double      '偏心圆台底面中心
Dim radiusBottom As Double     '偏心圆台底面半径
Dim radiusTop As Double        '偏心圆台顶面半径
Dim height As Double           '偏心圆台高度
Dim offset As Double           '偏心圆台偏移距离

Const PI As Double = 3.1415926535897
```

(3) 窗体中“确定”按钮的单击事件，用于根据用户输入的参数创建偏心圆台、圆锥或圆柱，其实现代码为：

```
Private Sub cmdOk_Click( )
    '用户输入非数值时，发出提示信息
    If Not (IsNumeric(txtRadiusBottom.Text)) Or Not (IsNumeric(txtRadiusBottom.Text)) _
        Or Not (IsNumeric(txtHeight.Text)) Or Not (IsNumeric(txtOffset.Text)) Then
        MsgBox ("您输入的数据有误，请重新输入!")
        Exit Sub
    Else
        radiusBottom = txtRadiusBottom.Text
        radiusTop = txtRadiusTop.Text
        height = txtHeight.Text
        offset = txtOffset.Text
    End If
```

```
    '用户输入负数时,发出提示信息
    If radiusBottom < 0 Or radiusTop < 0 Or height < 0 Or offset < 0 Then
        MsgBox ("您输入的数据不能为负数,请重新输入!")
        Exit Sub
    End If

    frmMain.Hide        '开始绘图前隐藏窗体,把焦点交给 AutoCAD

    On Error Resume Next

    Dim pt As Variant
    pt = ThisDrawing.Utility.GetPoint(, vbCrLf & "请输入偏心圆台的底面中心点<0,0,0>: ")
    If Err Then
        Err.Clear
        ptCenter(0) = 0: ptCenter(1) = 0: ptCenter(2) = 0
    Else
        pt = ThisDrawing.Utility.TranslateCoordinates(pt, acWorld, acUCS, False)
        ptCenter(0) = pt(0): ptCenter(1) = pt(1): ptCenter(2) = pt(2)
    End If

    '调用子程序
    If radiusBottom <> radiusTop Then
        Call PX_Yuantai        '上下底半径不等,绘制偏心圆台
    Else
        Call PX_Yuanzhu        '上下底半径相同,绘制偏心圆柱
    End If

    '更改视口的观察方向
    Dim newDirection(0 To 2) As Double
    newDirection(0) = 1
    newDirection(1) = -1
    newDirection(2) = 1
    ThisDrawing.ActiveViewport.Direction = newDirection

    ZoomExtents

    End
End Sub
```

(4) 在窗体代码窗口中添加一个过程 PX_Yuanzhu,用于绘制偏心圆,其实现代码为:

```
Private Sub PX_Yuanzhu( )
    Dim intCount As Integer                                    '当前模型空间的对象数
    intCount = ThisDrawing.ModelSpace.Count

    Dim objCircle(0) As AcadEntity
```

```
    Set objCircle(0) = AddCircle(ptCenter, radiusBottom)        '创建拉伸截面

    '创建面域
    Dim objRegion As Variant
    objRegion = ThisDrawing.ModelSpace.AddRegion(objCircle)
    objCircle(0).Delete

    Dim objLine As AcadLine, pt(2) As Double
    pt(0) = ptCenter(0) + offset
    pt(1) = ptCenter(1)
    pt(2) = ptCenter(2) + height
    Set objLine = Addline(ptCenter, pt)                         '创建拉伸路径

    '沿路径拉伸面域
    Dim objSolid As Acad3DSolid
    Set objSolid = ThisDrawing.ModelSpace.AddExtrudedSolidAlongPath(objRegion(0), objLine)
    objLine.Delete

    '删除已无用的面域
    Dim ent As AcadEntity
    If ThisDrawing.ModelSpace.Count > intCount Then
        Set ent = ThisDrawing.ModelSpace.Item(ThisDrawing.ModelSpace.Count - 2)
        ent.Delete
    End If
End Sub
```

其中，AddLine 和 AddCircle 是两个自定义函数。AddLine 函数用于根据指定的起点和终点(UCS 坐标)，创建一条直线，其实现代码为：

```
Private Function AddLine(ptStart As Variant, ptEnd As Variant) As AcadLine
    '将直线起点、端点的 UCS 坐标转换成 WCS 坐标
    Dim ptStWcs, ptEnWcs As Variant
    ptStWcs = PtUcs2Wcs(ptStart)
    ptEnWcs = PtUcs2Wcs(ptEnd)

    Set AddLine = ThisDrawing.ModelSpace.AddLine(ptStWcs, ptEnWcs)
End Function
```

AddCircle 函数用于根据指定的圆心坐标(UCS 坐标)和半径，创建一个圆，其实现代码为：

```
Private Function AddCircle(ptCenter As Variant, radius As Double) As AcadCircle
    '将圆心的 UCS 坐标转换成 WCS 坐标
    Dim ptCenWcs As Variant
    ptCenWcs = PtUcs2Wcs(ptCenter)

    Set AddCircle = ThisDrawing.ModelSpace.AddCircle(ptCenWcs, radius)
```

```
End Function
```

(5) 添加一个过程 PX_Yuantai,用于创建偏心圆台,其实现代码为:

```
Private Sub PX_Yuantai( )
    Dim r1 As Double, r2 As Double
    '将 radiusBottom 和 radiusTop 中比较大的设为 r1
  If radiusBottom > radiusTop Then
        r1 = radiusBottom
        r2 = radiusTop
    Else
        r1 = radiusTop
        r2 = radiusBottom
    End If

    '以下椭圆参数的计算以此为依据,进行椭圆锥有关参数的计算
    Dim dd As Double, h1 As Double
    h1 = height * r1/(r1 - r2)
    dd = h1 * offset/height

    Dim point0(2) As Double, point1(2) As Double, point2(2) As Double, point3(2) As Double
    point0(0) = dd + ptCenter(0)
    point0(1) = ptCenter(1)
    point0(2) = h1 + ptCenter(2)

    point1(0) = r1 + ptCenter(0)
    point1(1) = ptCenter(1)
    point1(2) = ptCenter(2)

    point2(0) = - r1 + ptCenter(0)
    point2(1) = ptCenter(1)
    point2(2) = ptCenter(2)

    Dim length1 As Double, length2 As Double
    length1 = Distance(point0, point1)
    length2 = Distance(point0, point2)

    point3(0) = r1 * (length2 - length1)/(length1 + length2) + ptCenter(0)
    point3(1) = ptCenter(1)
    point3(2) = ptCenter(2)

    Dim tmp1 As Double, ang1 As Double, a As Double, b As Double
    tmp1 = (Distance(point2, point0) ^ 2 + Distance(point3, point0) ^ 2 - _
            Distance(point3, point2) ^ 2)/(2 * Distance(point2, point0) * _
            Distance(point3, point0))
```

```
ang1 = Atn( - tmp1/Sqr( - tmp1 * tmp1 + 1)) + 2 * Atn(1)
b = Distance(point3, point0) * Tan(ang1)
a = Sqr(Distance(point3, point2) * Distance(point3, point1))

'画椭圆锥
Dim objSolid As Acad3DSolid
Set objSolid = AddEllipticalCone(point3, 2 * b, 2 * a, 2 * Distance(point3, point0))

'对椭圆锥进行三维旋转
Dim point4(2) As Double
point4(0) = point3(0)
point4(1) = point3(1) + 1
point4(2) = point3(2)

Dim tmp2 As Double, ang2 As Double
tmp2 = (Distance(point2, point3) ^ 2 + Distance(point3, point0) ^ 2 - _
    Distance(point0, point2) ^ 2)/(2 * Distance(point2, point3) * Distance(point3, point0))
ang2 = Atn( - tmp2/Sqr( - tmp2 * tmp2 + 1)) + 2 * Atn(1) - 0.5 * PI
objSolid.Rotate3D PtUcs2Wcs(point3), PtUcs2Wcs(point4), ang2

'剖切椭圆锥
Dim pt1(2) As Double, pt2(2) As Double, pt3(2) As Double
pt1(0) = ptCenter(0)
pt1(1) = ptCenter(1)
pt1(2) = ptCenter(2)

pt2(0) = ptCenter(0)
pt2(1) = ptCenter(1) + 1
pt2(2) = ptCenter(2)

pt3(0) = ptCenter(0) + 1
pt3(1) = ptCenter(1)
pt3(2) = ptCenter(2)
objSolid.SliceSolid PtUcs2Wcs(pt1), PtUcs2Wcs(pt2), PtUcs2Wcs(pt3), False

pt1(0) = ptCenter(0)
pt1(1) = ptCenter(1)
pt1(2) = ptCenter(2) + height

pt2(0) = ptCenter(0) + 1
pt2(1) = ptCenter(1)
pt2(2) = ptCenter(2) + height

pt3(0) = ptCenter(0)
```

```
    pt3(1) = ptCenter(1) + 1
    pt3(2) = ptCenter(2) + height
    objSolid.SliceSolid PtUcs2Wcs(pt1), PtUcs2Wcs(pt2), PtUcs2Wcs(pt3), False

    '对于上底半径大于下底半径的偏心圆台,还需要进行三维旋转和移动操作
    If radiusBottom < radiusTop Then
        pt1(0) = ptCenter(0)
        pt1(1) = ptCenter(1)
        pt1(2) = ptCenter(2) + 0.5 * height

        pt2(0) = ptCenter(0) + 1
        pt2(1) = ptCenter(1)
        pt2(2) = ptCenter(2) + 0.5 * height

        objSolid.Rotate3D PtUcs2Wcs(pt1), PtUcs2Wcs(pt2), PI

        pt1(0) = ptCenter(0) + offset
        pt1(1) = ptCenter(1)
        pt1(2) = 0

        pt2(0) = ptCenter(0)
        pt2(1) = ptCenter(1)
        pt2(2) = 0
        objSolid.Move PtUcs2Wcs(pt1), PtUcs2Wcs(pt2)

        pt1(0) = ptCenter(0)
        pt1(1) = ptCenter(1)
        pt1(2) = ptCenter(2)

        pt2(0) = ptCenter(0)
        pt2(1) = ptCenter(1)
        pt2(2) = ptCenter(2) + 1
        objSolid.Rotate3D PtUcs2Wcs(pt1), PtUcs2Wcs(pt2), PI
    End If
End Sub
```

其中用到了 4 个自定义函数 Distance、PtUcs2Wcs、GetActiveUCS 和 AddEllipticalCone。Distance 函数用于计算两个点之间的距离,其实现代码为:

```
Private Function Distance(pt1 As Variant, pt2 As Variant) As Double
    Distance = ((pt1(0) - pt2(0)) ^ 2 + (pt1(1) - pt2(1)) ^ 2 + (pt1(2) - pt2(2)) ^ 2) ^ 0.5
End Function
```

PtUcs2Wcs 函数用于将点的 UCS 坐标转化为 WCS 坐标,其实现代码为:

```
Private Function PtUcs2Wcs(ptUcs As Variant) As Variant
```

```
    PtUcs2Wcs = ThisDrawing.Utility.TranslateCoordinates(ptUcs, acUCS, acWorld, False)
End Function
```

GetActiveUCS 函数用于获得当前的 UCS,其实现代码为:

```
Public Function GetActiveUCS( ) As AcadUCS
    '如果 UCS 没有命名,需要创建这个 UCS
    If ThisDrawing.GetVariable("UCSNAME") = "" Then
        Dim ptOrigin(2) As Double          '要创建的 UCS 的原点
        Dim ptXAxis(2) As Double           'UCS 的 X 轴正半轴上一点
        Dim ptYAxis(2) As Double           'UCS 的 Y 轴正半轴上一点
        Dim xDir, yDir, org As Variant     '当前 UCS 的参数

        '获得当前 UCS 的参数
        xDir = ThisDrawing.GetVariable("UCSXDIR")
        yDir = ThisDrawing.GetVariable("UCSYDIR")
        org = ThisDrawing.GetVariable("UCSORG")

        'UCS 的原点
        ptOrigin(0) = org(0)
        ptOrigin(1) = org(1)
        ptOrigin(2) = org(2)

        '获得 UCS 的 X 轴正半轴上的一点
        ptXAxis(0) = org(0) + xDir(0)
        ptXAxis(1) = org(1) + xDir(1)
        ptXAxis(2) = org(2) + xDir(2)

        '获得 UCS 的 Y 轴正半轴上的一点
        ptYAxis(0) = org(0) + yDir(0)
        ptYAxis(1) = org(1) + yDir(1)
        ptYAxis(2) = org(2) + yDir(2)

        '创建和当前 UCS 重合的 UCS
        Set GetActiveUCS = ThisDrawing.UserCoordinateSystems.Add( _
            ptOrigin, ptXAxis, ptYAxis, "MyUCS")

        ThisDrawing.ActiveUCS = GetActiveUCS
    Else
        Set GetActiveUCS = ThisDrawing.ActiveUCS
    End If
End Function
```

AddEllipticalCone 函数用于创建椭圆锥体,其实现代码为:

```
Private Function AddEllipticalCone(ptCenter As Variant, xLength As Double,
```

```
    yLength As Double, height As Double) As Acad3DSolid
    Set AddEllipticalCone = ThisDrawing.ModelSpace.AddEllipticalCone( _
        ptCenter, xLength, yLength, height)

    '得到当前 UCS 的变换矩阵
    Dim transMatrix As Variant
    transMatrix = GetActiveUCS.GetUCSMatrix( )

    '对椭圆锥体进行变换操作
    AddEllipticalCone.TransformBy transMatrix
    AddEllipticalCone.Update
End Function
```

(6) 在“取消”按钮的单击事件中,结束程序的执行,其实现代码为:

```
Private Sub cmdCancel_Click( )
    End
End Sub
```

(7) 双击“工程资源管理器”窗口中的 ThisDrawing 图标,打开其代码窗口,在其中添加用于启动应用程序的宏:

```
Public Sub Main( )
    frmMain.Show
End Sub
```

9.5.4 实例效果

(1) 在 VBA 集成开发环境中,执行 Main 宏,系统会弹出如图 9.17 所示的对话框。

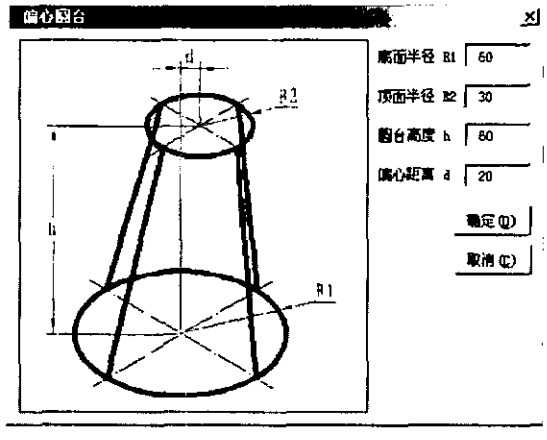


图 9.17 程序运行界面

(2) 在对话框中输入如图 9.17 中所示的数据,单击“确定”按钮,就能在图形窗口中得到一个偏心圆台,使用 AutoCAD 中的 3DORBIT 命令调整观察角度,能够得到如图 9.18 所示的结果。

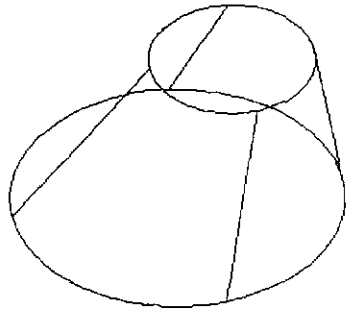


图 9.18　创建的偏心圆台

9.5.5　小结

这个程序对编程者的空间想象能力要求比较高，读者可以在此基础上完成创建其他类似实体的程序，例如创建“天圆地方”三维模型的程序。

学习本节内容之后，应该掌握下面的知识点：

- 更改视口的观察方向。
- SliceSolid 方法的使用。
- AddExtrudedSolidAlongPath 方法的使用。

第 10 章　布局和打印

布局和打印无论如何都应该作为一个大的主题来介绍，但是本章仅包含了一个实例，因为布局、视口的使用方法非常简单，并且无技巧可言，通过阅读帮助系统就能获得足够的资料，这里就不再浪费篇幅。

批量打印的程序是一个比较热门的话题，于是就介绍当前主流的几种方法，为用户创建批量打印的程序做好了全部的准备。

10.1　批 量 打 印

10.1.1　实例说明

本节的实例虽然没有提供一个完整的批量打印程序，但是所介绍的知识已经为创建完整的批量打印程序做好了所有的准备，只要在此基础上稍作修改，即可实现批量打印程序。

10.1.2　编程思路

在 ActiveX 模型中，Plot 对象的 PlotToDevice 方法使用指定的打印机打印指定的布局，PlotToFile 方法则使用虚拟打印机将指定的布局打印到文件中（例如生成 JPG、DWF 等文件）。

批量打印一般来说有三种方法：

- 打印图形中所有的布局。使用 Plot 对象的 SetLayoutsToPlot 方法可以设置所要打印的布局列表，将所有布局名称的列表存储在一个数组中，传递给 SetLayoutsToPlot 方法，再调用 PlotToDevice 方法即可打印图形中所有的布局。
- 打印图形模型空间中所有的打印区域（用某个图层上的矩形表示）。遍历模型空间所有的实体，如果该实体是轻量多段线，并且在指定的图层上，可使用 GetBoundingBox 方法获得其最小包围框的两个角点。使用 PlotType 属性设置模型空间布局的打印模式为窗口，使用 SetWindowToPlot 属性设置打印窗口的位置，然后使用 PlotToDevice 方法即可打印所有的打印区域。
- 打印图形模型空间中所有的打印区域（用图形中某个名称的块参照表示，一般是作为图框的块参照）。其原理与矩形表示的打印区域相同，不过是遍历模型空间时，获得打印区域的条件有所不同。

此外，还有可能需要获得 AutoCAD 中所有可用的打印机，可以使用 GetPlotDeviceNames 属性获得。

10.1.3　步骤

（1）在 AutoCAD 2005 中，打开 VBA 管理器，创建一个新图形，保存在适当的位置。进入 VBA 集成开发环境，双击“工程资源管理器”窗口中的 ThisDrawing 图标，打开其代码窗

口，在其中添加一个宏 PlotAllLayouts，用于打印当前图形中所有的布局：

```
Public Sub PlotAllLayouts( )
    Dim objPlot As AcadPlot     '打印对象
    Dim strLayouts( ) As String '当前图形中所有的布局名称

    Set objPlot = ThisDrawing.Plot

    Dim i As Integer
    For i = 0 To ThisDrawing.Layouts.Count - 1
        ReDim Preserve strLayouts(i)
        strLayouts(i) = ThisDrawing.Layouts.Item(i).Name
    Next

    objPlot.SetLayoutsToPlot strLayouts

    'objPlot.PlotToDevice strPlotConfig
    objPlot.PlotToFile "C:\Test\"
End Sub
```

将所有布局名称保存在一个数组 strLayouts 中，然后使用 SetLayoutsToPlot 方法设置所要打印的布局列表，最后使用 PlotToFile 或 PlotToDevice 方法打印图形。

(2) 添加一个过程 PlotAllPlotRgn，用于打印当前图形模型空间中所有的打印区域(用某个图层上的矩形表示)，参数 strLayerName 指定了打印矩形所在的图层，其实现代码为：

```
Public Sub PlotAllPlotRgn(ByVal strLayerName As String)
    '确保当前布局是模型空间
    ThisDrawing.ActiveLayout = ThisDrawing.Layouts.Item("Model")
    ThisDrawing.ActiveLayout.ConfigName = "DWF6 ePlot.pc3"

    '设置打印类型
    ThisDrawing.ActiveLayout.PlotType = acWindow

    '设置打印比例为"布满图纸"
    ThisDrawing.ActiveLayout.StandardScale = acScaleToFit

    '确保 AutoCAD 在前台进行打印，这样后一次打印会在前一次打印完成之后才开始'避免出现错误
    ThisDrawing.SetVariable "BACKGROUNDPLOT", 0

    Dim objPlot As AcadPlot
    Set objPlot = ThisDrawing.Plot

    '对所有的打印区域进行窗口打印
    Dim ptMin As Variant, ptMax As Variant
    Dim ent As AcadEntity
    Dim i As Integer
    For Each ent In ThisDrawing.ModelSpace
        If StrComp(ent.Layer, strLayerName, vbTextCompare) = 0 Then
```

```
            If TypeOf ent Is AcadLWPolyline Then
                ent.GetBoundingBox ptMin, ptMax

                '将三维点转化为二维点坐标
                ReDim Preserve ptMin(0 To 1)
                ReDim Preserve ptMax(0 To 1)

                '设置打印窗口
                ThisDrawing.ActiveLayout.SetWindowToPlot ptMin, ptMax

                '打印当前的区域
                'objPlot.PlotToDevice strPlotConfig
                objPlot.PlotToFile "C:\Test\" & ThisDrawing.Name & " - " & i & ".dwf"
                i = i + 1
            End If
        End If
    Next ent

    '恢复系统变量的值
    ThisDrawing.SetVariable "BACKGROUNDPLOT", 2
End Sub
```

上面的代码中，有两点需要注意：

- SetWindowToPlot 方法所接受的两个角点是二维数组，因此在使用 GetBoundingBox 方法获得矩形的最小包围框交点之后，必须使用 ReDim 语句舍弃它的第三维坐标，同时要使用 Preserve 关键字保证前两个元素的值不变。
- BACKGROUNDPLOT 系统变量用于控制 AutoCAD 是否在后台进行打印，默认值为 2，表示在后台打印。但是如果 AutoCAD 在后台打印，VBA 程序不等一次打印进行完毕就会进入下一次打印，因为打印机正忙着处理其他的打印任务，就会导致一个错误。为避免这种错误，在使用 PlotToFile 或 PlotToDevice 方法打印之前，将 BACKGROUNDPLOT 系统变量的值设置为 0，要求 AutoCAD 在前台进行打印。

(3) 添加一个宏 PlotAllPlotBlk，用于打印当前图形模型空间中所有的打印区域(用图形中的图框块来表示)，参数 strBlkName 指定了代表打印区域的图框块的名称，其实现代码为：

```
Public Sub PlotAllPlotBlk(ByVal strBlkName As String)
    '确保当前布局是模型空间
    ThisDrawing.ActiveLayout = ThisDrawing.Layouts.Item("Model")
    ThisDrawing.ActiveLayout.ConfigName = "DWF6 ePlot.pc3"

    '设置打印类型
    ThisDrawing.ActiveLayout.PlotType = acWindow

    '设置打印比例为"布满图纸"
    ThisDrawing.ActiveLayout.StandardScale = acScaleToFit

    '确保 AutoCAD 在前台进行打印，这样后一次打印会在前一次打印完成之后才开始'避免出现错误
```

```
    ThisDrawing.SetVariable "BACKGROUNDPLOT", 0

    Dim objPlot As AcadPlot
    Set objPlot = ThisDrawing.Plot

    '对所有的打印区域进行窗口打印
    Dim ptMin As Variant, ptMax As Variant
    Dim ent As AcadEntity
    Dim i As Integer
    For Each ent In ThisDrawing.ModelSpace
        If TypeOf ent Is AcadBlockReference Then
            If StrComp(ent.Name, strBlkName, vbTextCompare) = 0 Then
                ent.GetBoundingBox ptMin, ptMax

                '将三维点转化为二维点坐标
                ReDim Preserve ptMin(0 To 1)
                ReDim Preserve ptMax(0 To 1)

                '设置打印窗口
                ThisDrawing.ActiveLayout.SetWindowToPlot ptMin, ptMax

                '打印当前的区域
                'objPlot.PlotToDevice strPlotConfig
                objPlot.PlotToFile "C:\Test\" & ThisDrawing.Name & " - " & i & ".dwf"
                i = i + 1
            End If
        End If
    Next ent

    '恢复系统变量的值
    ThisDrawing.SetVariable "BACKGROUNDPLOT", 2
End Sub
```

(4) 添加一个宏 ListPrinters,用于获得 AutoCAD 中所有可用的打印机,并将这些打印机的名称显示在立即窗口中,其实现代码为:

```
Public Sub ListPrinters( )
    Dim objLayout As AcadLayout
    Set objLayout = ThisDrawing.ActiveLayout
    objLayout.RefreshPlotDeviceInfo

    '获得所有的可用打印机
    Dim plotDevices As Variant
    plotDevices = objLayout.GetPlotDeviceNames

    '显示打印机列表
    Dim i As Integer
    For i = 0 To UBound(plotDevices)
```

```
        Debug.Print plotDevices(i)
    Next i
End Sub
```

10.1.4 实例效果

(1) 在 AutoCAD 2005 中打开一个图形，在 VBA 集成开发环境中，执行 PlotAllLayouts 宏，AutoCAD 会分别将该图形所有的布局都打印到 DWF 文件中。

(2) 在图形中创建一个名为“打印区域”的图层，创建若干个矩形，并将其放置在“打印区域”图层上，如图 10.1 所示。

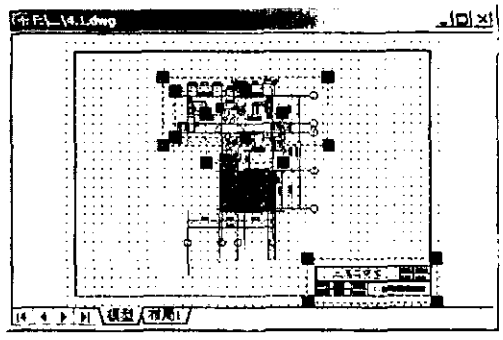


图 10.1 创建打印矩形

(3) 在 ThisDrawing 模块中添加一个宏 TestPlot，用于测试打印所有打印区域的效果，其实现代码为：

```
Public Sub TestPlot( )
    Call PlotAllPlotRgn("打印区域")
End Sub
```

(4) 执行 TestPlot 宏，能在 C:\Test 文件夹中得到 4 个 DWF 文件，如图 10.2 所示。

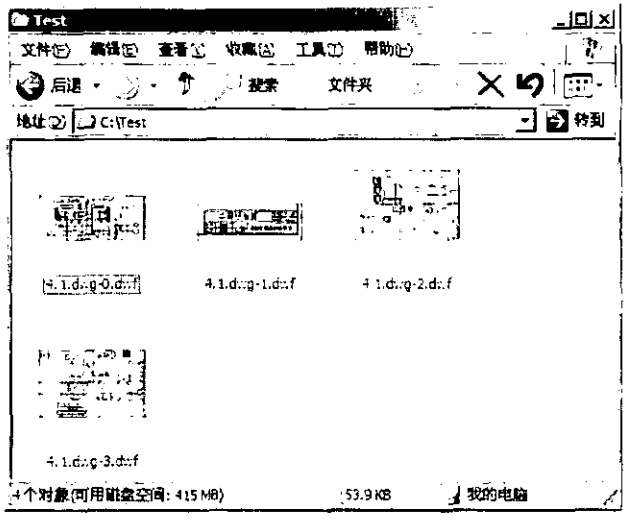


图 10.2 批量打印的结果

(5) 执行 ListPrinters 宏，然后打开立即窗口，能够得到如图 10.3 所示的结果。

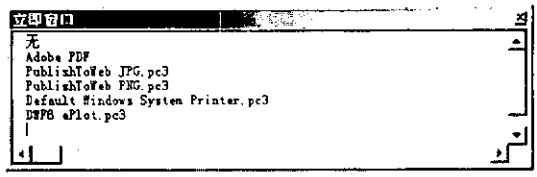


图 10.3　查看 AutoCAD 中所有可用的打印机

10.1.5　小结

能够在一个图形内进行批量打印，那么对多个文件进行批量打印也非难事，下面来分析一下还有什么问题：

- 使用“打开文件”对话框添加图形文件，这在 5.4 节(使用 ObjectDBX 实现批量文字替换)已经介绍过。
- 使用“浏览”对话框获得一个路径，在 11.3 节(使用 SummaryInfo 对象保存图形信息)中将会介绍。
- 获得一个文件夹中所有的 DWG 文件，在 16.2 节(搜索某个文件夹中所有的文件)将会介绍。

然后，就是依次打开每一个图形文件，进行批量打印的操作。

10.2　布　　局

AutoCAD 用户界面中的布局与 ActiveX 模型中布局的表现形式不太一样，在 ActiveX 模型中，AutoCAD 布局功能的实现由两个部分来完成：布局(Layout 对象)和块(Block)。其中，布局对象包含了打印设置和 AutoCAD 界面中包含的其他可见的特性，块对象则包含了布局的几何对象。

要创建新布局，可以使用下面的语句：

```
ThisDrawing.Layouts.Add"NewLayout"
```

设置当前布局，可以使用下面的语句：

```
ThisDrawing.Activelayout = ThisDrawing.Layouts.Item("Model")
```

每个布局都和惟一的一个块关联，可以使用 Block 属性获得布局所关联的块。可以使用下面的 GetLayoutOfEnt 宏来判断选择的实体位于哪个布局：

```
Public Sub GetLayoutOfEnt( )
    Dim objSelect As Object
    Dim ptPick As Variant
    ThisDrawing.Utility.GetEntity objSelect.ptPick

    Dim objEnt As AcadEntity
```

```
    Set objEnt = objSelect

    MsgBox"实体所在布局："& Id2Layout(objEnt.OwnerID)
End Sub
```

其中，Id2Layout 是一个自定义函数，用于根据布局关联的块的 ObjectId 获得对应的布局名称，其实现代码为：

```
Public Function Id2Layout(Byvalid As Long)As String
    Dim objLayout As AcadLayout
    For Each objlayouth In ThisDrawing.Layouts
        If objLayout.Block.objectId = id Then
            Id2Layout = objLayout.Name
            Exit Function
        End If
    Next objlayout
End Function
```

第 11 章　高级绘图与组织技术

扩展数据作为二次开发应用程序中标识实体的一种手段，已经有着相当长的历史，本节就从一个典型的实例来剖析扩展数据使用。扩展字典是 Autodesk 用来取代扩展数据的一种技术，在功能上比扩展数据更为强大，本章的实例从两者的区别出发来介绍扩展字典和命名对象字典的使用。

SummaryInfo 对象提供了访问图形特性的接口，在关闭图形的时候向图形写入信息无疑是比较理想的手段，此外，根据图形特性来查找图形是一个不可缺少的程序，既是一种工具，也是一个好的教学实例。

绘图顺序用于调整图形对象的显示层次，在创建图形时可能会需要在程序中完成绘图顺序的调整，“绘图顺序”一节就用较少的篇幅介绍了这个不算复杂的应用。

11.1　扩 展 数 据

11.1.1　实例说明

在 AutoCAD 中，用户无法向某个图形对象追加信息。在 VBA 中，可以将扩展数据用于连接信息与图形中对象的方式。

扩展数据在编程过程中有很重要的作用。

例如，在一幅平面规划的图形中，有大量多段线实体，一些用来代表道路中心线，还有一些代表地下管线，也有的代表输电线。这些多段线从外观上来看没有什么区别，可能用户将代表不同对象的多段线分别放在不同的图层中，但是这种方式在编程中并不总是那么可靠。

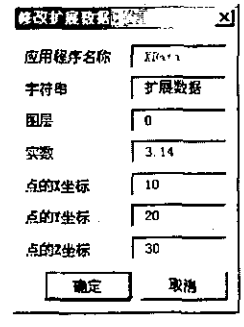


图 11.1　程序运行界面

这时候，就可以为不同类型的多段线添加代表其类型的扩展数据，代表道路中心线的多段线包括扩展数据“道路”，代表地下管线的多段线包含扩展数据“地下管线”等。此时，要获得图形中所有的道路中心线，可以在遍历图形对象时获得其扩展数据，发现包括“道路”信息的多段线，就可以判断出这条多段线代表的是道路中心线，将其添加到选择集中。

本节的实例以一个形如属性窗口的对话框来为实体添加扩展数据，或者显示实体中包含的扩展数据，如图 11.1 所示。

当然，AutoCAD 中许多非实体的对象（例如图层、线型、块和字典等对象）也可能有扩展数据，这在本节的实例中没有讨论。

11.1.2 编程思路

1. 添加和获得对象扩展数据

向对象添加或修改扩展数据可以使用 SetXdata 方法，其定义为：

```
object.SetXData XDataType, XData
```

其中，XDataType 是一个 short 类型变量数组，数组中的每个元素用于说明扩展数据的类型；XData 是一个 Variant 类型的数组，数组中的每个元素包含了扩展数据的内容。XDataType 和 XData 的维数必须一致，XDataType 中的元素一一对应地说明 XData 中元素的类型，其具体含义可以参考下面的代码：

```
Dim DataType(0 To 7) As Integer
Dim Data(0 To 7) As Variant

DataType(0) = 1001: Data(0) = "Test_Application"          '应用程序名称
DataType(1) = 1000: Data(1) = "This is a test for xdata"  '字符串

DataType(2) = 1003: Data(2) = "0"                         '图层
DataType(3) = 1040: Data(3) = 1.23479137438413E+40        '实数
DataType(4) = 1041: Data(4) = 1237324938                  '距离
DataType(5) = 1070: Data(5) = 32767                       '16 位整数
DataType(6) = 1071: Data(6) = 32767                       '32 位整数
DataType(7) = 1042: Data(7) = 10                          '比例因子

'向实体中添加扩展数据
lineObj.SetXData DataType, Data
```

获得对象的扩展数据可以使用 GetXData 方法，其定义为：

```
object.GetXData AppName, XDataType, XDataValue
```

其中，AppName 指定添加扩展数据的应用程序名称(多个应用程序可以向同一个对象添加扩展数据)，如果输入一个空字符串表示要获得所有应用程序添加的扩展数据；XDataType 返回一个 short 类型的数组，包含对象扩展数据的类型信息；XdataValue 返回一个 Variant 类型的数组，包含了对象的扩展数据。获得对象扩展数据的基本方法为：

```
Dim xdataOut As Variant
Dim xtypeOut As Variant
lineObj.GetXData "", xtypeOut, xdataOut
```

2. 扩展数据类型的含义

从前面添加扩展数据的示例代码中可以看出，对象扩展数据的类型用一个整数来表示，例如 1001 代表应用程序名称，1000 代表字符串，1003 代表图层等。这些代表扩展数据类型的整数实际上是 DXF 组码，可以从 AutoCAD 的帮助文档中查阅，如图 11.2 所示。

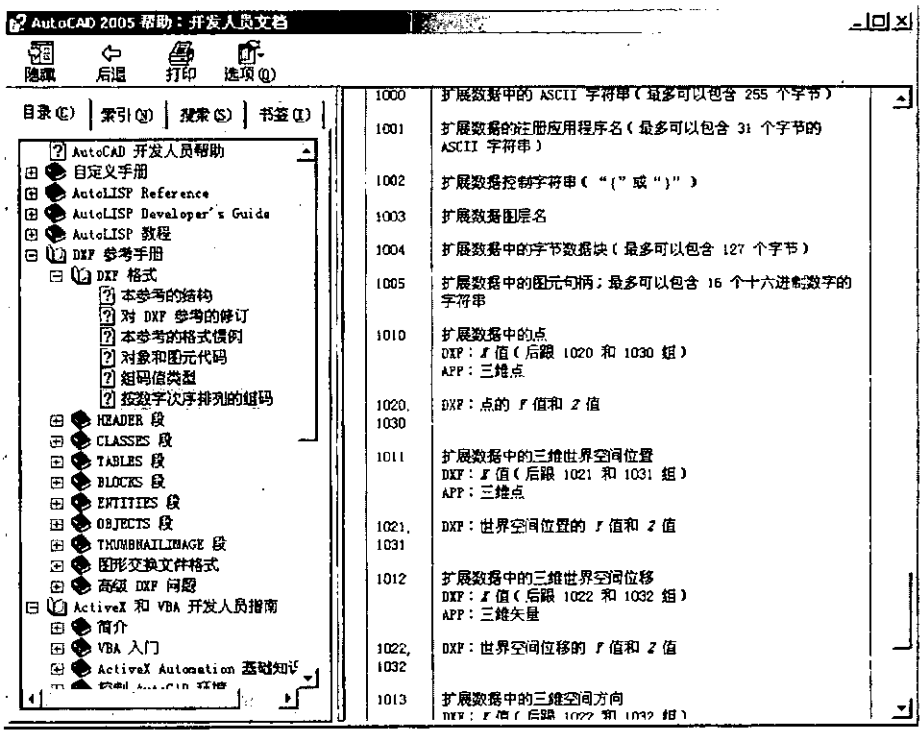


图 11.2　查阅 DXF 组码

11.1.3　步骤

(1) 在 VBA 集成开发环境中，创建一个新工程，保存在适当的位置。选择“插入”|“模块”菜单项，向工程中添加一个新的标准模块，将其命名为 mdlMain。在 mdlMain 模块中添加一个函数 CheckKey，用于检测用户是否按下了指定的键。为了实现这个函数，还要声明一个 API 函数 GetAsyncKeyState 和一个常量 VK_ESCAPE(可以通过 Visual Studio 6.0 提供的一个工具 API Text Viewer 来添加)。其相关代码为：

```
Private Const VK_ESCAPE = &H1B '代表 Esc 键
Private Declare Function GetAsyncKeyState Lib "user32" (ByVal vKey As Long) As Integer

'功能：判断用户是否按下某一个键
'输入：代表键的常量(从 API Viewer 中获得)
'调用：API 函数 GetAsyncKeyState
'返回：如果用户按下了指定的键，返回 True；否则返回 False
'示例：If CheckKey(&H1B) = True Then do sth
Private Function CheckKey(lngKey As Long) As Boolean
    If GetAsyncKeyState(lngKey) Then
        CheckKey = True
    Else
        CheckKey = False
    End If
```

```
End Function
```

(2) 添加两个全局变量 addMode 和 objCurrent,其相关代码为:

```
Public addMode As Boolean                    '选择的实体是否包含扩展数据
Public objCurrent As AcadEntity              '当前被选择的实体
```

(3) 创建一个宏 OperateXData,用于向实体添加扩展数据,或者显示、修改实体的扩展数据,其实现代码为:

```
Public Sub OperateXData( )
    Dim objSelect As AcadEntity
    Din basePnt As Variant
    '选择对象并判断类型
    On Error Resume Next

Retry:
    ThisDrawing.Utility.GetEntity objSelect, basePnt

    '处理按下 Esc 键的错误
    If objSelect Is Nothing Then
        If CheckKey(VK_ESCAPE) = True Then
            Exit Sub
        Else
            GoTo Retry
        End If
    End If

    '处理未选择到实体的错误
      If Err <> 0 Then
        Err.Clear
        GoTo Retry
    End If

    '判断实体是否包含扩展数据
    If HasXData(objSelect, "XData") Then
        addMode = False
    Else
        addMode = True
    End If

    Set objCurrent = objSelect
    frmMain.Show
End Sub
```

上面的代码用 GetEntity 方法提示用户选择一个实体,然后判断该实体是否已经包含扩展数据,根据不同的结果设置 addMode 变量的值。

其中,HasXData 是一个自定义函数,用于判断选择的实体是否包含扩展数据,其实现

代码为：

```
Private Function HasXData(ByVal ent As AcadEntity, ByVal strAppName As String) As Boolean
    Dim dataType As Variant
    Dim data As Variant
    ent.GetXData strAppName, dataType, data

    HasXData = True
    If IsEmpty(dataType) Then
        HasXData = False
    End If
End Function
```

判断实体是否包含扩展数据的依据是代表实体扩展数据类型的变量 dataType 是否已经被初始化，如果被初始化就表示实体包含了扩展数据，反之则不包含扩展数据。

(4) 选择“插入”|“用户窗体”菜单项，向工程中添加一个对话框，并在对话框上放置如图 11.3 所示的控件。文本框的名称分别修改为 txtAppName、txtString、txtLayer、txtReal、txtPtX、txtPtY 和 txtPtZ，两个命令按钮的名称修改为 cmdOk 和 cmdCancel。

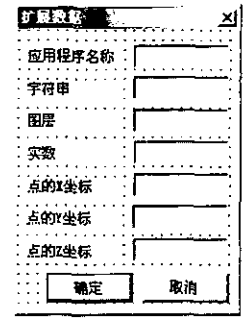


图 11.3　设计对话框

(5) 在窗体的初始化事件中，根据所选择的对象是否包含扩展数据，修改窗体的标题，并且根据需要决定是否显示对象的扩展数据，其相关代码为：

```
Private Sub UserForm_Initialize( )
    If addMode Then
        Me.Caption = "添加扩展数据"
        txtAppName.Text = "XData"
    Else
        Me.Caption = "修改扩展数据"

        '显示扩展数据
        Call ShowXData
    End If
End Sub
```

其中，ShowXData 函数的定义为：

```
Private Sub ShowXData( )
    Dim dataType As Variant
    Dim data As Variant

    objCurrent.GetXData "XData", dataType, data

    Dim i As Integer
    Dim pt As Variant     '存储点的坐标
    For i = LBound(dataType) To UBound(dataType)
        Select Case dataType(i)
```

```
            Case 1001
                txtAppName.Text = data(i)
            Case 1000
                txtString.Text = data(i)
            Case 1003
                txtLayer.Text = data(i)
            Case 1040
                txtReal.Text = data(i)
            Case 1011
                pt = data(i)
                txtPtX.Text = pt(0)
                txtPtY.Text = pt(1)
                txtPtZ.Text = pt(2)
        End Select
    Next i
End Sub
```

LBound 函数返回一个长整型的变量，其值为指定数组的最小下标；UBound 函数返回一个长整型的变量，其值为指定数组的最大下标。

(6) 在“确定”按钮的单击事件中，添加或修改实体的扩展数据，其实现代码为：

```
Private Sub cmdOk_Click( )
    '确保所有的文本框不为空
    Dim txt As Control
    For Each txt In frmMain.Controls
        If TypeOf txt Is TextBox Then
            If txt.Text = "" Then
                MsgBox "文本框内容不能为空!", vbCritical
                Exit Sub
            End If
        End If
    Next txt

    '添加或修改实体的扩展数据
    Call SetEntXData
    End
End Sub
```

其中，SetEntXData 是一个自定义函数，其实现代码为：

```
Private Sub SetEntXData( )
    Dim dataType(0 To 4) As Integer
    Dim data(0 To 4) As Variant
    Dim pt(0 To 2) As Double

    dataType(0) = 1001: data(0) = "XData"
    dataType(1) = 1000: data(1) = txtString.Text
```

```
    dataType(2) = 1003: data(2) = txtLayer.Text
    dataType(3) = 1040: data(3) = txtReal.Text
    pt(0) = txtPtX.Text: pt(1) = txtPtY.Text: pt(2) = txtPtZ.Text
    dataType(4) = 1011: data(4) = pt

    objCurrent.SetXdata dataType, data
End Sub
```

(7) 在“取消”按钮中，添加退出程序的代码：

```
Private Sub cmdCancel_Click( )
    End
End Sub
```

至此为止，编码工作全部完成。

11.1.4　实例效果

(1) 在 AutoCAD 2005 中，创建若干个图形对象。执行 OperateXData 宏，在图形窗口中选择一个实体，系统会弹出如图 11.4 所示的“添加扩展数据”对话框。

(2) 在“添加扩展数据”对话框中，将要添加的扩展数据输入到对应的文本框中，单击“确定”按钮，完成扩展数据的添加。

(3) 再次执行 OperateXData 宏，在图形窗口中选择一个实体，如果选择的是没有添加扩展数据的实体，系统仍会弹出“添加扩展数据”对话框。如果选择的是添加了扩展数据的实体，则会弹出“修改扩展数据”对话框，在其中修改一些数据，单击“确定”按钮，完成扩展数据的修改。

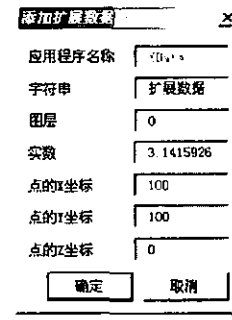


图 11.4　添加扩展数据的界面

11.1.5　小结

学习本节的内容之后，需要掌握下面几个知识点：

- GetEntity 方法的错误处理。
- 判断用户是否按下 Esc 键的方法。
- 添加和获得对象扩展数据。
- DXF 组码的查询。

11.2　扩展字典和命名对象字典

11.2.1　实例说明

与扩展数据的作用类似，但是字典提供了一种更好的保存外部数据的机制，与扩展数据相比，字典有两个优势：

- 每个对象的扩展数据不能超过 16KB，字典没有这种限制。

• 扩展数据中仅能保存有限种类的数据，字典没有这种限制。

本节的实例演示 AutoCAD 中两种字典(扩展字典和对象命名字典)的使用，包含了存储和读出保存在两种字典中信息的方法。

11.2.2 编程思路

1. 字典概述

字典提供一种机制，使用关联的关键字符串来保存和检索对象。字典中的对象可以使用它们的关键词来引用。字典能包含任何类型的对象，包括其他字典。字典不会执行项目的类型检查。

通过 VBA，用户可以创建新的字典，向已存在的字典添加条目，获得给定对象的关键词或者给定关键词的对象。用户可以修改与给定关键词相关联的对象，或者更改给定对象的关键词。

2. 扩展字典和命名对象字典

AutoCAD 中提供了两种字典，分别是扩展字典和命名对象字典(Named Object Dictionary)。

扩展字典是一种与具体的对象关联的字典，每个对象仅能拥有一个扩展字典，其中可以保存与该对象有关的信息。

命名对象字典则用来保存与实体无关的数据，AutoCAD 本身就使用对象命名字典来保存一些信息，例如组字典(保存编组的信息)、多线样式字典等。

此外还有一个名词必须在这里介绍，那就是扩展记录(AcadXRecord)。前面已经介绍过，字典能够保存多种类型的数据，其中大部分的数据都是保存在扩展记录中。一个扩展记录可以作为字典中的一个检索对象，可以通过其对应的关键字来访问。

3. 使用扩展字典

在扩展字典中保存信息的基本步骤如下：

(1) 使用 GetExtensionDictionary 方法创建一个对象的扩展字典。

(2) 使用 AddXRecord 方法向扩展字典添加一条扩展记录。

(3) 使用 SetXRecordData 方法将数据保存在扩展记录中。

读取扩展字典中数据的基本步骤如上：

(1) 使用 GetExtensionDictionary 方法获得一个对象的扩展字典。

(2) 使用 GetObject 方法获得指定的扩展记录。

(3) 使用 GetXRecordData 方法读取保存在扩展记录中的数据。

4. 使用命名对象字典

在命名对象字典中保存信息的基本步骤如下：

(1) 使用 Dictionaries 集合的 Add 方法添加一个命名对象字典。

(2) 使用 AddXRecord 方法向字典添加一条扩展记录。

(3) 使用 SetXRecordData 方法将数据保存在扩展记录中。

读取命名对象字典中的数据的基本步骤如下：

(1) 使用集合对象的基本操作，获得 Dictionaries 集合中指定的字典。

(2) 使用 GetObject 方法获得指定的扩展记录。

(3) 使用 GetXRecordData 方法读取保存在扩展记录中的数据。

11.2.3 步骤

(1) 在 AutoCAD 2005 中，创建一个新工程，保存在适当的位置。进入 VBA 集成开发环境，在“工程资源管理器”窗口中，双击 ThisDrawing 图标，打开其代码窗口，在其中输入下面的代码：

```
Private Const VK_ESCAPE = &H1B '代表 Esc 键
Private Declare Function GetAsyncKeyState Lib "user32" (ByVal vKey As Long) As Integer

'判断用户是否按下某一个键
Private Function CheckKey(lngKey As Long) As Boolean
        If GetAsyncKeyState(lngKey) Then
            CheckKey = True
        Else
            CheckKey = False
    End If
End Function

'向实体添加扩展字典和扩展记录
Public Sub AddOrPrintXRecord( )
    Dim objSelect As AcadEntity
    Dim ptPick As Variant
    '选择对象并判断类型
    On Error Resume Next

Retry:
    ThisDrawing.Utility.GetEntity objSelect, ptPick

    '处理按下 Esc 键的错误
    If objSelect Is Nothing Then
        If CheckKey(VK_ESCAPE) = True Then
            Exit Sub
        Else
            GoTo Retry
        End If
    End If

    '处理未选择到实体的错误
    If Err <> 0 Then
        Err.Clear
        GoTo Retry
    End If
```

```
    '保存或者显示扩展记录中存储的数据
    '等待插入具体的实现代码
End Sub
```

上面的代码是对 GetEntity 方法进行错误处理的基本框架。

(2) 在 AddOrPrintXRecord 宏的待插入实现代码部分，添加下面的代码，实现向对象扩展字典中添加数据和读取扩展字典中的数据：

```
Dim objDict As AcadDictionary
Dim objXRecord As AcadXRecord

'判断对象是否已经拥有扩展字典
If objSelect.HasExtensionDictionary Then
    '获得扩展字典
    Set objDict = objSelect.GetExtensionDictionary
    Set objXRecord = objDict.GetObject("Demo")

    '获得保存数据的内容
    Dim XRecordDataTypeOut As Variant
    Dim XRecordDataOut As Variant
    objXRecord.GetXRecordData XRecordDataTypeOut, XRecordDataOut

    '显示扩展记录中保存的数据
    Dim i As Integer
    For i = 0 To 3
        Select Case XRecordDataTypeOut(i)
            Case 1
                Debug.Print ("字符串：" & XRecordDataOut(i))
            Case 10
                Debug.Print ("点：" & "(" & Round(XRecordDataOut(i)(0), 4) & "," & _
                             Round(XRecordDataOut(i)(1), 4) & "," & _
                             Round(XRecordDataOut(i)(2), 4) & ")")
            Case 40
                Debug.Print ("实数：" & XRecordDataOut(i))
            Case 90
                Debug.Print ("整数：" & XRecordDataOut(i))
        End Select
    Next i
Else
    '创建扩展字典
    Set objDict = objSelect.GetExtensionDictionary
    Set objXRecord = objDict.AddXRecord("Demo")

    '指定保存数据的内容
    0Dim XRecordDataType(0 To 3) As Integer
```

```
    Dim XRecordData(0 To 3) As Variant
    XRecordDataType(0) = 90: XRecordData(0) = 6                    '整数
    XRecordDataType(1) = 40: XRecordData(1) = 3.1415926            '实数
    XRecordDataType(2) = 1: XRecordData(2) = "存储了扩展数据"'字符串
    XRecordDataType(3) = 10: XRecordData(3) = ptPick               '点

'在扩展记录中保存数据
    objXRecord.SetXRecordData XRecordDataType, XRecordData
End If
```

在扩展记录中存储和读取数据的方法，都与扩展数据类似，但是扩展记录保存数据时使用的组码是 AutoCAD 的标准组码，这与扩展数据不同。要查询 AutoCAD 的标准组码，可以打开 AutoCAD 的开发人员文档(默认情况下是 AutoCAD 安装路径中的 Help\acad_dev.chm 文件)，查询 DXF 参考手册中的相关部分，如图 11.5 所示。

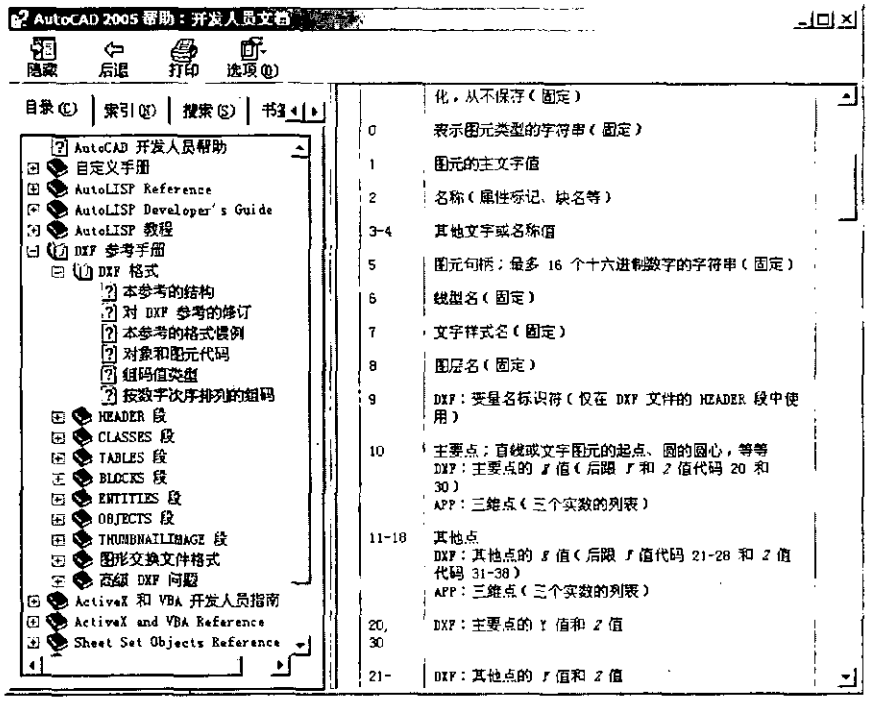


图 11.5　查询标准的 DXF 组码

需要注意的是，创建和获得对象扩展字典都使用了 GetExtensionDictionary 方法，该方法在执行时会自动判断对象是否包含扩展字典，如果已经包含，直接返回其扩展字典，否则创建与该对象关联的扩展字典，并且返回新建的字典。

(3) 创建一个新的宏 NameObjectDict，用于向命名对象字典中存储数据，并在存储之后将其读取出来，其实现代码为：

```
'   Public Sub NameObjectDict( )
    On Error Resume Next

    '判断是否已经存在"MyDict"字典
```

```
    Dim objDict As AcadDictionary
    Dim objXRecord As AcadXRecord
    Set objDict = ThisDrawing.Dictionaries("MyDict")

    '如果发生错误,表示指定的字典不存在
    If Err Then
        Err.Clear
        '添加字典
        Set objDict = ThisDrawing.Dictionaries.Add("MyDict")
        Set objXRecord = objDict.AddXRecord("图形用途")

        '指定保存数据的内容
        Dim XRecordDataType(0) As Integer
        Dim XRecordData(0) As Variant
        XRecordDataType(0) = 1: XRecordData(0) = "该图形是所有图形的基础"

        objXRecord.SetXRecordData XRecordDataType, XRecordData
    Else
        Set objXRecord = objDict.GetObject("图形用途")

        '获得保存数据的内容
        Dim XRecordDataTypeOut As Variant
        Dim XRecordDataOut As Variant
        objXRecord.GetXRecordData XRecordDataTypeOut, XRecordDataOut

        MsgBox "图形用途:" & XRecordDataOut(0)
    End If
End Sub
```

上面的代码中,值得注意的是判断命名对象字典是否存在的方法,其核心语句是:

```
Set objDict = ThisDrawing.Dictionaries("MyDict")

'如果发生错误,表示指定的字典不存在,否则表示存在
If Err Then
'如果字典不存在,做如下的处理
Else
'如果字典存在,做如下的处理
End If
```

这种错误处理的方法适用于判断其他集合对象中是否包含指定的元素。

11.2.4 实例效果

(1) 在 VBA 集成开发环境中,执行 AddOrPrintXRecord 宏,AutoCAD 会在命令窗口提示"选择对象:",在图形窗口中任意拾取一个图形对象。

(2) 确保 VBA 开发环境中的立即窗口打开,如果没有打开,可以选择"视图"|"立即窗口"菜单项将其打开。再次执行 AddOrPrintXRecord 宏,并在 AutoCAD 图形窗口中选择步骤(1)选择的实体,能够在立即窗口得到如图 11.6 所示的结果。

(3) 连续两次执行 NameObjectDict 宏，第一次执行之后没有任何提示，第二次执行之后系统会弹出如图 11.7 所示的对话框。

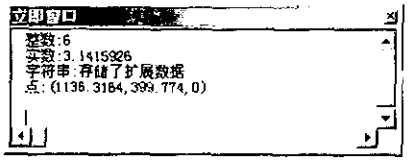


图 11.6　显示扩展字典中保存的数据

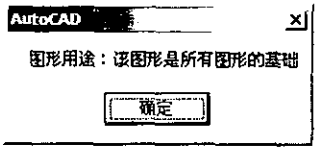


图 11.7　显示对象命名字典中保存的数据

11.2.5　小结

学习本节内容之后，需要掌握下面的几个知识点：

- 使用扩展字典来保存对象的附加数据。
- 使用对象命名字典来保存与对象无关联的数据。

扩展记录是 AutoCAD 设计用来代替扩展数据的，因此在向对象附加数据的时候，应优先考虑使用扩展记录。

11.3　使用 SummaryInfo 对象保存图形信息

11.3.1　实例说明

虽然自 AutoCAD 2000 版本已经引入了图形属性的概念，但在对象模型中并没有把它当成一个对象，而是保存在 Xrecord 扩展记录的 DWGPROPS 名称下。

自从 AutoCAD 2004 版本开始，对象模型单独把它做为一个独立的对象以方便编程。

SummryInfo 中的内容可以通过在 Windows 资源管理器中查看图形的属性时查到，所以把图纸中一些关键的内容写入到 SummryInfo 有助于图形的查询。

本节的实例包括两个部分，第一部分是在保存图形时向图形属性添加一些信息，第二部分是根据图形特性查找图形。

11.3.2　编程思路

将每一特殊的图形保存其图形信息到 SummryInfo 对象中，最好的方法就是在保存文件时触发事件来自动保存这些信息。

什么是图形中有用的信息？像机械行业中，每张图纸都需要有图框，图框的大小，也有零件或产品的名称，图号、材料等信息，所以对这些信息进行采集并保存到 SummryInfo 对象中是最好不过的。

可以通过 ThisDrawing. SummaryInfo 访问当前图形的 SummaryInfo 对象，通过其 Title、Subject 等属性即可向图形属性中添加一些信息。

11.3.3　步骤

(1) 在 AutoCAD 2005 中，打开 VBA 管理器，创建一个新工程，保存在适当的位置。进

入 VBA 集成开发环境，双击“工程资源管理器”窗口中的 ThisDrawing 图标，打开其代码窗口，在左上角的列表中选择 AcadDocument 对象，然后在右上角的列表中选择 BeginCommand 事件，在其中向图形的 SummaryInfo 对象添加信息，其相关代码为：

```
Private Sub AcadDocument_BeginCommand(ByVal CommandName As String)
    If StrComp(CommandName, "qsave", vbTextCompare) = 0 _
        Or StrComp(CommandName, "saveas", vbTextCompare) = 0 Then
        Dim objSummaryInfo As AcadSummaryInfo
        Set objSummaryInfo = ThisDrawing.SummaryInfo

        '添加一些预置类型的信息
        objSummaryInfo.Title = "示例图形"
        objSummaryInfo.Subject = "教学"
        objSummaryInfo.Author = "明经通道"
        objSummaryInfo.Keywords = "明经通道 CAD mjtd VBA"
        objSummaryInfo.Comments = "明经通道是一家国内著名的 CAD 站点，提供了大量免费的应用
            和开发资料。"
        objSummaryInfo.LastSavedBy = "zfbj"

        '添加自定义的信息
        objSummaryInfo.AddCustomInfo "图纸尺寸", "A3"
        objSummaryInfo.AddCustomInfo "图形比例", "1：100"
    End If
End Sub
```

(2) 选择“插入”|“用户窗体”菜单项，向工程中添加一个对话框，将其名称修改为 frmMain，并在其中添加如图 11.8 所示的控件。

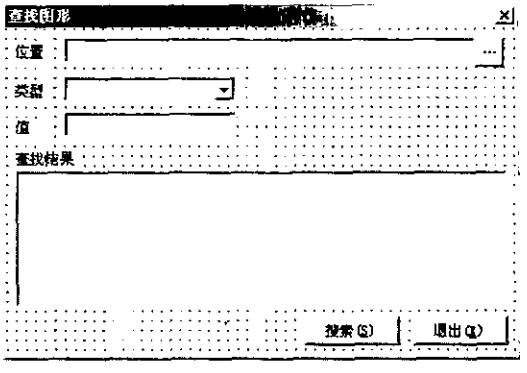


图 11.8 设计窗体

其中，两个文本框的名称为 txtPath 和 txtValue，“类型”组合框的名称为 cboType，“查找结果”列表框的名称为 lstFiles，三个命令按钮的名称为 cmdBrowser、cmdSearch 和 cmdExit。

(3) 在窗体代码窗口的通用区，添加常量和 API 函数的声明：

```
Private Type BrowseInfo
```

```
    hOwner As Long
    pidlRoot As Long
    pszDisplayName As String
    lpszTitle As String
    ulFlags As Long
    lpfn As Long
    lParam As Long
    iImage As Long
End Type
Private Const MAX_PATH = 260

'API 函数的声明
Private Declare Function SHBrowseForFolder Lib "shell32.dll" _
    Alias "SHBrowseForFolderA" (lpBrowseInfo As BrowseInfo) As Long
Private Declare Function FindWindow Lib "user32" Alias "FindWindowA" (ByVal lpClassName As
    String, _
    ByVal lpWindowName As String) As Long
Private Declare Function SHGetPathFromIDList Lib "shell32.dll" Alias "SHGetPathFromIDListA"
    (ByVal _
    pidl As Long, ByVal pszPath As String) As Long
```

(4) 在窗体初始化的事件中，向“类型”组合框中添加几个预置属性的名称，其实现代码为：

```
Private Sub UserForm_Initialize( )
    cboType.AddItem "标题"
    cboType.AddItem "主题"
    cboType.AddItem "作者"
    cboType.AddItem "关键字"
    cboType.AddItem "注释"
End Sub
```

(5) 窗体中“浏览”按钮的单击事件，包含了打开 Windows 系统浏览对话框的功能，其实现代码为：

```
Private Sub cmdBrowser_Click( )
    '在文本框中显示获得的路径
    txtPath.Text = ReturnFolder(0)
End Sub
```

其中，ReturnFolder 是一个自定义函数，用于显示浏览文件夹的对话框，返回用户选择的文件夹，其实现代码为：

```
Public Function ReturnFolder(lngHwnd As Long) As String
    Dim Browser As BrowseInfo
    Dim lngFolder As Long
```

```
    Dim strPath As String
    Dim strTemp As String

    With Browser
        .hOwner = lngHwnd
        .lpszTitle = "选择工作路径"
        .pszDisplayName = String(MAX_PATH, 0)
    End With

    '用空格填充字符串
    strPath = String(MAX_PATH, 0)
    '调用 API 函数显示文件夹列表
    lngFolder = SHBrowseForFolder(Browser)

    '使用 API 函数获取返回的路径
    If lngFolder Then
        SHGetPathFromIDList lngFolder, strPath
        strTemp = Left(strPath, InStr(strPath, vbNullChar) - 1)

        If (Right(strTemp, 1) <> "\") Then
            strTemp = strTemp & "\"
        End If

        ReturnFolder = strTemp
    End If
End Function
```

(6) 选择“插入”|“模块”菜单项，向当前工程中添加一个标准模块，名称修改为mdlSearchFile，然后打开配套光盘中第 16.2 节(搜索某个文件夹中所有的文件)的程序，将其中的 mdlSearchFile 模块中的代码复制到当前工程的 mdlSearchFile 模块中。

(7) 在“搜索”按钮的单击事件中，查找指定路径下所有的 DWG 文件，判断该文件是否包含指定的信息，如果包含了指定的信息就将其添加到列表框中，其相关代码为：

```
Private Sub cmdSearch_Click( )
    '确保类型和参数不为空
    If Len(txtValue.Text) = 0 Or Len(cboType.Text) = 0 _
        Or Len(txtPath.Text) = 0 Then
        MsgBox "未指定查询条件!"
        Exit Sub
    End If

    '清除前一次查找的结果
    lstFiles.Clear
```

```
'查找路径下所有的 DWG 文件
Dim fileCollection As New Collection
Dim count As Long
count = mdlSearchFile.FindAllFiles("*.dwg", txtPath.Text, fileCollection)

'根据 AutoCAD 的版本来决定使用 ObjectDBX 的版本
Dim objDBX As Object
If Left(Version, 2) = "15" Then
    Set objDBX = CreateObject("ObjectDBX.AxDbDocument.1")
ElseIf Left(Version, 2) = "16" Then
    Set objDBX = CreateObject("ObjectDBX.AxDbDocument.16")
End If

'将符合条件的文件添加到列表框中
Dim i As Integer
For i = 1 To fileCollection.count
    '在图形未被 AutoCAD 打开的情况下才能被 ObjectDBX 打
    If Not IsOpen(fileCollection.item(i)) Then
    objDBX.Open CStr(fileCollection.item(i))

    Dim objSummaryInfo As AcadSummaryInfo
    Set objSummaryInfo = objDBX.SummaryInfo

    If StrComp(cboType.Text, "标题") = 0 Then
        If StrComp(objSummaryInfo.Title, txtValue.Text) = 0 Then
            lstFiles.AddItem fileCollection(i)
        End If
    ElseIf StrComp(cboType.Text, "主题") = 0 Then
        If StrComp(objSummaryInfo.Subject, txtValue.Text) = 0 Then
            lstFiles.AddItem fileCollection(i)
        End If
    ElseIf StrComp(cboType.Text, "作者") = 0 Then
        If StrComp(objSummaryInfo.Author, txtValue.Text) = 0 Then
            lstFiles.AddItem fileCollection(i)
        End If
    ElseIf StrComp(cboType.Text, "关键字") = 0 Then
        If StrComp(objSummaryInfo.Keywords, txtValue.Text) = 0 Then
            lstFiles.AddItem fileCollection(i)
        End If
    ElseIf StrComp(cboType.Text, "注释") = 0 Then
        If StrComp(objSummaryInfo.Comments, txtValue.Text) = 0 Then
              lstFiles.AddItem fileCollection(i)
        End If
    Else
        Dim strValue As String
```

```
                objSummaryInfo.GetCustomByKey cboType.Text, strValue
                If StrComp(strValue, txtValue.Text) = 0 Then
                    lstFiles.AddItem fileCollection(i)
                End If
            End If
        End If
    Next i
End Sub
```

其中，IsOpen 是一个自定义函数，用于判断一个图形是否已经被 AutoCAD 打开，其实现代码为：

```
Private Function IsOpen(ByVal strDwgFile As String) As Boolean
    IsOpen = False

    Dim i As Integer
    For i = 0 To ThisDrawing.Application.Documents.count - 1
        If StrComp (ThisDrawing.Application.Documents.item(i).Name, strDwgFile,
            vbTextCompare) = 0 Then
            IsOpen = True
            Exit Function
        End If
    Next i
End Function
```

(8) 如果列表框中的文件名太长，可能会无法查看到完整的文件名，因此在列表框的双击事件中显示当前选择的文件的名称，其相关代码为：

```
Private Sub lstFiles_DblClick(ByVal Cancel As MSForms.ReturnBoolean)
    MsgBox lstFiles.List(lstFiles.ListIndex)
End Sub
```

(9) 在“退出”按钮的单击事件中添加退出程序的代码：

```
Private Sub cmdExit_Click( )
    End
End Sub
```

(10) 最后，在 ThisDrawing 模块中添加一个启动宏 SearchDwgFile：

```
Public Sub SearchDwgFile( )
    frmMain.Show
End Sub
```

11.3.4 实例效果

(1) 在 AutoCAD 2005 中，打开一个图形，然后将其保存(用 QSAVE 或者 SAVEAS 命令均可)。在保存的文件夹中右击该图形，从弹出的快捷菜单中选择“属性”菜单项，系统会

弹出如图 11.9 所示的结果。

当然，在 AutoCAD 中选择“文件”|“图形特性”菜单项就能手工修改图形的这些属性信息。

(2) 在 VBA 集成开发环境中，执行 SearchDwgFile 宏，系统会弹出如图 11.10 所示的对话框。

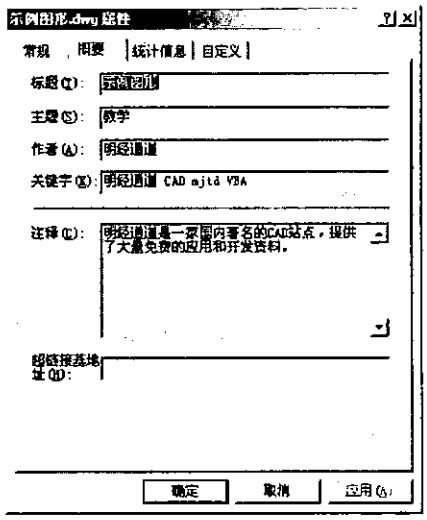


图 11.9　包含了属性信息的图形

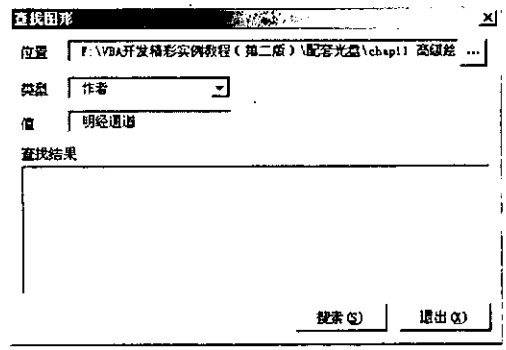


图 11.10　查找图形的界面

(3) 单击浏览按钮选择要查找的文件夹，从“类型”下拉列表框中选择“作者”选项(也可以输入自定义的属性名称)，在“值”文本框中输入“明经通道”，单击“搜索”按钮，能够得到如图 11.11 所示的结果。

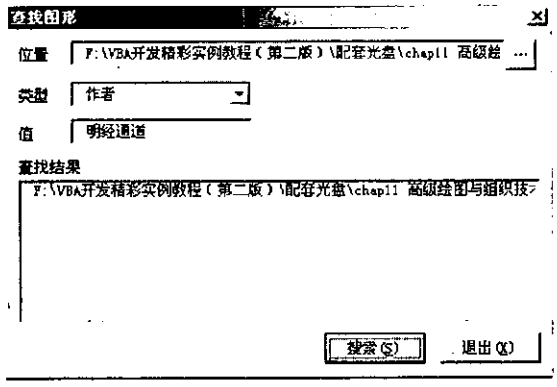


图 11.11　查找图形的结果

(4) 双击列表框中的文件名称，就能在弹出的对话框中查看其完整的路径，如图 11.12 所示。

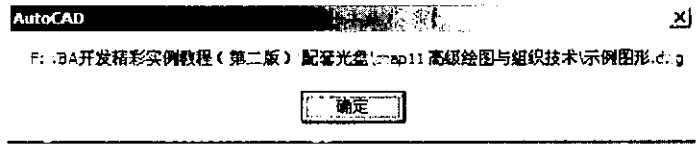


图 11.12 显示文件完整名称

11.3.5 小结

本节使用的是 BeginCommand 事件，而不是采用 BeginSave 事件，理由是，BeginSave 事件除了保存文件时会触发，在通过 Ctrl+C 或 Ctrl+X 来复制或剪切图形时，也会触发该事件(因为复制和剪切也存在保存部分图元到剪贴板，也就是说产生保存的动作)，所以不采用 BeginSave 事件。

SummryInfo 中自定义数据的顺序，是按照数据写入的先后顺序来排列的，所以删除了某一自定义数据，再在原位置写入新的数据，则新的数据会被排到最后面。为了解决这个问题，只能先将所有自定义数据保存成变量，修改了变量后，把自定义数据清空后，重新写入。以保证按要求顺序排列。

SummryInfo 对象在 2004 版本已经提供，只是它没有被显示在帮助文档中，所以没有被人注意。

对于 2004 以前版本，要对 SummryInfo 进行修改，只能通过修改 XRecord 中相应的对象来解决。

11.4 绘图顺序

11.4.1 实例说明

在 AutoCAD 2005 中，可以选择“工具”|“绘图顺序”中的菜单项，调整几个实体相互之间的绘图顺序(其效果是上面的实体遮住下面的实体，如图 11.13 所示)，使图形的显示和打印效果符合一定的要求。

11.4.2 编程思路



图 11.13 文字在填充下面的效果

在 AutoCAD 2005 的 ActiveX 模型中增加了一个 SortentsTable 对象，也就是绘图顺序对象。SortentsTable 对象在 ActiveX 模型中所处的位置很奇怪，绘图顺序是一种图元对象，而表格样式也是图元对象。实际上，虽然它显示在图元对象处，其实只是词典中的一个对象而已，为了方便编程而提出来。

同样道理，表格样式也是词典中的一个对象，而图形属性是扩展记录中的一个对象。获得和使用 SortentsTable 对象都比较简单，因此将在其

体的操作步骤中向大家介绍调整对象绘图顺序的方法。

绘图顺序对象中包含的几个主要方法为：

- MoveAbove：将指定实体的绘图顺序设置为目标实体的上方。
- MoveBelow：将指定实体的绘图顺序设置为目标实体的下方。
- MoveToBottom：将指定实体的绘图顺序设置为最低端。
- MoveToTop：将指定实体的绘图顺序设置为最顶端。
- SetRelativeDrawOrder：设置几个实体的相对的绘图顺序。
- SwapOrder：交换两个实体的绘图顺序。

11.4.3　步骤

(1) 在 AutoCAD 2005 中，打开 VBA 管理器，创建一个新工程，保存在适当的位置。进入 VBA 集成开发环境，双击“工程资源管理器”窗口的 ThisDrawing 图标，打开其代码窗口，在其中添加几个模块级变量(注意区分模块级变量和全局变量)：

```
Dim objPline As AcadPolyline      '多段线
Dim objLine As AcadLine           '直线
Dim objCircle As AcadCircle       '圆
Dim sentityObj As Object          '绘图顺序对象
```

之所以将这几个对象定义为模块级变量，是因为要把创建对象、调整对象绘图顺序分成不同的过程，便于观察程序执行的结果。

(2) 在代码窗口中添加一个宏 CreateEnts，用于向图形窗口中添加三个实体(直线、多段线和圆)，其实现代码为：

```
Public Sub CreateEnts( )
    '设置当前图形显示线宽
    ThisDrawing.Preferences.LineWeightDisplay = True

    '创建一个真彩色对象
    Dim objColor As AcadAcCmColor
    Set objColor = AcadApplication.GetInterfaceObject("AutoCAD.AcCmColor.16")
    Call objColor.SetRGB(80, 100, 244)

    '创建一条多段线
    Dim points(0 To 8) As Double
    points(0) = 7: points(1) = 5: points(2) = 0
    points(3) = -10: points(4) = 5: points(5) = 0
    points(6) = -12: points(7) = 3: points(8) = 0
    Set objPline = ThisDrawing.ModelSpace.AddPolyline(points)
    objPline.Lineweight = acLnWt211
    Call objColor.SetRGB(90, 110, 150)
    objPline.TrueColor = objColor
```

```
    '创建一条直线
    Dim startPoint(0 To 2) As Double
    Dim endPoint(0 To 2) As Double
    startPoint(0) = -5: startPoint(1) = 3.5: startPoint(2) = 0
    endPoint(0) = 10: endPoint(1) = 8: endPoint(2) = 0
    Set objLine = ThisDrawing.ModelSpace.AddLine(startPoint, endPoint)
    objLine.Lineweight = acLnWt211
    Call objColor.SetRGB(50, 80, 230)
    objLine.TrueColor = objColor

    '创建一个圆
    Dim centerPoint(0 To 2) As Double
    Dim radius As Double
    centerPoint(0) = 0: centerPoint(1) = 0: centerPoint(2) = 0
    radius = 5
    Set objCircle = ThisDrawing.ModelSpace.AddCircle(centerPoint, radius)
    objCircle.Lineweight = acLnWt211
    Call objColor.SetRGB(60, 200, 220)
    objCircle.TrueColor = objColor

    ZoomExtents
    AcadApplication.Update

    '获得扩展字典,并从其中获得绘图顺序对象
    Dim eDictionary As Object
    Set eDictionary = ThisDrawing.ModelSpace.GetExtensionDictionary
    '防止 GetObject 调用失败
    On Error Resume Next

    Set sentityObj = eDictionary.GetObject("ACAD_SORTENTS")

    On Error GoTo 0
    '如果字典中不包含绘图顺序对象,就添加一个绘图顺序对象
    If sentityObj Is Nothing Then
        Set sentityObj = eDictionary.AddObject("ACAD_SORTENTS", "AcDbSortentsTable")
    End If
End Sub
```

为了能清楚地观察到实体因绘图顺序不同而产生的“遮盖”效果,可以打开图形的线宽显示,并且为每个实体都指定不同的颜色。真彩色对象的使用比较特殊,因为它是以 OLE 方式调用的一个组件,并不是 ActiveX 模型中的一部分。

创建实体的部分并没有什么特殊,惟一有点不同的是为每个实体都指定了一个真彩色。获得绘图顺序对象的完整代码为:

```
Dim eDictionary As Object
Set eDictionary = ThisDrawing.ModelSpace.GetExtensionDictionary
'防止 GetObject 调用失败
On Error Resume Next

Set sentityObj = eDictionary.GetObject("ACAD_SORTENTS")

On Error GoTo 0
'如果字典中不包含绘图顺序对象,就添加一个绘图顺序对象
If sentityObj Is Nothing Then
        Set sentityObj = eDictionary.AddObject("ACAD_SORTENTS", "AcDbSortentsTable")
End If
```

首先获得模型空间的扩展字典,从字典中获得键为 ACAD_SORTENTS 的字典项,就是绘图顺序对象。如果字典中尚未包含键为 ACAD_SORTENTS 的字典项,就为其添加一个绘图顺序对象,也就是添加了这个字典项。

(3) 添加一个宏 SetRelativeOrder,用于排列三个实体的相对绘图顺序,其实现代码为:

```
Public Sub SetRelativeOrder( )
    '数组中第一个元素将被排在最下方,依次向上
    Dim arr(0 To 2) As AcadObject
    Set arr(0) = objLine
    Set arr(1) = objCircle
    Set arr(2) = objPline

    sentityObj.SetRelativeDrawOrder arr

    '重生成视口的内容,反映出绘图顺序变化的结果
    ThisDrawing.Regen acActiveViewport
End Sub
```

需要注意的是,SetRelativeDrawOrder 方法会按照数组中元素的顺序来决定这些实体的绘图顺序,数组中的第一个元素会被放在最下方,第二个元素在第一个元素上面,依此类推。在调整绘图顺序之后,必须执行重生成的操作,反映调整绘图顺序之后的结果。

(4) 添加一个宏 SetEntsOrder,用于调整多个实体相对于一个实体的绘图顺序,其实现代码为:

```
Public Sub SetEntsOrder( )
    Dim arr(0 To 1) As AcadObject
    Set arr(0) = objCircle
    Set arr(1) = objPline

    sentityObj.MoveBelow arr, objLine
    ThisDrawing.Regen acActiveViewport
End Sub
```

(5) 添加一个宏 Swap2EntsOrder,用于交换两个实体的绘图顺序,其实现代码为:

```
Public Sub Swap2EntsOrder( )
    sentityObj. SwapOrder objPline, objLine
    ThisDrawing. Regen acActiveViewport
End Sub
```

11.4.4 实例效果

(1) 在 VBA 集成开发环境中,执行 CreateEnts 宏,能够在图形窗口得到如图 11.14 所示的结果。此时,三个实体从下到上的顺序是多段线、直线、圆。

之所以呈现这种绘图顺序,是因为在 AutoCAD 中,后来创建的实体默认情况下会在先创建的实体的上层。

(2) 执行 SetRelativeOrder 宏来调整三个实体的绘图顺序,得到如图 11.15 所示的结果。

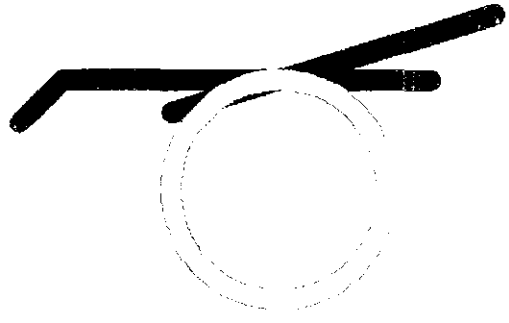


图 11.14 创建实体之后的绘图顺序



图 11.15 调整绘图顺序的结果(一)

(3) 执行 SetEntsOrder 宏来调整三个实体的绘图顺序,得到如图 11.16 所示的结果。

(4) 执行 Swap2EntsOrder 宏来调整三个实体的绘图顺序,得到如图 11.17 所示的结果。

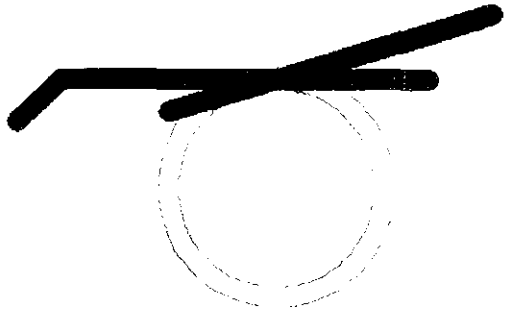


图 11.16 调整绘图顺序的结果(二)

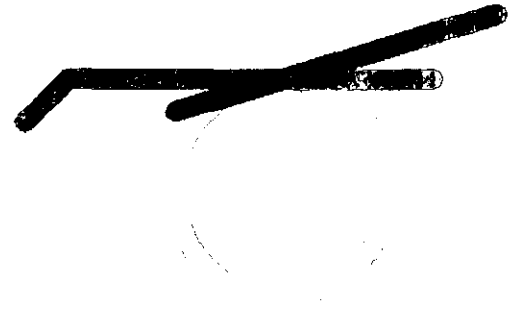


图 11.17 调整绘图顺序的结果(三)

11.4.5 小结

在 SortentsTable 对象中提供了 Block 方法,用来获得绘图顺序对象对哪个块有效,默认情况下是模型空间。而在对象模型中却没有提供能指定其他块作为绘图顺序的对象,这样的话,用户就无法指定某个布局给 SortentsTable 对象,以便排序。

所以在编程中，对模型空间的图元进行排序比较容易，而在布局空间中排序则不是一件容易的事。

SortentsTable 对象提供了几种排序的方法，置最前、置最后、置某对象前、置某对象后，(MoveAbove、MoveBelow 、MoveToBottom 、MoveToTop)，另外还提供了交换两个对象的顺序(SwapObjects)。再结合它提供了只对部分图元排序这样的功能，可以编制出比系统提供的调整绘图顺序命令更强大的功能。

第 12 章　VBA 与应用程序

在使用 VBA 开发应用程序时，经常需要调试应用程序，本章将给出一些调试应用程序的技巧。一个好的程序必须具有较好的健壮性，VBA 应用程序的错误处理一节就将注意力集中在提高程序的容错能力。

DwgThumbnail、Accont、Slide、MSScript 是几个很有特色的组件，能够为 VBA 程序增加一些特殊的功能，本章的几个实例就详细描述了这几个组件的使用，在配套光盘中可以获得这些组件。

在 VBA 中不包含定时器控件，上一本书中用了第三方的 XTimer 控件来代替，本章将要创建自己的定时器控件！

12.1　调试 VBA 应用程序

12.1.1　实例介绍

在编写所有程序的过程中，都会面临一个非常费时间的工作－调试程序，再好的程序员也不可能每次都写出完全正确执行的代码。

本节将介绍在编程过程中会遇到的三种错误，以及调试各种错误的方法和技巧，帮助用户高效率地找出程序中的错误。

12.1.2　编程思路

在开发应用程序的过程中，会遇到如下所示的三种错误。

- 编译错误：在编写代码或者执行程序之前显示，一般是由于语法错误或者方法和属性的非法使用。这种错误的排除比较容易，编译器就能找出这类错误。
- 运行错误：运行错误比编译错误隐蔽一些，要在程序运行的过程中才会出现。在执行语法正确的代码时，如果因为当前环境条件而无法执行就会发生运行错误。例如，试图使用 Kill 函数删除一个并不存在的文件，或者向 Documents 集合的 Open 方法传递了一个并不存在的文件，就会发生运行错误。
- 逻辑错误：执行一段语法正确的代码，并且在执行过程中未产生任何的运行错误，但是却得到了不正确的结果，这就是产生了逻辑错误。这种错误一般是由于编程人员的疏忽，或者算法自身的缺陷引起，是最难跟踪和调试的错误。

处理这三种错误有一些基本的方针：

- 编译和运行之前消除编译错误，这一点编译器会自动完成。
- 编程过程中，应该对可能出现的运行错误事先进行错误处理，增强程序的健壮性，这在下一节还会详细讨论。

- 使用断点、单步执行和监视等方法观察程序运行的整个过程(或可能出现问题的部分),与正确的情况进行对照,找出逻辑错误。

12.1.3　步骤

(1) 在 AutoCAD 2005 中,打开 VBA 管理器,创建一个新工程,保存在适当的位置。进入 VBA 集成开发环境,双击"工程资源管理器"窗口中的 ThisDrawing 图标,打开其代码窗口,在其中添加一个宏 SyntaxError,其实现代码为:

```
Public Sub SyntaxError( )
    Dim strMsg As strong
    strMsg = "Hello, VBA!"
    MsgBox strMsg
End Sub
```

(2) 将光标置于 SyntaxError 宏内部,按下 F5 键执行该宏,系统会弹出如图 12.1 所示的对话框,提示发生了错误,并且在代码窗口中标识出错误的位置。

(3) 如果在 SyntaxError 宏内部,输入 If Len(strMsg) <> 0,然后按下 Enter 键,系统会弹出如图 12.2 所示的对话框。这是编译错误的另一种情况,在撰写代码过程中,被编译器所发现的错误会自动提示,这只有在选择了"自动语法检测"功能之后才会启用。在 VBA 集成开发环境中选择"工具"|"选项"菜单项,在弹出的"选项"对话框中可以设置,如图 12.3 所示。

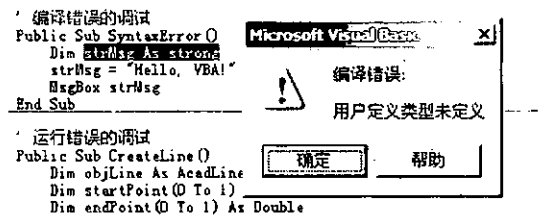


图 12.1　编译错误的提示

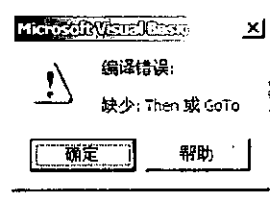


图 12.2　实时提示语法错误

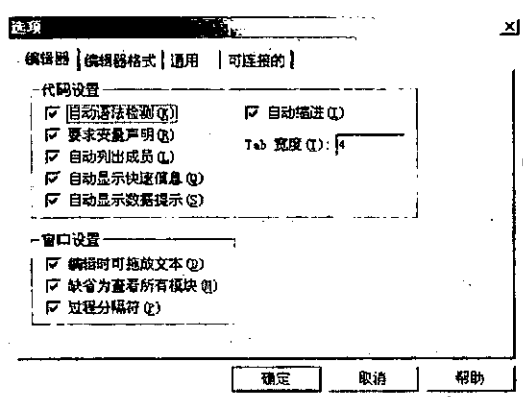


图 12.3　启用自动语法检测功能

(4) 在 VBA 集成开发环境中,选择"插入"|"用户窗体"菜单项,向工程中添加一个窗体,在窗体上放置一个按钮,双击该按钮,编辑其单击事件的执行代码:

```
Private Sub CommandButton1_Click( )
    Dim ptPick As Variant
    ptPick = ThisDrawing.Utility.GetPoint(, "拾取点:")
    MsgBox ptPick(0) & vbCrLf & ptPick(1) & vbCrLf & ptPick(2)
End Sub
```

代码的意图很简单，要提示用户在 AutoCAD 图形窗口中选择一个点，并且弹出一个对话框，显示该点的坐标。

(5) 在“工程资源管理器”窗口中双击该窗体的图标，显示其窗体设置的界面，然后按下 F5 键运行该窗体(在测试过程中可以直接显示该窗体，而不必从一个宏中使用语句 Form1.Show 来显示窗体)，系统会弹出如图 12.4 所示的对话框。

(6) 在对话框中单击“拾取点”按钮，系统会弹出如图 12.5 所示的对话框，指出在执行程序的过程中发生了错误。此时可以单击“调试”按钮，系统会在代码窗口中显示发生错误的位置，如图 12.6 所示。



图 12.4 显示窗体

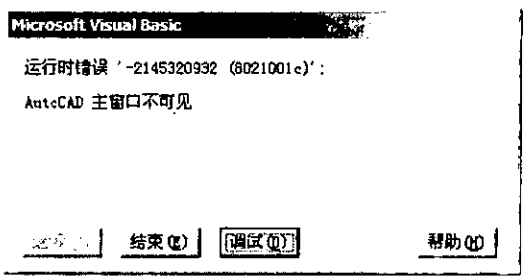


图 12.5 提示发生运行错误

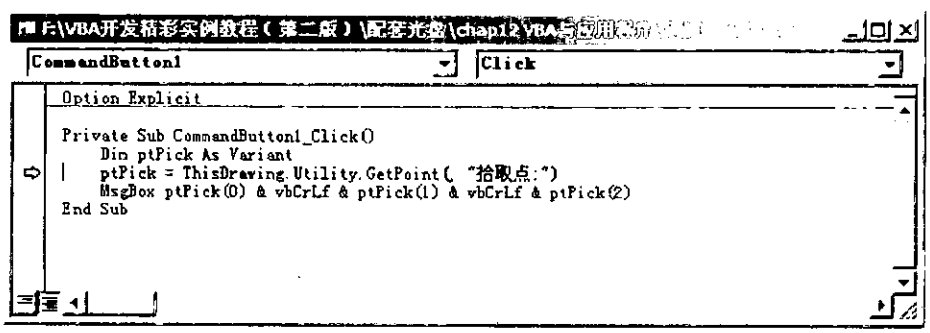


图 12.6 显示发生错误的位置

这是初学者经常遇到的一个错误。其解决的方法很简单，只需在 GetPoint 执行前将当前窗体隐藏，控制权交给 AutoCAD，执行之后再显示该窗体即可，相关代码为：

```
Private Sub CommandButton1_Click( )
    Dim ptPick As Variant
    Form1.Hide      '隐藏窗体,将控制权交给 AutoCAD

    ptPick = ThisDrawing.Utility.GetPoint(, "拾取点:")
```

```
    MsgBox ptPick(0) & vbCrLf & ptPick(1) & vbCrLf & ptPick(2)
    Form1.Show    '显示窗体
End Sub
```

(7) 最后，把目光转向最难调试的逻辑错误。下面就以笔者编写 3.6 节(多段线)反转多段线的代码过程中调试逻辑错误的真实经历来阐述其中的方法和技巧。代码如下(已经排除了编译错误和运行错误)：

```
Public Sub RevPline( )
    Dim ent As AcadEntity
    Dim pnt As Variant
    Dim NewCoord( ) As Double                          '新的顶点坐标数组
    Dim i As Integer

    '拾取多段线
    On Error Resume Next
    Do
        ThisDrawing.Utility.GetEntity ent, pnt, "选择多段线："
        If Err Then Exit Sub
        OIf TypeName(ent) Like "IAcad * Polyline" Then Exit Do
    Loop

    '反转多段线的顶点
    Dim Coord As Variantp                              '原来的顶点坐标数组

    If TypeOf ent Is AcadLWPolyline Then
        Coord = ent.Coordinates
        ReDim NewCoord(UBound(Coord)) As Double
        For i = 0 To UBound(Coord) - 1 Step 2
            NewCoord(UBound(Coord) - i - 1) = Coord(i)
            NewCoord(UBound(Coord) - i) = Coord(i + 1)
        Next
    ElseIf TypeOf ent Is AcadPolyline Then
        Coord = ent.Coordinates
        ReDim NewCoord(UBound(Coord)) As Double
        For i = 0 To UBound(Coord) - 1 Step 3
            NewCoord(UBound(Coord) - i - 2) = Coord(i)
            NewCoord(UBound(Coord) - i - 1) = Coord(i + 1)
            NewCoord(UBound(Coord) - i) = Coord(i + 2)
        Next
    End If

    ent.Coordinates = NewCoord                         '更新顶点坐标集合

    '修改凸度
    Dim bulgeCollection As New Collection
    Set bulgeCollection = GetAllBulges(ent)            '获得凸度集合
```

```
    Dim newBulges As New Collection                    '新的凸度集合
    Set newBulges = RevCollection(bulgeCollection)     '反转集合
    Call SetAllBulges(ent, newBulges)                  '重新设置凸度

    ThisDrawing.Regen acActiveViewport

    End
End Sub
```

(8) 在 AutoCAD 图形窗口中,创建如图 12.7 所示的多段线用于反转多段线的测试。执行 RevPline 宏对多段线进行反转,得到如图 12.8 所示的结果。

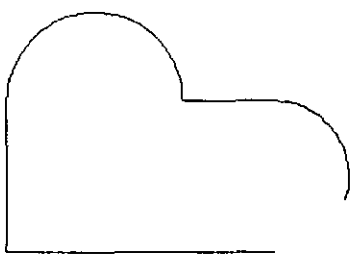


图 12.7 创建用于测试的多段线



图 12.8 使用 RevPline 宏反转多段线的结果

结果很明显是错误的,多段线反转之后形状应该保持不变,可是这里不仅形状发生变化,甚至多出了一个圆弧段!

(9) 由于多段线的顶点位置并未发生变化,因此确定是修改凸度的部分有问题,将光标放置在下面的代码行:

```
Set bulgeCollection = GetAllBulges(ent)
```

按下 F9 快捷键(或者选择"调试"|"切换断点"菜单项),在该代码行设置一个断点,当程序执行到这一行的时候就会停止执行。在代码窗口中选中 bulgeCollection,然后右击,从弹出的快捷菜单中选择"添加监视"菜单项,系统会弹出如图 12.9 所示的"添加监视"对话框,单击"确定"按钮,添加对 bulgeCollection 变量的监视,系统会在"监视窗口"对话框中添加该变量。

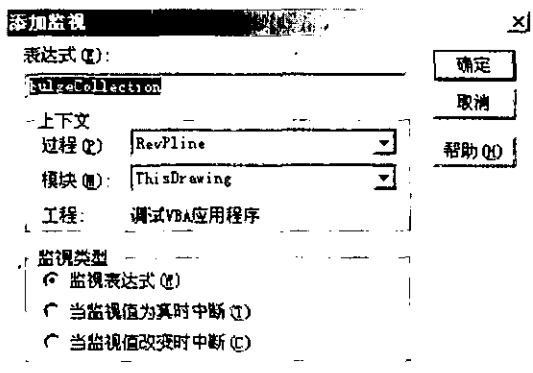


图 12.9 对 bulgeCollection 变量添加监视

(10) 在 AutoCAD 图形窗口中，按下 Ctrl＋Z 组合键撤销反转的操作，恢复原来的形状。切换到 VBA 集成开发环境，执行 RevPline 宏，在图形窗口中选择多段线，系统会在断点处停下，如图 12.10 所示。

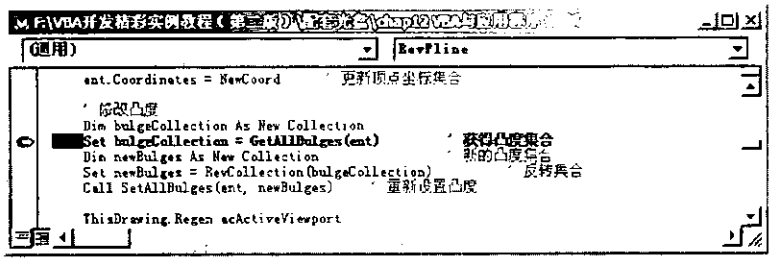


图 12.10　在断点处程序停止执行

(11) 按下 F8 快捷键(或选择“调试”|“逐语句”菜单项)，程序会向下执行一步，进入 GetAllBulges 函数，由于已经确定该函数的实现不会有问题，可以选择“调试”|“跳出”菜单项，重新回到 RevPline 宏的内部，此时通过 GetAllBulges 函数中的赋值，已经可以在“监视窗口”对话框中查看 bulgeCollection 变量的值，如图 12.11 所示。

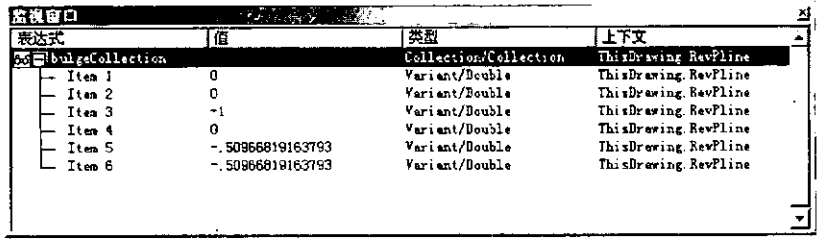


图 12.11　显示 bulgeCollection 变量的值

(12) 在下面的代码行单击：

```
Call SetAllBulges(ent, newBulges)
```

选择“调试”|“运行到光标处”菜单项，为 newBulges 变量添加监视，查看执行 RevCollection 函数(反转凸度集合)反转 bulgeCollection 变量的结果，如图 12.12 所示。

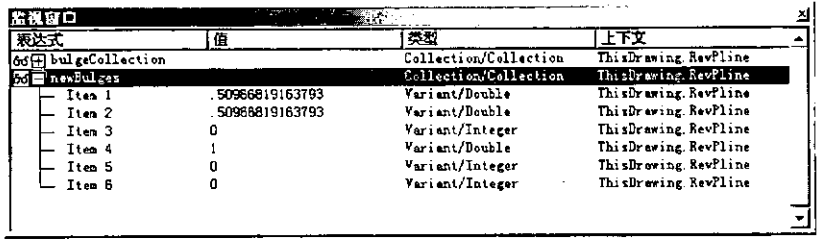


图 12.12　查看反转凸度集合的结果

该函数执行的结果是正确的，因为 RevCollection 函数会自动对非零的凸度值取相反数。凸度为 0 表示直线段，从凸度值的集合也可以看出反转之后的多段线确实是包含了 3 个圆弧段。

(13) 从原来多段线的形状来分析，反转之后的多段线应该在顶点 1、3 处凸度值为非零，其他顶点处的凸度值都为 0。看来只将凸度值反转和取相反数还是不够的，应该将反转后的凸度集合的第一个元素删除，并在最后添加一个元素 0，对应于原来的凸度集合，就应该删除凸度集合的最后一个元素，并在前面添加一个元素 0。于是，在 RevPline 宏中添加下面的语句(添加的语句为粗体)：

```
Dim bulgeCollection As New Collection
Set bulgeCollection = GetAllBulges(ent)                    '获得凸度集合
bulgeCollection.Remove bulgeCollection.Count               '删除最后一个元素
bulgeCollection.Add 0.,1                                   '在开始处添加一个新元素
Dim newBulges As New Collection                            '新的凸度集合
Set newBulges = RevCollection(bulgeCollection)             '反转集合
Call SetAllBulges(ent, newBulges)                          '重新设置凸度
```

(14) 单击“标准”工具栏的“重新设置”按钮，或者选择“运行”|“重新设置”菜单项，停止当前的调试，将光标放置在包含断点的语句上，按下 F9 快捷键，删除当前语句中的断点，重新执行该宏，得到正确的结果。

至此为止，调试顺利完成。

12.1.4 小结

1. 其他的调试方法和工具

在 VBA 中还可以使用本地窗口来查看变量在某个时刻的值，当程序执行到某个位置时(单步执行或者使用断点均能使程序在某个位置暂时停止执行)，选择“视图”|“本地窗口”菜单项，系统会弹出如图 12.13 所示的窗体，其中显示了当前所有在生存期内的变量的值。

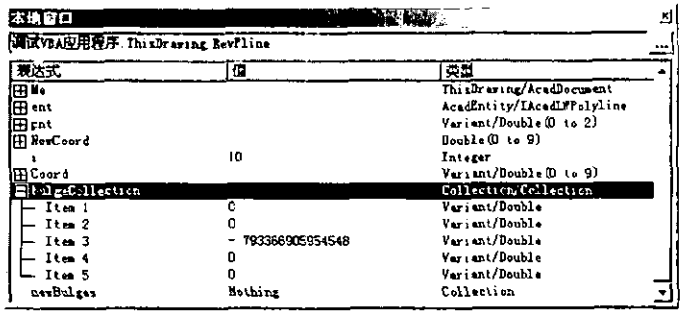


图 12.13 查看当前所有在生存期内的变量的值

此外，在程序执行过程中，如果需要在某个位置查看变量的值，还可以使用 Debug.Print 语句，该语句会将变量或表达式的值打印到立即窗口中。程序执行完毕，可以选择“视图”|“立即窗口”菜单项，在弹出的立即窗口中查看变量的值。

2. 努力编写无错的代码

尽管不可能总是编写出无错的代码，但是一些良好的编程习惯确实能够有效地减少程序中的错误，下面列出几个非常有用的编程习惯。

- 总是使用 Option Explicit 语句。如果程序中存在大量没有声明就直接使用的变量，会使程序变得非常难读(变量一般会在声明时注释其含义)，而且在这种情况下系统无法检测到变量拼写错误，在程序调试过程中为这样的错误花费时间实在是不值得。因此，建议在 VBA 开发环境中选择“工具”|“选项”菜单项，在弹出的“选项”对话框中，切换到“编辑器”选项卡，选中“要求变量声明”复选框，这样创建新的代码模块时系统会自动添加 Option Explicit 语句，如图 12.14 所示。

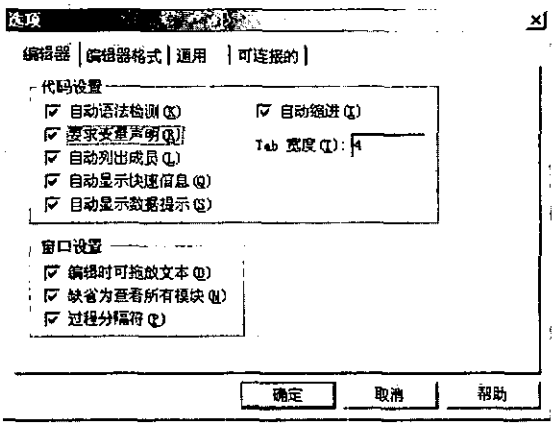


图 12.14　启用“要求变量声明”功能

- 将程序的功能分解成多个函数来实现，并且尽可能早地测试代码。调试一小段代码总是比调试上百行的代码简单，在编写某个函数之后应该立即测试，这样最终消耗在调试程序上的时间会大大减少。同时，将程序的功能分解到函数中，还有利用代码的重用。
- 正确、认真使用注释和说明。很多人写程序的时候不愿意写注释和设计文档，这对于程序设计是非常危险的。现代程序设计最讲究的就是代码的可重用性，如果一段代码缺少注释和说明文档，要使用这段代码的程序员读懂这段代码所花费的时间可能多于重写相同功能的代码，甚至过一段时间编写这段程序的程序员自己也读不懂那些代码了。
- 注意代码的格式和变量命名。好的变量命名会加强程序的可读性，按照标准缩进格式编排的代码更容易让人读懂。
- 小心常见的 Dim 语句错误。初学者多会用“Dim i, j As Integer”来声明两个整型变量，实际的结果并不是这样，上面的语句只会将 j 声明为整型，i 将会被作为 Variant 类型的变量。要在一个 Dim 语句中声明多个变量，必须为每个变量指定具体的类型，例如“Dim i As Integer, j As Integer”。
- 在需要的场合使用断言来提高程序的健壮性。何谓断言？断言是强制某一逻辑条件成立的语句，如果条件不成立，则会提示错误的信息，并且停止程序的执行。由于 VBA 中没有断言函数，因此编写了一个函数 Assert 来实现断言的功能。下面的代码演示了断言函数的用法：

```
Public Sub Math( )
    Dim a As Double, b As Double
    a = 10
    b = 0

    Call Assert(Abs(b) > 0.0001)
    MsgBox a/b
End Sub

'断言函数
Public Sub Assert(ByVal bCondition As Boolean, Optional strMessage As String = "断言失败!")
    If Not bCondition Then
        MsgBox strMessage, vbCritical, "断言"
        Stop    '暂停程序的执行
    End If
End Sub
```

执行 Math 宏，由于变量 b 的值为 0，不满足断言条件 Abs(b) > 0.0001，则在执行过程中会弹出如图 12.15 所示的对话框，单击“确定”按钮之后，程序会暂停执行。此时程序员就可以检查断言函数调用之前的部分，很方便地找到程序中可能存在的错误。



图 12.15 断言失败的提示信息

12.2 VBA 应用程序的错误处理

12.2.1 实例说明

一个好的应用程序必须有好的健壮性，也就是在程序执行过程中遇到错误的时候能够消除错误，不影响应用程序的继续运行。一个最简单的例子就是，AutoCAD 中所有的命令执行过程中都可以按下 Esc 键退出执行，按下 Esc 键对于应用程序来说就是一种意外的中断，如果最终应用程序中包含这样的一个宏：

```
Public Sub CreateLine( )
    Dim ptStart As Variant, ptEnd As Variant

    ptStart = ThisDrawing.Utility.GetPoint(, "拾取起点：")
    ptEnd = ThisDrawing.Utility.GetPoint(ptStart, "拾取终点：")

    ThisDrawing.ModelSpace.AddLine ptStart, ptEnd
End Sub
```

这个程序仍然存在着缺陷，当系统提示“拾取起点”或者“拾取终点”的时候，如果用户按下了 Esc 键，系统会出现运行错误，如图 12.16 所示。

本节的内容正是要处理程序中的各种错误，创建具有很强的健壮性的应用程序。

12.2.2　编程思路

从处理错误的原则上讲，VBA 中的错误处理可以分为两种：

- 出错前处理：一般在和用户交互之后立即检查用户输入数据的有效性，或者在调用函数的时候检查参数的有效性，其目的都是为了避免发生错误。
- 出错后处理：某些函数或方法执行的过程中，由于要和用户交互或者进行其他操作，这时发生错误的概率非常大，而且这种错误是无法预先避免的，一般可以使用 On Error 系列语句在发生错误后进行处理。

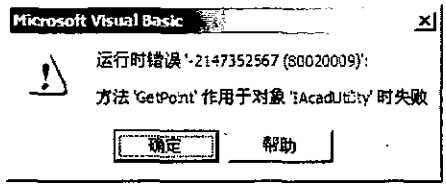


图 12.16　用户按下 Esc 键出现运行错误

12.2.3　步骤

(1) 在 AutoCAD 2005 中，打开 VBA 管理器，创建一个新工程，保存在适当的位置。进入 VBA 集成开发环境，双击“工程资源管理器”窗口中的 ThisDrawing 图标，打开其代码窗口，创建一个函数 AddCircle，其相关代码为：

```
Private Function AddCircle(ByVal ptCenter As Variant, ByVal radius As Double) As AcadCircle
    If Not VarType(ptCenter) = vbArray + vbDouble Or _
        (UBound(ptCenter) - LBound(ptCenter)) <> 2 Or radius <= 0 Then
        Debug.Print "执行 AddCircle 函数时参数无效!"
        Exit Function
    End If

    AddCircle = ThisDrawing.ModelSpace.AddCircle(ptCenter, radius)
End Function
```

在执行 AddCircle 方法之前，先对 ptCenter 进行检查，保证其存储了一个包含 3 个元素的 Double 类型数组，并且确保 radius 变量的值大于 0。如果有一个条件不满足，就会导致 AddCircle 方法产生错误。对程序进行错误处理时，MsgBox 函数常用于提示用户修改输入的参数值，Debug.Print 语句多用于提示程序员。



图 12.17　设计窗体中的控件

(2) 在 VBA 集成开发环境中，选择“插入”|“用户窗体”菜单项，向工程中添加一个对话框，并在其中添加如图 12.17 所示的控件。

(3) 如果“参数 1”文本框中要求输入 0～12 之间的整数，那么就可以在 KeyPress 事件中限制用户只能输入数字，在 BeforeUpdate 事件中检查文本框内的数值是否在 0～12 内，其相关代码为：

```
Private Sub txtParam1_BeforeUpdate(ByVal Cancel As MSForms.ReturnBoolean)
    If CInt(txtParam1.Text) < 0 Or CInt(txtParam1.Text) > 12 Then
        MsgBox "参数 1 取值必须在 0～12 之间!"
        txtParam1.Text = ""
```

```
    End If
End Sub

Private Sub txtParam1_KeyPress(ByVal KeyAscii As MSForms.ReturnInteger)
    If (KeyAscii < Asc("0") Or KeyAscii > Asc("9")) Then
        KeyAscii = 0
    End If
End Sub
```

Asc 函数会返回一个 Integer 数值，表示字符串中首字符的字符代码。

(4) 在对话框的“确定”按钮的单击事件中，需要确保所有文本框都不为空，可以使用下面的代码来进行处理：

```
Private Sub cmdOk_Click( )
    Dim objControl As control
    For Each objControl In UserForm1.Controls
        If TypeOf objControl Is TextBox Then
            If Len(objControl.Text) = 0 Then
                MsgBox "文本框的值不能为空!"
                Exit Sub
            End If
        End If
    Next objControl

    '进行其他的处理
End Sub
```

(5) 安全创建选择集在 VBA 编程中是一个非常常见的错误处理，其错误处理的方法对于所有的集合对象都适用，相关代码为：

```
Public Sub CreateSelectionSet( )
    '安全创建选择集
    On Error Resume Next
    Dim SSet As AcadSelectionSet
    If Not IsNull(ThisDrawing.SelectionSets.Item("Example")) Then
        Set SSet = ThisDrawing.SelectionSets.Item("Example")
        SSet.Delete
    End If
    Set SSet = ThisDrawing.SelectionSets.Add("Example")

    '删除选择集
    SSet.Delete
End Sub
```

其中，On Error Resume Next 语句的作用是当一个运行错误发生时，控件转到紧接着发生错误的语句之后的语句，并在此继续运行。使用该语句之后，带来的负面作用是给调试代码增加困难，因为使用该语句之后将不会出现所有的运行错误提示。

如果在执行包含 On Error Resume Next 语句的宏时产生了意想不到的结果，并且执行过程中未遇到任何的错误提示，就可以暂时去掉 On Error Resume Next 语句（将其设置为注释即可），先对后面的代码进行调试，调试完毕之后再取消 On Error Resume Next 语句的注释即可。

（6）和用户交互过程中，一般需要处理的错误是用户按下 Enter 或者 Esc 键，或者不正确地输入了其他参数，对 GetEntity 方法的错误处理就是一个非常典型的例子，其相关代码为：

```
Private Const VK_ESCAPE = &H1B '代表 Esc 键
Private Declare Function GetAsyncKeyState Lib "user32" (ByVal vKey As Long) As Integer

'功能：判断用户是否按下某一个键
'输入：代表键的常量(从 API Viewer 中获得)
'调用：API 函数 GetAsyncKeyState
'返回：如果用户按下了指定的键，返回 True；否则返回 False
'示例：If CheckKey(&H1B) = True Then do sth
Private Function CheckKey(lngKey As Long) As Boolean
  If GetAsyncKeyState(lngKey) Then
    CheckKey = True
  Else
    CheckKey = False
  End If
End Function

'对 GetEntity 方法进行错误处理
Public Sub GetEntity( )
    Dim objSelect As AcadEntity
    Dim basePnt As Variant
    '选择对象并判断类型
    On Error Resume Next

Retry:
    ThisDrawing.Utility.GetEntity objSelect, basePnt, "选择所要添加属性的对象："

    '处理按下 Esc 键的错误
    If objSelect Is Nothing Then
        If CheckKey(VK_ESCAPE) = True Then
            Exit Sub
        Else
            GoTo Retry
        End If
    End If

    '处理未选择到实体的错误
    If Err <> 0 Then
        Err.Clear
```

```
        GoTo Retry
    End If

    '执行操作
    Debug.Print objSelect.ObjectName
End Sub
```

(7) 最后对本节开头提到的 CreateLine 宏进行错误处理，确保用户创建直线失败不会影响到程序的继续运行，添加错误处理之后的宏包含了下面的代码：

```
Public Sub CreateLineR2( )
    On Error GoTo ERRHANDLE
    Dim ptStart As Variant, ptEnd As Variant

    ptStart = ThisDrawing.Utility.GetPoint(, "拾取起点：")
    ptEnd = ThisDrawing.Utility.GetPoint(ptStart, "拾取终点：")

    ThisDrawing.ModelSpace.AddLine ptStart, ptEnd
    Exit Sub        '不要缺少这个语句

ERRHANDLE:
    ThisDrawing.Utility.Prompt vbCrLf & "指定点的过程中遇到错误，未创建任何直线!" & vbCrLf
End Sub
```

上面的代码中，使用 On Error GoTo 语句保证在程序发生错误之后转到指定的代码行继续执行，一般的错误处理语句都放在过程(函数)的最后。

使用这种结构的错误处理时，最常见的问题就是忽略了错误处理语句前面的 Exit Sub，如果缺少这个语句，在程序执行正确的情况下仍然会提示“指定点的过程中遇到错误，未创建任何直线!”，这就引入了另外一个错误。

12.2.4 实例效果

(1) 在 VBA 集成开环境中，打开 ThisDrawing 模块的代码窗口，可以在其中添加一个宏测试 AddCircle 函数，其相关代码为：

```
Sub TestAddCircle ( )
    Dim ptCenter(0 To 1) As Double
    ptCenter(0) = 0: ptCenter(1) = 0
    AddCircle ptCenter, 3
End Sub
```

执行 TestAddCircle 宏之后，系统会在立即窗口中打印语句“执行 AddCircle 函数时参数无效!”。

(2) 在“工程资源管理器”窗口中双击 UserForm1 图标，打开对话框的设计界面，按下 F5 键运行该窗体。在“参数 1”文本框中输入英文字母时，不能向其中添加任何内容。如果输入的整数值不在 0～12 之间，当焦点移动到其他的文本框时，系统会弹出如图 12.18 所示的对话框，并且将前一文本框的值设置为空字符串。

(3) 如果对话框中的任何一个文本框为空，在单击“确定”按钮时都会弹出如图 12.19 所示的对话框。

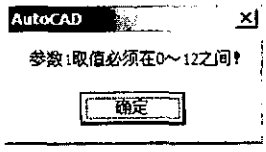


图 12.18　参数值无效的提示



图 12.19　文本框值为空的错误提示

(4) 执行 GetEntity 宏，系统会提示用户选择对象，如果用户单击时没有选择到任何实体，系统会继续提示用户选择对象。只有在用户选择到一个实体或者按下 Esc 键时，系统会停止提示用户选择对象。

(5) 执行 CreateLineR2 宏，系统会提示用户输入直线的起点、终点，如果在输入点时用户按下了 Esc 或者 Enter 键，系统都会在命令行提示“指定点的过程中遇到错误，未创建任何直线!”。

12.2.5　小结

错误处理的基本目的为了提高程序的健壮性，但是在 VBA 中还可以完成一些特殊的任务，例如在 2.4 节(和用户交互)中，就曾经利用错误处理的原理在使用 GetPoint 方法的同时允许用户输入关键字响应某些选项。

12.3　显示缩略图——DwgThumbnail 组件

12.3.1　实例效果

在 AutoCAD 2005 中，选择“工具”|“选项”菜单项，打开“选项”对话框，切换到“打开和保存”选项卡，如图 12.20 所示。

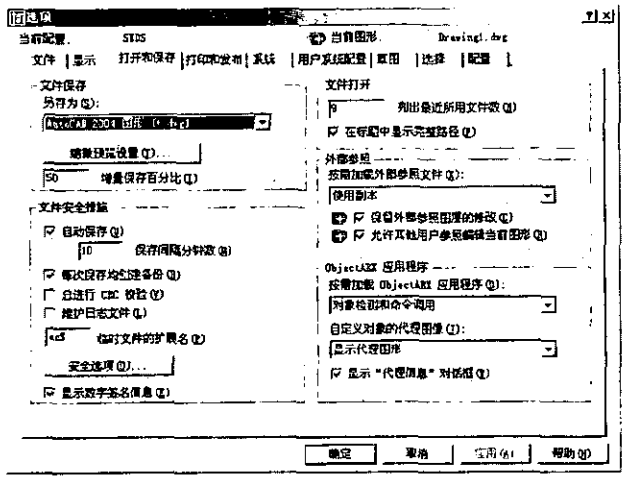


图 12.20　“选项”对话框

单击“缩微预览设置”按钮，系统弹出如图 12.21 所示的“缩微预览设置”对话框。

图 12.21 中的设置，决定了是否在文件中保存预览图，AutoCAD 默认是保存预览图。使用 DwgThumbnail 组件，可以在用户的窗体中显示 DWG 文件保存的缩略图。本实例的运行界面如图 12.22 所示，其中包含了选择图形文件和预览缩略图。

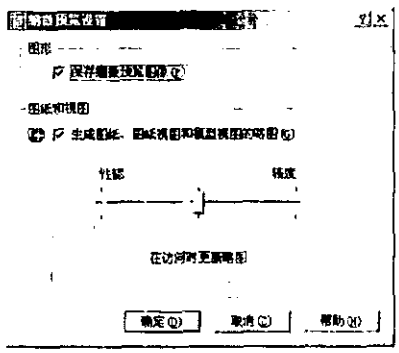


图 12.21 “缩略预览设置”对话框

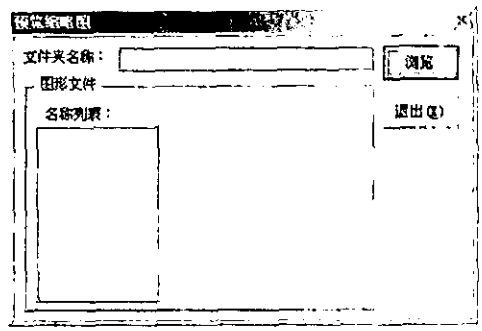


图 12.22 程序运行界面

12.3.2 编程思路

要预览图形文件中的缩略图，只需将 DwgThumbnail 控件的 DwgFileName 属性设置为图形文件的名称，例如 c:\r14\support\chroma.dwg。

12.3.3 步骤

(1) 在 AutoCAD 2005 中，打开“VBA 管理器”，创建一个新工程，保存在适当的位置，进入 VBA 集成开发环境。

(2) 选择“插入”|“用户窗体”菜单项，向程序中添加一个用户窗体，附加控件 AutoCAD DwgThumbnail Control，并在其中放置如图 12.23 所示的控件（列表右侧是一个 DwgThumbnail 控件）。

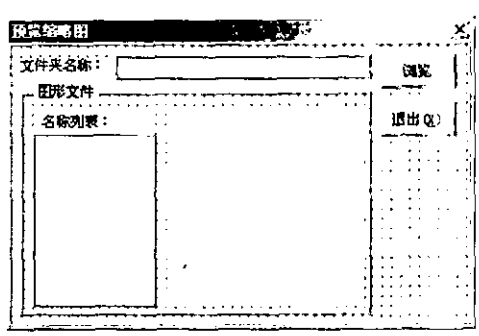


图 12.23 添加窗体控件

提示：要向窗体中添加 DwgThumbnail 控件，可先从配套光盘中复制 DwgThumbnail 组件.zip 文件，并将其解压到计算机中的某个文件夹中。在 VBA 集成开发环境中，选择“工具”|“引用”菜单项，在弹出的“引用”对话框中单击“浏览”按钮，选择解压后得到的 DwgThumbnail.ocx 文件，在“可使用的引用”列表框中选中 DwgThumbnail 组件，如图 12.24 所示。关闭“引用”对话框，选择“工具”|“附加控件”菜单项，在弹出的“附加控件”对话框中，选中 DwgThumbnail 控件，如图 12.25 所示。此时，DwgThumbnail 控件就会出现在工具箱中，可以将其添加到窗体中。

(3) 选择“插入”|“模块”菜单项，向程序中添加一个标准模块，在其中添加宏的启动代码：

```
Public Sub ShowPreview( )
    UserForm1. Show
End Sub
```

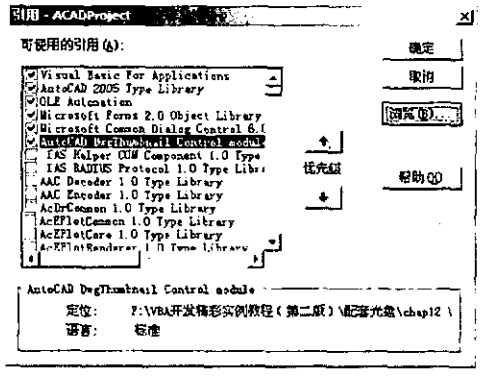


图 12.24　添加 DwgThumbnail 组件的引用

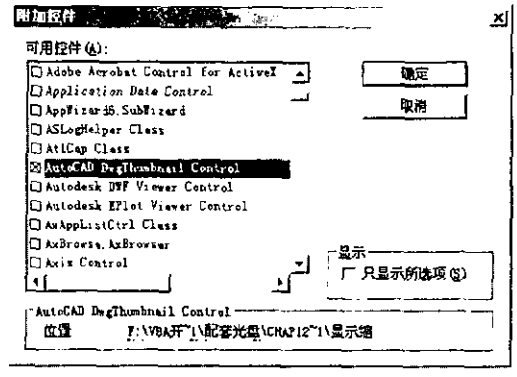


图 12.25　附加 DwgThumbnail 控件

(4) 在窗体的代码窗口中,添加 API 函数、常量和一个自定义数据类型 BrowseInfo 的声明,其相关代码为:

```
Private Declare Function SHBrowseForFolder Lib "shell32" (lpbi As BrowseInfo) As Long
Private Declare Function SHGetPathFromIDList Lib " shell32" (ByVal pidList As Long, ByVal
lpBuffer As String) As Long

Private Type BrowseInfo
    hWndOwner As Long
    pIDLRoot As Long
    pszDisplayName As Long
    lpszTitle As Long
    ulFlags As Long
    lpfnCallback As Long
    lParam As Long
    iImage As Long
End Type

Const BIF_RETURNONLYFSDIRS = 1
Const MAX_PATH = 260
```

(5) 窗体中"浏览"按钮的单击事件,用于显示标准的浏览文件夹对话框,获取用户的选择,并将选择的结果添加到列表框中,其实现代码为:

```
Private Sub cmdOpen_Click( )
    Dim s As String

    '显示浏览文件夹对话框
    TextBox1. Text = ReturnFolder
    If TextBox1. Text = "\" Then Exit Sub
```

```
    '添加图形文件名称到列表框
    ListBox1.Clear
    s = Dir(TextBox1.Text & "*.dwg")
    Do While s <> ""
        ListBox1.AddItem s
        s = Dir
    Loop
    If ListBox1.ListCount <> 0 Then ListBox1.ListIndex = 0
End Sub
```

(6) 窗体中列表框的单击事件,用于显示指定文件的缩略图,其实现代码为:

```
Private Sub ListBox1_Click( )
    'ListIndex 属性为 - 1 表示没有选中
    If ListBox1.ListIndex = - 1 Then Exit Sub
    On Error Resume Next
    '显示指定的图形文件中的缩略图
    DwgThumbnail1.DwgFileName = TextBox1.Text & ListBox1.Text
End Sub
```

上面代码中使用了自定义函数 ReturnFolder,用来返回所选择的文件夹,其实现代码为:

```
'返回文件夹
Public Function ReturnFolder( ) As String
    Din udtBI As BrowseInfo
    Din lpIDList As Long
    Dim sPath As String

    '定义浏览结构
    With udtBI
        .hWndOwner = 0
        .ulFlags = BIF_RETURNONLYFSDIRS
    End With

    '浏览文件夹
    lpIDList = SHBrowseForFolder(udtBI)
    If lpIDList Then
        sPath = String(MAX_PATH, 0)
        SHGetPathFromIDList lpIDList, sPath
        sPath = Left(sPath, InStr(sPath, vbNullChar) - 1)
    End If
    If Right(sPath, 1) <> "\" Then sPath = sPath & "\"
    ReturnFolder = sPath
End Function
```

(7) 最后,在窗体中"退出"按钮的单击事件中,添加用于退出程序的代码:

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

12.3.4　实例效果

(1) 在 VBA 集成开发环境中，按下 F5 键运行程序，系统会弹出如图 12.26 所示的“预览缩略图”对话框。

(2) 单击“浏览”按钮，系统会弹出如图 12.27 所示的“浏览文件夹”对话框，用户可以从中选择一个目录，单击“确定”按钮，就可以将文件夹名称添加到“预览缩略图”对话框的“文件夹名称”文本框中。

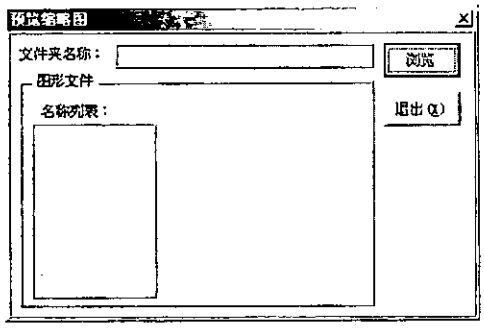


图 12.26　程序运行界面

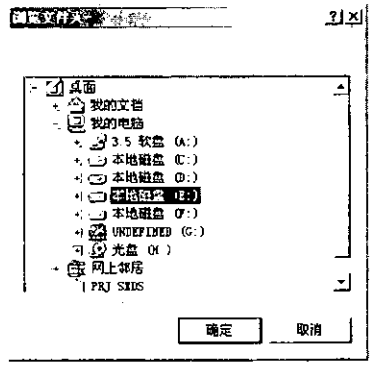


图 12.27　浏览文件夹

(3) 在“图形文件名称列表”列表框中选择要预览的文件，就可以在 DwgThumbnail 控件中显示它的缩略图，如图 12.28 所示。

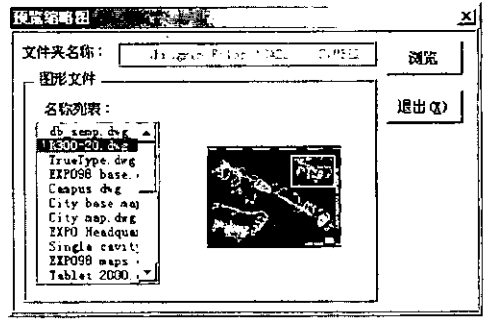


图 12.28　预览缩略图

12.3.5　小结

预览图形文件，只要设置 DwgThumbnail 控件的 DwgFileName 属性。

另外，通过分析图形文件的格式，找出文件中存放位图的位置，将位图数据读入到图像控件中，也可以达到预览的目的。

12.4　显示停靠窗口——Accont 组件

12.4.1　实例说明

AutoCAD R2002 以上版本中，虽然用户窗体可以设置成非模态，但却不能停靠四周。使用 Accont 组件，可以创建类似“对象特性”或者“设计中心”的窗口。

本实例的运行界面如图 12.29 所示，其中包含了用户控件的创建，读写实体的扩展数据。

12.4.2 编程思路

使用 Implements 关键字继承 IretrieveDockableContainer 的特性，在 SetDockContainer 事件中设置窗口停靠的特性。

使用 GetXData 和 SetXData 来读取和写入实体的扩展数据。

12.4.3 步骤

(1) 运行 VB，创建一个新的 ActiveX Control 工程，命名为 CH3_4.vbp，保存在适当的位置，进入 VB 开发环境，引用 AutoCAD Dockable Container Type Library 类型库。

(2) 选择“工程”|“添加用户控件”菜单项，向程序中添加一个用户控件，并在其中放置如图 12.30 所示的控件。

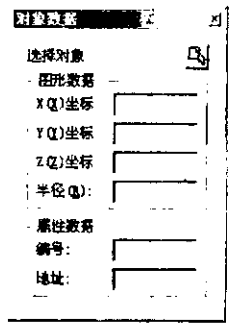


图 12.29 程序运行界面

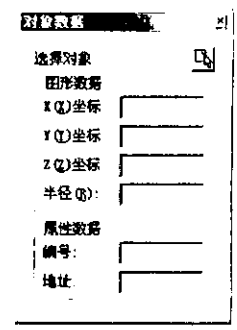


图 12.30 添加窗体控件

(3) 在控件的初始化中，用于返回 AutoCAD 的 Application 对象，其实现代码：

```
Private Sub UserControl_Initialize( )
    Set AcadApp = GetObject(, "AutoCAD.Application")
    Set EntObj = Nothing
End Sub
```

(4) 在 SetDockContainer 事件中，用于设置窗口停靠的特性，其实现代码：

```
Private Sub IRetrieveDockableContainer _ SetDockContainer ( ByVal newVal As AC _ CONT.
    IAcadDockableContainer)
    Set dockContainer = newVal
    '设置可停靠窗体的标题
    dockContainer.Caption = "对象数据"
    dockContainer.EnableDockPositions = acDockLeft + acDockRight

    '设置可停靠窗体的大小
    Dim rect(0 To 3) As Integer
    rect(0) = 0
    rect(1) = 0
```

```
    rect(2) = 180
    rect(3) = 240
    dockContainer.SetPreferredDockPosition acFloating, rect

End Sub
```

(5) 窗体中“选择”按钮的单击事件，用于选择实体，返回实体的图形数据和扩展数据，其实现代码：

```
Private Sub Command1_Click( )
    Dim pPt As Variant
    Dim xdt As Variant
    Dim xdv As Variant

    On Error GoTo ErrTrap
    '选择对象
    AcadApp.ActiveDocument.Utility.GetEntity EntObj, pPt
    '判断对象的类型是否是圆
    If StrComp(EntObj.objectname, "AcDbCircle", vbTextCompare) = 0 Then
        '返回圆心
        pPt = EntObj.Center
        Text1.Text = Round(pPt(0), 4)
        Text2.Text = Round(pPt(1), 4)
        Text3.Text = Round(pPt(2), 4)
        '返回半径
        Text4.Text = Round(EntObj.Radius, 4)
        Text5.Text = ""
        Text6.Text = ""
        '返回扩展数据
        EntObj.GetXData "电杆", xdt, xdv
        If UBound(xdt) = 2 Then
            Text5.Text = xdv(1)
            Text6.Text = xdv(2)
        End If
    End If
    Exit Sub

ErrTrap:
    '没有选中对象时，ACAD 在系统变量 ERRNO 中记录
    If AcadApp.ActiveDocument.GetVariable("ERRNO") = 7 Then
        Resume
    End If
End Sub
```

(6) 其他文本框失去焦点事件和窗体缩放事件，其实现代码：

```
Private Sub Text1_LostFocus( )
    Dim pPt(0 To 2) As Double
```

```
    On Error Resume Next
    '改变圆的位置
    pPt(0) = Text1.Text
    pPt(1) = Text2.Text
    pPt(2) = Text3.Text
    EntObj.Center = pPt
    EntObj.Update
End Sub

Private Sub Text2_LostFocus( )
    Dim pPt(0 To 2) As Double

    On Error Resume Next
    '改变圆的位置
    pPt(0) = Text1.Text
    pPt(1) = Text2.Text
    pPt(2) = Text3.Text
    EntObj.Center = pPt
    EntObj.Update
End Sub

Private Sub Text3_LostFocus( )
    Dim pPt(0 To 2) As Double

    On Error Resume Next
    '改变圆的位置
    pPt(0) = Text1.Text
    pPt(1) = Text2.Text
    pPt(2) = Text3.Text
    EntObj.Center = pPt
    EntObj.Update
End Sub

Private Sub Text4_LostFocus( )

    On Error Resume Next
    '改变圆的大小
    EntObj.Radius = Text4.Text
    EntObj.Update
End Sub

Private Sub Text5_LostFocus( )
    Dim xdt(0 To 2) As Integer
    Dim xdv(0 To 2) As Variant

    On Error Resume Next
```

```
    '设置扩展数据
    xdt(0) = 1001
    xdv(0) = "电杆"
    xdt(1) = 1000
    xdv(1) = Text5.Text
    xdt(2) = 1000
    xdv(2) = Text6.Text
    EntObj.SetXData xdt, xdv
End Sub

Private Sub Text6_LostFocus( )
    Dim xdt(0 To 2) As Integer
    Dim xdv(0 To 2) As Variant

    On Error Resume Next
    '设置扩展数据
    xdt(0) = 1001
    xdv(0) = "电杆"
    xdt(1) = 1000
    xdv(1) = Text5.Text
    xdt(2) = 1000
    xdv(2) = Text6.Text
    EntObj.SetXData xdt, xdv
End Sub

Private Sub UserControl_Resize( )
    '调整控件大小和位置
    If UserControl.Width < 2500 Then UserControl.Width = 2500
    If UserControl.Height < 3400 Then UserControl.Height = 3400

    Frame1.Width = UserControl.Width - 240
    Command1.Left = Frame1.Left + Frame1.Width - Command1.Width
    Text1.Width = Frame1.Width - 1200
    Text2.Width = Frame1.Width - 1200
    Text3.Width = Frame1.Width - 1200
    Text4.Width = Frame1.Width - 1200
    Frame2.Width = UserControl.Width - 240
    Text5.Width = Frame2.Width - 1200
    Text6.Width = Frame2.Width - 1200
End Sub
```

(7) 选择“文件”|“生成 CH3_4.ocx”菜单项，编译工程。

12.4.4 实例效果

(1) 在 AutoCAD R2000 中，选择“工具”|“加载应用程序”菜单项，弹出“加载”|“卸载应用程序”对话框，选择 accont.arx 文件加载，如图 12.31 所示。

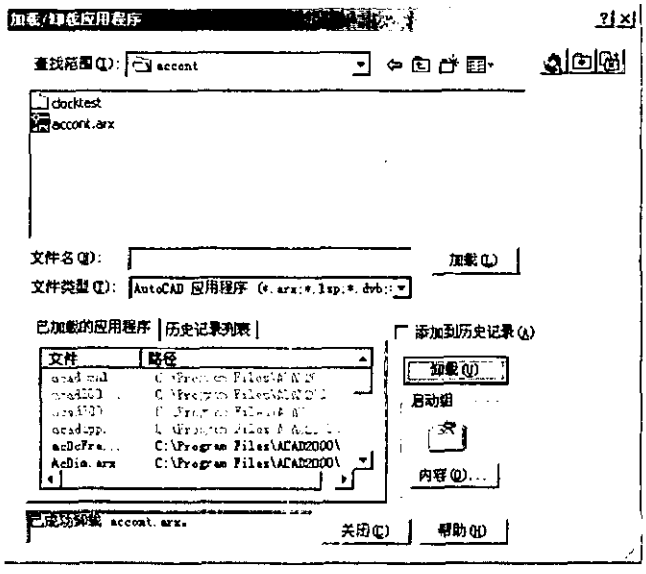


图 12.31 加载应用程序

(2) 在命令行提示下,输入 dc,提示输入 ProgID of control 时,输入 ch3_4.usercontrol1,如图 12.32 所示。

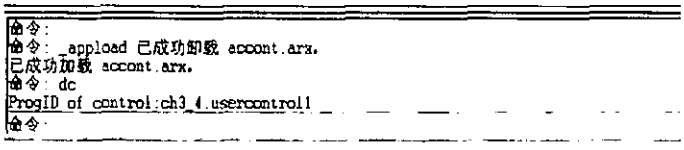


图 12.32 输入命令过程

(3) 单击窗体中的"选择"按钮,选择一个实体,就可以在窗体上显示它的数据,如图 12.33所示。

(4) 更改"地址"一栏,输入新的地址,如北京路,移走光标,再一次选择该实体,窗体上将显示修改后的数据,如图 12.34 所示。

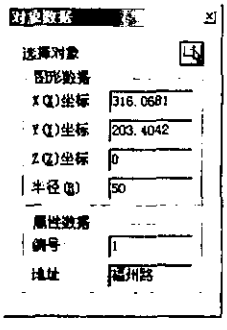


图 12.33 显示实体数据

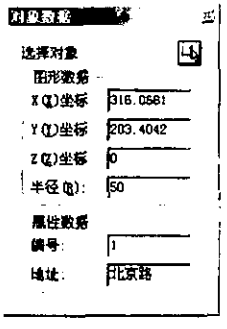


图 12.34 显示修改数据

12.4.5　小结

本节介绍了使用 Accont 组件显示停靠窗口，技术上主要是如何在 VB 中创建用户控件，然后如何在 AutoCAD 中加载。

12.5　显示幻灯片——Slide 组件

12.5.1　实例说明

在 AutoCAD 中，在命令提示窗口中执行 MSLIDE 命令，创建幻灯片，如图 12.35 所示。

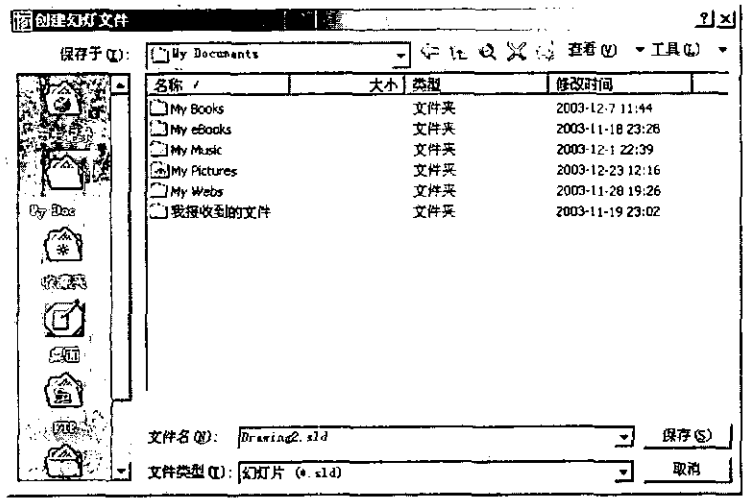


图 12.35　“创建幻灯文件”对话框

使用 AutoCAD 提供的 SLIDELIB 实用程序，可以从幻灯片(SLD)文件构造幻灯库(SLB)文件。

幻灯库文件的格式为头标(32 字节)、幻灯片索引记录(每项 36 字节)和幻灯片图像记录。

幻灯片索引记录由幻灯片名称(32 字节)和位置(4 字节)组成。

使用 Slide 组件，可以显示幻灯片或者幻灯库文件中的图像。

本实例的运行界面如图 12.36 所示，其中包含了选择幻灯片和幻灯库文件、提取幻灯库文件中的幻灯片名称和显示幻灯片。

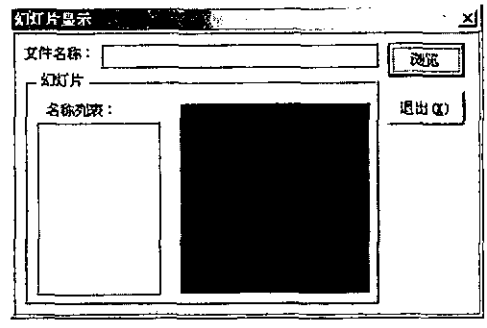


图 12.36　程序运行界面

12.5.2　编程思路

要显示幻灯片的图像，只需将 Slide 控件的 FileName 属性设置为幻灯片文件的名称，

例如 c: \r14\support\chroma. sld。

要显示幻灯库文件中的图像，除将 Slide 控件的 FileName 属性设置为幻灯库文件的名称之外，还要指定幻灯片的名称，例如 c: \r14\support\acad. slb(AILOGO)。

自定义函数 GetSLDNames 用于返回幻灯库文件中所有的幻灯片名称集合。

12.5.3 步骤

(1) 在 AutoCAD 中，打开“VBA 管理器”，创建一个新工程，保存在适当的位置，进入 VBA 集成开发环境。

(2) 选择“插入”|“用户窗体”菜单项，向程序中添加一个用户窗体，包括附加控件 AutoCAD Slide OLE Control 和 Microsoft Common Dialog Control, version 6.0(使用附加控件的方法，参见 12.5 节，本书的配套光盘中包括了 Slide 控件)，并在其中放置如图 12.37 所示的控件。

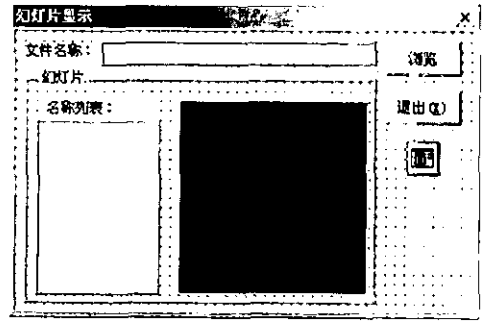


图 12.37 添加窗体控件

(3) 选择“插入”|“模块”菜单项，向程序中添加一个标准模块，在其中添加宏的启动代码：

```
Sub ShowSlide ( )
    frmMain. Show
End Sub
```

(4) 窗体中“浏览”按钮的单击事件，用于显示标准的打开对话框，获取用户的选择，并将选择的结果添加到文本框中，其实现代码：

```
Private Sub CommandButton1_Click( )
    Din s As String

    '显示打开文件对话框
    CommonDialog1. Filter = "幻灯片 ( * . sld)| * . sld|幻灯库文件 ( * . slb)| * . slb"
    CommonDialog1. FilterIndex = 1
    CommonDialog1. ShowOpen
    TextBox1. Text = CommonDialog1. FileName
    If TextBox1. Text = "" Then Exit Sub

    '添加幻灯片名称到列表框
    ListBox1. Clear
    If StrComp(Right(TextBox1. Text, 3), "SLB", vbTextCompare) = 0 Then
        ListBox1. List = GetSLDNames(TextBox1. Text) 'List 属性可直接使用数组赋值
    Else
        '返回不包含路径的文件名称
        s = TextBox1. Text
        Do While InStr(s, "\")
```

```
            s = Mid(s, InStr(s, "\") + 1)
        Loop
        s = Left(s, Len(s) - 4)
        ListBox1.AddItem s
    End If
End Sub
```

其中,GetSLDNames 是一个自定义函数,用于返回幻灯库中所有的幻灯片名称集合,它被定义在标准模块中,其实现代码:

```
Public Function GetSLDNames(ByVal FileName As String) As Variant
    Dim lngFile As Long
    Dim n As Long
    Dim s As String
    Dim sa( ) As String
    Dim SLB As SLDRECORD     '自定义数据类型

    On Error GoTo ErrTrap
    '返回一个可用的文件号
    lngFile = FreeFile( )
    '以 Binary 方式打开文件
    Open FileName For Binary As lngFile
    '定位幻灯片记录的开始位置
    Seek lngFile, 33

    Do While Not EOF(lngFile)
      '返回一条幻灯片记录
      Get lngFile, , SLB
      s = Mid(SLB.sName, 1, InStr(1, SLB.sName, Chr(0)) - 1)
      '幻灯片记录的结束使用空的名称为标志
      If s = "" Then Exit Do
      ReDim Preserve sa(n)
      sa(n) = s
      n = n + 1
    Loop
    '关闭文件
    Close lngFile
    GetSLDNames = sa
    Exit Function

ErrTrap:
      Debug.Print Err.Description
End Function
```

SLDRECORD 是一个自定义数据类型,用于保存幻灯片记录项,它被定义在标准模块

的开始部分，相关代码：

```
Private Type SLDRECORD
    sName As String * 32 '名称
    lLoc As Long '位置
End Type
```

(5) 窗体中列表框的单击事件，用于从列表框中显示指定的幻灯片，其实现代码：

```
Private Sub ListBox1_Click( )
    'ListIndex 属性为 - 1 表示没有选中
    If ListBox1.ListIndex = - 1 Then Exit Sub
    If StrComp(Right(TextBox1.Text, 3), "SLB", vbTextCompare) = 0 Then
        '显示幻灯库中指定的幻灯片
        Slide1.FileName = TextBox1.Text & "(" & ListBox1.Text & ")"
    Else
        '显示幻灯片
        Slide1.FileName = TextBox1.Text
    End If
End Sub
```

(6) 最后，在窗体中“退出”按钮的单击事件中，添加用于退出程序的代码：

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

12.5.4 实例效果

(1) 在 VBA 集成开发环境中，按下 F5 键运行程序，系统会弹出如图 12.38 所示的“幻灯片显示”对话框。

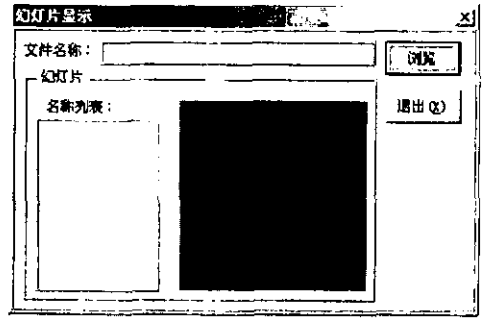


图 12.38 程序运行界面

(2) 单击“浏览”按钮，系统会弹出如图 12.39 所示的“打开”对话框，用户可以从中选择一个幻灯片或者幻灯库文件，单击“打开”按钮，就可以将文件的名称添加到文本框中。

(3) 在“幻灯片名称列表”列表框中选择要预览的幻灯片，就可以在 Slide 控件中显示，如图 12.40 所示。

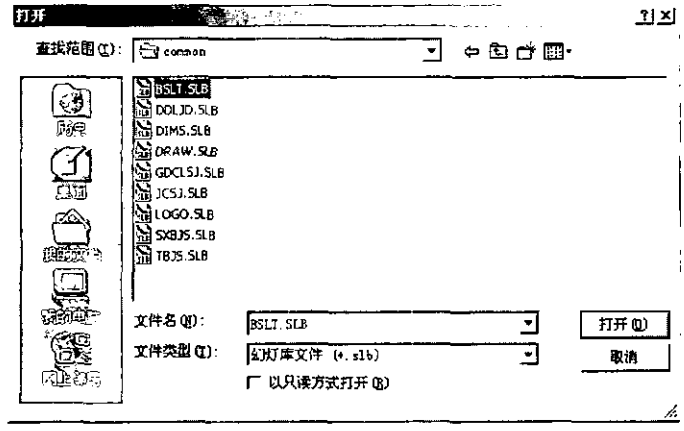


图 12.39　选择一个幻灯库文件

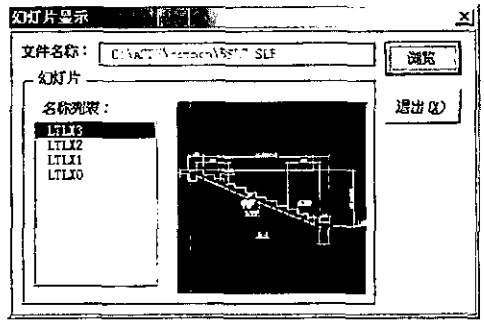


图 12.40　选择一个幻灯片

12.5.5　小结

本节介绍了使用 Slide 控件显示幻灯片文件和幻灯库文件中的幻灯片的方法，区别仅仅在于 FileName 属性的设置。

读者要通过分析幻灯库文件的格式，了解它的结构，掌握幻灯片名称列表读取的方法。

12.6　解析表达式——MSScript 组件

12.6.1　实例说明

本节使用了脚本解析工具来扩充 AutoCAD 的功能。MSScript 组件是脚本宿主部件，可以对 VBScript 的代码进行解析，而 VBScript 与 VBA 同是 VB 的子集，因此在语法上是相近的，易于使用。它可以用来解析表达式，解决工程中常用的计算功能，可以包含自定义变量和自定义函数，同时也能调用外部组件中的函数。

本实例的运行界面如图 12.41 所示，其中包含了变量的输入、函数的输入，以及输入表达式，返回计算结果。

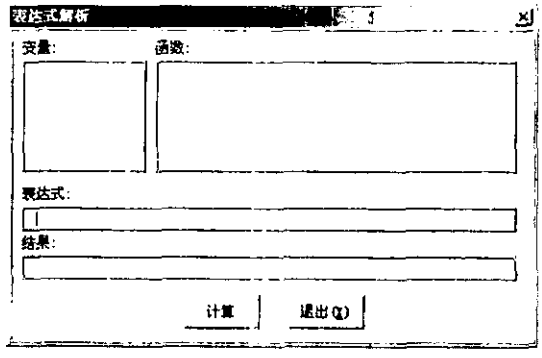


图 12.41 程序运行界面

12.6.2 编程思路

设置 MSScript 控件的 Language 属性为 VBScript。

使用 MSScript 控件的 AddCode 方法，添加变量、函数、表达式的代码。

使用 MSScript 控件的 Eval 方法，解析代码，返回结果。

12.6.3 步骤

(1) 在 AutoCAD 中，打开“VBA 管理器”，创建一个新工程，保存在适当的位置，进入 VBA 集成开发环境。选择“工具”|“引用”菜单项，在弹出的对话框中，引用 Microsoft Script Control 1.0 类型库，如图 12.42 所示。

(2) 选择“插入”“用户窗体”菜单项，向程序中添加一个用户窗体，并在其中放置如图 12.43所示的控件。

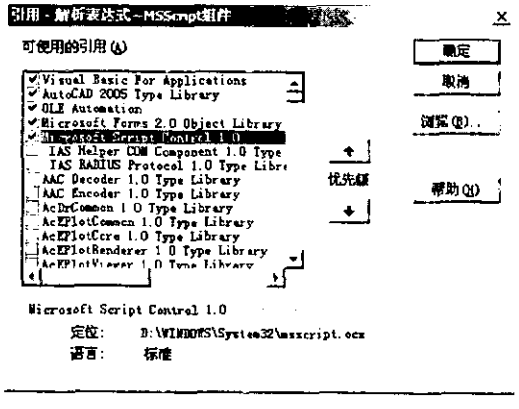


图 12.42 引用 MSScript 类型库

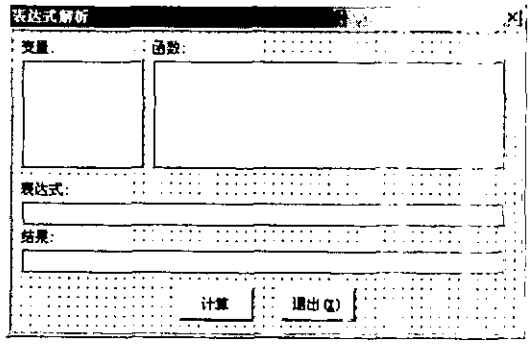


图 12.43 添加窗体控件

(3) 选择“插入”|“模块”菜单项，向程序中添加一个标准模块，其中添加宏的启动代码：

```
Sub ReplaceText( )
    frmMain.Show
End Sub
```

(4) 在窗体初始化的过程中,修改各个文本框的属性,其相关代码:

```
Private Sub UserForm_Initialize( )
    '设置多行文本
    TextBox1.MultiLine = True
    TextBox2.MultiLine = True
    '回车换行
    TextBox1.EnterKeyBehavior = True
     TextBox2.EnterKeyBehavior = True
    '垂直滚动条
    TextBox2.ScrollBars = fmScrollBarsVertical
End Sub
```

(5) 在窗体模块中添加两个自定义函数 Max 和 Min,分别用于返回三个数中的最大值或最小值,其相关代码:

```
'最大值
Function Max(number1, number2, number3)
    If number1 > = number2 And number1 > = number3 Then
        Max = number1
    ElseIf number2 > = number1 And number2 > = number3 Then
        Max = number2
    Else
        Max = number3
    End If
End Function

'最小值
Function Min(number1, number2, number3)
    If number1 < = number2 And number1 < = number3 Then
        Min = number1
    ElseIf number2 < = number1 And number2 < = number3 Then
        Min = number2
    Else
        Min = number3
    End If
End Function
```

(6) 窗体中"计算"按钮的单击事件,用于解析表达式,返回计算的结果,其实现代码:

```
Private Sub CommandButton1_Click( )

    On Error Resume Next
    '初始化控件
    Dim sc As MSScriptControl.ScriptControl
    Set sc = New MSScriptControl.ScriptControl
    '解析语言为 VBScript
    sc.Language = "VBScript"
```

```
    '添加变更字义代码
    sc.AddCode TextBox1.Text
    If Err Then
        MsgBox "变量定义错误!"
        Exit Sub
    End If
    '添加自定义函数代码
    sc.AddCode TextBox2.Text
    If Err Then
        MsgBox "函数定义错误!"
        Exit Sub
    End If
    '解析表达式
    TextBox4.Text = sc.Eval(TextBox3.Text)
    If Err Then
        MsgBox "表达式错误!"
    End If
End Sub
    If lstFile.ListCount = 0 Then
        MsgBox "请添加所要操作的图形!"
        Exit Sub
    End If
```

(7) 最后,在窗体中"退出"按钮的单击事件中,添加用于退出程序的代码:

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

12.6.4 实例效果

(1) 在 VBA 集成开发环境中,按下 F5 键运行程序,系统会弹出如图 12.44 所示的"表达式解析"对话框。

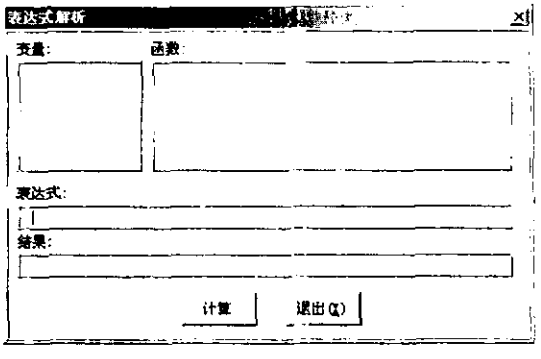


图 12.44 程序运行界面

(2) 在“变量”文本框中输入：

```
x = 1
y = 8
z = 4
```

(3) 在“函数”文本框中输入：

```
'最大值
Function Max(number1, number2, number3)
    If number1 > = number2 And number1 > = number3 Then
        Max = number1
    ElseIf number2 > = number1 And number2 > = number3 Then
        Max = number2
    Else
        Max = number3
    End If
End Function

'最小值
Function Min(number1, number2, number3)
    If number1 < = number2 And number1 < = number3 Then
        Min = number1
    ElseIf number2 < = number1 And number2 < = number3 Then
        Min = number2
    Else
        Min = number3
    End If
End Function
```

(4) 在“表达式”文本框中输入 max(x,y,z)+min(x,y,z)，如图 12.45 所示。

(5) 单击“计算”按钮，就可以根据设置的变理和函数，对表达式进行计算，并返回结果，如图 12.46 所示。

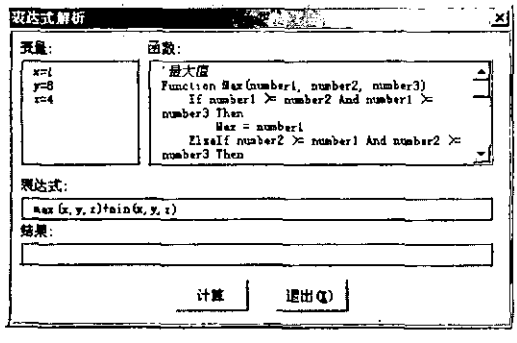


图 12.45　窗体输入结果

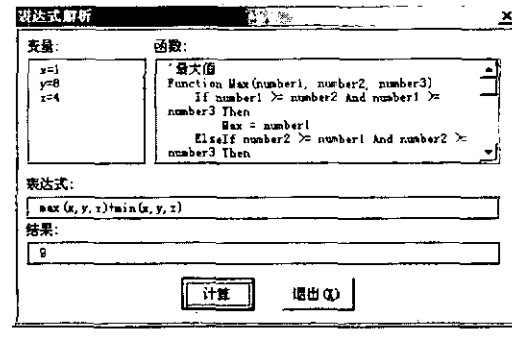


图 12.46　程序运行结果

12.6.5 小结

本节介绍了使用 MSScript 控件解析表达式的方法。AddCode 方法用于添加代码，Eval 方法用于执行代码。

12.7 创建和使用定时器

12.7.1 实例说明

在 VB 中有一个 Timer 控件，可以有规律地隔一段时间执行一次代码，这样就可以实现简单的动画，或者其他的效果。遗憾的是，VBA 中无法使用这个控件，本节实例利用一个类模块实现了类似的功能，并且介绍了将其封装成 DLL 调用的方法。

12.7.2 编程思路

VBA 中有一个函数 RaiseEvent，能够引发一个类、窗体或者文档中的模块级声明的一个事件。可以使用下面的循环结构来触发一个按指定间隔时间执行的 Timer 事件：

```
Do
    '如果定时器已经被关闭，退出循环
    If Not bEnable Then Exit Do

    If GetTickCount >= startTime + Interval Then
        startTime = GetTickCount
        RaiseEvent Timer
    End If

    '在循环过程中，用 DoEvents 函数将控制权转让给操作系统，这样可以实现后台运行定时事件的效果
    DoEvents
Loop
```

在循环开始处指定了循环的退出条件，然后判断当前的时间和上一次触发 Timer 事件的时间间隔，如果已经大于指定的间隔(Interval 属性指定)，就再次触发 Timer 事件。在循环过程中，使用 DoEvents 函数将控制权转交给操作系统，处理其他的事件。

12.7.3 步骤

1. 在类模块中使用定时器

(1) 在 AutoCAD 2005 中，打开 VBA 管理器，创建一个新工程，保存在适当的位置。进入 VBA 集成开发环境，选择“插入”|“类模块”菜单项，向工程中添加一个类模块，并在其中添加 API 函数和几个模块级变量的声明：

```
Private Declare Function GetTickCount Lib "kernel32" ( ) As Long
```

```
'bEnable 变量用于控制定时事件的状态,True 表示启动定时事件,False 表示停止定时事件
Private bEnable As Boolean '当前定时器是否可用
Private intervalTime As Long '间隔的时间(以毫秒为单位)
'Timer 事件是一个每隔固定秒数自动触发的事件,该秒数由 Interval 属性控制
Public Event Timer( )
```

(2) 在类模块被初始化时,将时间间隔设置为 1000 毫秒,但是并未启用定时器,其实现代码:

```
Private Sub Class_Initialize( )
    bEnable = False
    intervalTime = 1000
End Sub
```

(3) 选择"插入"|"过程"菜单项,系统会弹出如图 12.47 所示的"添加过程"对话框。

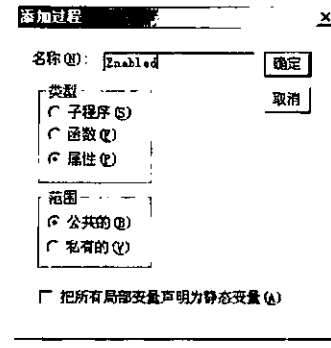


图 12.47　添加 Enabled 属性

在"名称"文本框中输入 Enabled,单击"确定"按钮,能够在代码窗口中得到 Enabled 属性的声明,在其中添加下面的语句:

```
Public Property Get Enabled( ) As Boolean
    Enabled = bEnable
End Property

Public Property Let Enabled(ByVal vNewValue As Boolean)
    bEnable = vNewValue

    '用 GetTickCount 函数返回从系统开始运行经过的毫秒数
    Dim startTime As Long
    startTime = GetTickCount

    If intervalTime < 1 Then
        MsgBox "Interval 属性的值必须是 1～32767 之间的整数", , "错误"
        Exit Property
    End If

    '用循环结构重复触发 Timer 事件
    Do
        '如果定时器已经被关闭,退出循环
        If Not bEnable Then Exit Do

        If GetTickCount > = startTime + Interval Then
            startTime = GetTickCount
            RaiseEvent Timer
        End If

        '在循环过程中,用 DoEvents 函数将控制权转让给操作系统,这样可以实现后台运行定
```

```
            时事件的效果
        DoEvents
        Loop
End Property
```

(4) 使用同样的方法,向类模块中添加一个属性 Interval,用于设置和查看定时器的间隔,其实现代码:

```
Public Property Get Interval( ) As Long
    Interval = intervalTime
End Property

Public Property Let Interval(ByVal vNewValue As Long)
    intervalTime = vNewValue
End Property
```

(5) 在类模块终止的事件中,停止定时器,其相关代码:

```
Private Sub Class_Terminate( )
    bEnable = False
End Sub
```

(6) 选择"插入"|"用户窗体"菜单项,向工程中添加一个对话框,其名称修改为 frmMain,并在其中添加两个命令按钮,名称修改为 cmdGo 和 cmdStop,如图 12.48 所示。

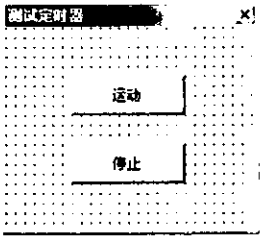


图 12.48 设计窗体

(7) 打开窗体的代码窗口,添加一个模块级变量:

```
Dim WithEvents myTimer As Timer
```

(8) 在窗体初始化的事件中,生成 Timer 类的一个实例,其相关代码:

```
Private Sub UserForm_Initialize( )
    Set myTimer = New Timer
End Sub
```

(9) 在"运动"按钮的单击事件中,启动定时器,并且将其时间间隔修改为 500 毫秒,其相关代码:

```
Private Sub cmdGo_Click( )
    myTimer.Interval = 500
    myTimer.Enabled = True
End Sub
```

(10) 在"停止"按钮的单击事件中,停止定时器,其相关代码:

```
Private Sub cmdStop_Click( )
    myTimer.Enabled = False
End Sub
```

(11) 在窗体终止的事件中,停止定时器,并且释放 myTimer 对象的存储空间,其相关代码:

```
Private Sub UserForm_Terminate( )
    myTimer.Enabled = False
    Set myTimer = Nothing
End Sub
```

(12) 在代码窗口左上角的列表中选择 myTimer 选项，在代码窗口右上角的列表中选择 Timer 事件，并在该事件中调整窗体在水平方向上的位置，其相关代码：

```
Private Sub MyTimer_Timer( )
    frmMain.Left = frmMain.Left + 5
End Sub
```

(13) 最后，双击“工程资源管理器窗”口中的 ThisDrawing 图标，打开代码窗口，在其中添加启用程序的宏：

```
Public Sub TestTimer( )
    frmMain.Show
End Sub
```

2. 将定时器封装在 DLL 中

(1) 启动 Visual Basic 6.0，在“New Project”对话框中选择“ActiveX DLL”选项，如图 12.49 所示。单击“打开”按钮，创建新的工程。

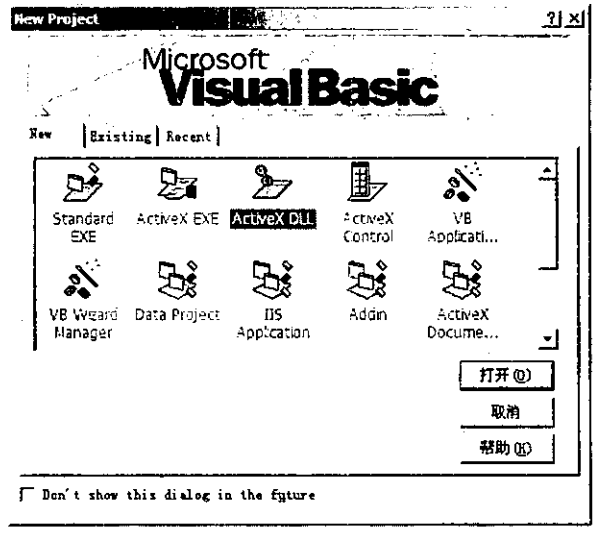


图 12.49　创建 ActiveX DLL 工程

(2) 系统会自动创建一个类模块 Class1，在属性窗口中将其名称修改为 Timer。另外，在属性窗口中将工程名称(默认情况下为 Project1)修改为 VBA_Timer。

(3) 将前面创建的类模块 Timer 中的代码全部复制到当前工程的 Timer 类模块中，将工程保存在适当的位置。

(4) 选择“File|Make VBA_Timer.dll”菜单项，系统会弹出如图 12.50 所示的“Make Project”对话框，选择一个适当的位置保存生成的 DLL 文件，单击“OK”按钮，完成 DLL 的生成。

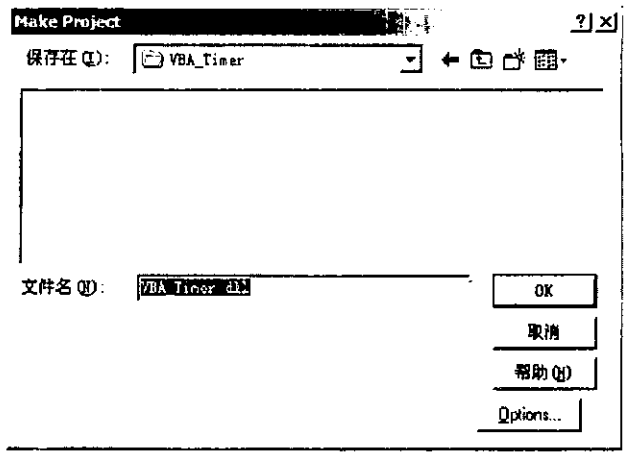


图 12.50 生成 DLL 文件

(5) 在 AutoCAD 2005 中，打开前面创建的 DVB 文件，在 VBA 集成开发环境中，选择“插入”|“用户窗体”菜单项，向工程中添加一个新的对话框，将其名称修改为 frmDllTimer，并在其中添加两个命令按钮 cmdGo 和 cmdStop。

(6) 选择“工具”|“引用”菜单项，在弹出的“引用”对话框中找到 VBA_Timer 选项，将其引用到当前工程中，单击“确定”按钮，关闭该对话框，如图 12.51 所示。

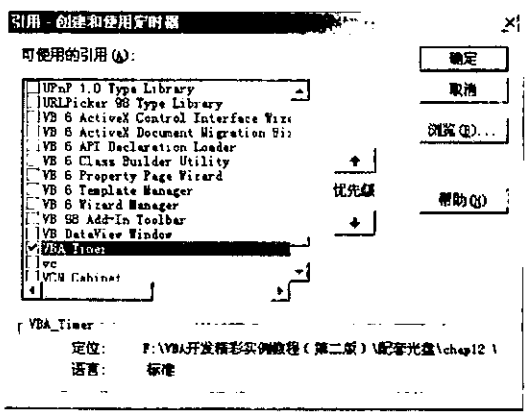


图 12.51 引用 VBA_Timer 控件

(7) 打开 frmDllTimer 对话框的代码窗口，在其中声明一个模块级变量：

```
Dim WithEvents myTimer As VBA_Timer.Timer
```

(8) 对窗体中的各事件进行响应，代码与 frmMain 窗体基本相同，这里仅列出相关的代码：

```
Private Sub cmdGo_Click( )
    myTimer.Interval = 500
    myTimer.Enabled = True
End Sub
```

```
Private Sub cmdStop_Click( )
    myTimer.Enabled = False
End Sub

Private Sub myTimer_Timer( )
    frmDllTimer.Left = frmDllTimer.Left + 5
End Sub

Private Sub UserForm_Initialize( )
    '用 VBA_Timer 来区分 Timer 类模块
    Set myTimer = New VBA_Timer.Timer
End Sub

Private Sub UserForm_Terminate( )
    myTimer.Enabled = False
    Set myTimer = Nothing
End Sub
```

(9) 最后，在 ThisDrawing 模块代码窗口中添加一个宏 TestVBA_Timer，用于显示 frmDllTimer 对话框，其相关代码：

```
Public Sub TestVBA_Timer( )
    frmDllTimer.Show
End Sub
```

12.7.4　实例效果

在 VBA 集成开发环境中，执行 TestVBA_Timer 或者 TestTimer 宏，都能得到同样的效果：单击“运动”按钮，窗体每隔 0.5 秒就向右移动一段距离；单击“停止”按钮，窗体会停止运动。

由于使用 DLL 要通过 COM 技术传递数据，因此在性能上要比类模块低一些，frmDllTimer 窗体开始运动之前有一个短时间的停顿。但是从代码的安全性来说，使用 DLL 显然要安全得多。

12.7.5　小结

VBA 和 VB 所能使用的控件、函数不完全一样，但是可以通过在 VB 中创建 DLL，然后调用 DLL 在 VBA 中实现对应的功能。学习本节的实例之后，读者需要掌握类模块的创建以及类模块向外部 DLL 转换的方法。

第 13 章　与其他应用程序和 Windows API 交互

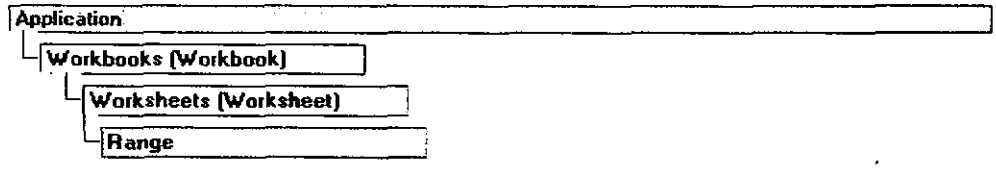
自从 AutoCAD 2000 版本开始提供 VBA 开发工具以来，关于 VBA 和 Excel 的数据交换就一直是一个热点问题。本章提供了 Excel 表格向 AutoCAD 的转换、使用 Excel 数据绘图、转换 AutoCAD 数据到 Word 和 Excel 中的几个实例，相信已经能够详尽地解决 AutoCAD VBA 和 Excel 合作编程的问题。

在使用 Windows API 方面，提供了去掉对话框的关闭按钮和添加快捷菜单的实例。当然 Windows API 对 VBA 应用程序功能的扩展是非常灵活和强大的，例如第 15 章修改 AutoCAD 应用程序的图标和标题栏仍然是利用 Windows API 来实现的。

13.1　转换 Excel 表格到 AutoCAD 中

13.1.1　实例说明

Excel 对象模型中的主要对象：



运行 Microsoft Excel 应用程序，就生成了一个 Application(应用程序)对象的实例，再选择“文件”|“新建”菜单项，创建一个新的文档，就在 Workbooks 集合中添加了一个 Workbook(工作簿)对象，同时系统默认的在 Worksheets 集合中创建了三个 Worksheet(工作表)对象。

实际应用中，一个 Worksheet 对象对应于一张表格，由若干个 Range(单元格)对象组成，而几个单元格又可以组成更大的 Range(单元格区域)对象，因而 Range 对象可以表示 Row(行)、Column(列)、Cell(单元格)等对象。

通过 Excel 中的数据生成 AutoCAD 中的表格，必须了解如何与 Excel 建立通信，如何读取 Excel 数据的问题，这些都可以通过 Excel 对象模型所提供的对象及其方法、属性来实现。

本实例的运行界面如图 13.1 所示，其中包含了转换后表格位置的指定，表格转换的区域和方向，以及转换后表格边框的对象类型和表格文字的对象类型的选择。

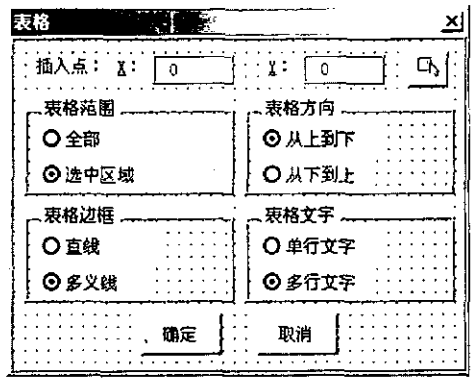


图 13.1　程序运行界面

13.1.2　编程思路

要转换一张表格，首先明确程序所要实现的以下几种主要功能：

- 表格的转换区域，通常是使用区域(UsedRange)，即包含数据的单元格区域，以及选中区域(Selection)，即用户当前选定的单元格区域。
- 表格的转换方向，主要是针对垂直方向而言，有从上到下和从下到上两种方式，水平方向都是按从左到右的方式进行处理。机械行业中使用的明细表即为从下到上的表格。
- 表格的插入点以及表格的范围，即它包含的行数和列数。
- 表格的行高，Excel 中的行高(RowHeight)是以磅为单位，而 AutoCAD 中一般是以毫米为单位，1 磅等于 0.35 毫米。
- 表格的列宽，Excel 中的列宽(ColumnWidth)是以字符为单位，也可以返回以磅为单位的宽度(Width)，而 AutoCAD 中一般是以毫米为单位。
- 表格边框转换后生成的对象类型，主要考虑直线(Line)和轻量多段线(LWPolyline)两种。
- 表格边框的线型、线宽和颜色等，依据实际的应用情况来决定是否需要转换。
- 表格文字转换后生成的对象类型，有单行文字(Text)和多行文字(MText)两种，如果是转换到块中，还可以使用属性(Attribute)。
- 表格文字的字体、字号和颜色等，Excel 中文字的字号以磅为单位，而 AutoCAD 中的字高一般是以毫米为单位，但以此转换之后的文字会稍稍偏大，可以依据实际的应用情况决定如何转换。

注意，为了转换方便，可以人为设置转换关系，如 Excel 中的 1 磅相当于 AutoCAD 中的 1.0 毫米。另外，基于速度的考虑，边框的线条宜用直线，文字宜用单行文字，而且转换的内容越多，转换所需要的时间也就越长。

13.1.3　步骤

(1) 在 VBA 管理器中，创建一个新的工程，保存在适当的位置，进入 VBA 集成开发环境。

(2) 选择“插入”|“用户窗体”菜单项,向程序中添加一个用户窗体,并在其中放置如图 13.2 所示的控件。

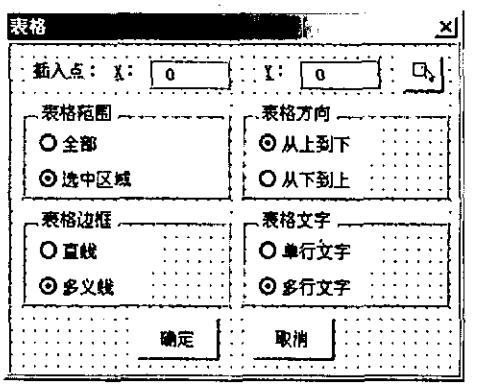


图 13.2 向窗体中添加控件

(3) 选择“插入”|“模块”菜单项,向程序中添加一个标准模块,并在其中添加下面的代码:

```
Option Explicit

Public Sub ConvertExcel( )
    UserForm1.Show
End Subb
```

上面的代码用于显示对话框,运行程序。

(4) 在对话框中双击“插入点”按钮,进入代码窗口,在该按钮的 Click 事件中添加下面的代码:

```
Private Sub cmdPickPt_Click( )
    Dim iPt As Variant

    On Error GoTo ErrTrap
    '隐藏窗体,注意与用户交互时必须先隐藏窗体,然后等待用户操作,最后显示窗体
    Me.Hide
    '返回选择或者输入的点
    iPt = ThisDrawing.Utility.GetPoint(,"指定插入点: ")
    TextBox1.Text = iPt(0)
    TextBox2.Text = iPt(1)
    iPt(2) = 0
    '显示窗体
    Me.Show
    Exit Sub

ErrTrap:
    '防止出错时窗体不显示
    Me.Show
```

```
    On Error GoTo 0
End Sub
```

在这段代码中，获取点的输入使用 Utility 对象的 GetPoint 方法，它返回 Variant 类型的值。操作过程中，还要在适当的时候隐藏和显示窗体，不然 AutoCAD 无法获得控制权，与用户交互，程序就会出错而退出。

(5) 在对话框中双击“确定”按钮，进入代码窗口，在该按钮的 Click 事件中添加下面的代码：

```
Private Sub cmdOK_Click( )
    Dim iPt(0 To 2) As Double
    Dim xlApp As Object
    Dim xlSheet As Object

    '表格的插入点
    iPt(0) = TextBox1.Text
    iPt(1) = TextBox2.Text
    iPt(2) = 0

    '发生错误时跳到下一个语句继续执行
    On Error Resume Next
    '连接 Excel 应用程序
    Set xlApp = GetObject(,"Excel.Application")
    If Err.Number <> 0 Then
        MsgBox "Excel 应用程序没有运行!"
        Exit Sub
    End If

    '返回当前活动的工作表
    Set xlSheet = xlApp.ActiveSheet

    '表格的转换方向
    If OptionButton3.Value = True Then
        DirectionScale = 1 '从上到下
    Else
        DirectionScale = -1 '从下到上
    End If

    Dim r As Object
    '表格的转换区域
    If OptionButton1.Value = True Then '使用区域
        '遍历要转换的单元格区域
        For Each r In xlSheet.UsedRange
            '转换表格的边框
            AddTableLine ThisDrawing.Blocks(" * Model_Space"),iPt,r
            '转换表格的文字
```

```
            AddTableText ThisDrawing.Blocks(" * Model_Space"),iPt,r
        Next
    Else '选中区域
        '遍历要转换的单元格区域
        For Each r In xlApp.Selection
            '转换表格的边框
            AddTableLine ThisDrawing.Blocks(" * Model_Space"),iPt,r
            '转换表格的文字
            AddTableText ThisDrawing.Blocks(" * Model_Space"),iPt,r
        Next
    End If

    '更新
    ThisDrawing.Regen acActiveViewport

    '释放对象
    Set xlSheet = Nothing
    Set xlApp = Nothing

    Unload Me
End Sub
```

上面代码中使用了自定义函数 AddTableLine，用来完成表格边框的绘制，自定义函数 AddTableText 用来完成表格文字的绘制，自定义函数 LineWidth 返回边框线条的粗细，自定义函数 LineColor 返回边框线条的颜色，自定义函数 TextColor 返回文字的颜色，其实现代码：

```
'绘制单元格中的边框
Sub AddTableLine(ByRef BlockObj As AcadBlock,ByVal iPt As Variant,ByVal xlRange As Range)
    '记录单元格的位置和尺寸
    Dim rl As Double
    Dim rt As Double
    Dim rw As Double
    Dim rh As Double

    Dim sPt(0 To 2) As Double
    Dim ePt(0 To 2) As Double
    Dim pPt(0 To 3) As Double
    Dim LineObj As AcadLine
    Dim LWPLineObj As Object

    Dim i As Integer

    '单元格没有边框不做处理
    If xlRange.Borders(xlEdgeLeft).LineStyle = xlNone _
        And xlRange.Borders(xlEdgeBottom).LineStyle = xlNone _
```

```
        And xlRange.Borders(xlEdgeRight).LineStyle = xlNone _
        And xlRange.Borders(xlEdgeTop).LineStyle = xlNone Then Exit Sub

    If ColumnWidthUnitType = 1 Then
        '字符单位时的列宽
        For i = TableRange.Column To xlRange.Column-1
            rl = rl + TableRange.Columns(i).ColumnWidth * ColumnWidthScale
        Next
    ElseIf ColumnWidthUnitType = 2 Then
        '以磅为单位时的列宽
        rl = (xlRange.Left-TableRange.Left) * ColumnWidthScale
    End If
    rt = (xlRange.top-TableRange.top) * RowHeightScale
    If ColumnWidthUnitType = 1 Then
        rw = xlRange.ColumnWidth * ColumnWidthScale
    ElseIf ColumnWidthUnitType = 2 Then
        rw = xlRange.Width * ColumnWidthScale
    End If
    rh = xlRange.Height * RowHeightScale

    '左边框,只有第一列才转换,避免重复,因为单元格的右边框为右方单元格的左边框
    If xlRange.Borders(xlEdgeLeft).LineStyle <> xlNone And xlRange.Column = TableRange.
Column Then
        If OptionButton5.Value = True Then
            '创建直线
            sPt(0) = iPt(0) + rl: sPt(1) = iPt(1)-rt * DirectionScale
            ePt(0) = iPt(0) + rl: ePt(1) = iPt(1)-(rt + rh) * DirectionScale
            Set LineObj = BlockObj.AddLine(sPt,ePt)
            LineObj.color = LineColor(xlRange.Borders(xlEdgeLeft))
        Else
            '创建多段线
            pPt(0) = iPt(0) + rl: pPt(1) = iPt(1)-rt * DirectionScale
            pPt(2) = iPt(0) + rl: pPt(3) = iPt(1)-(rt + rh) * DirectionScale
            Set LWPLineObj = BlockObj.AddLightWeightPolyline(pPt)
            LWPLineObj.ConstantWidth = LineWidth(xlRange.Borders(xlEdgeLeft))
            LWPLineObj.color = LineColor(xlRange.Borders(xlEdgeLeft))
        End If
    End If

    '下边框,对于合并单元格的内部不转换
    If xlRange.Borders(xlEdgeBottom).LineStyle <> xlNone And (xlRange.Row = xlRange.
MergeArea.Row + xlRange.MergeArea.Rows.Count-1) Then
        If OptionButton5.Value = True Then
            '创建直线
            sPt(0) = iPt(0) + rl: sPt(1) = iPt(1)-(rt + rh) * DirectionScale
```

```
            ePt(0) = iPt(0) + rl + rw: ePt(1) = iPt(1)-(rt + rh) * DirectionScale
            Set LineObj = BlockObj.AddLine(sPt,ePt)
            LineObj.color = LineColor(xlRange.Borders(xlEdgeLeft))
        Else
            '创建多段线
            pPt(0) = iPt(0) + rl: pPt(1) = iPt(1)-(rt + rh) * DirectionScale
            pPt(2) = iPt(0) + rl + rw: pPt(3) = iPt(1)-(rt + rh) * DirectionScale
            Set LWPLineObj = BlockObj.AddLightWeightPolyline(pPt)
            LWPLineObj.ConstantWidth = LineWidth(xlRange.Borders(xlEdgeBottom))
            LWPLineObj.color = LineColor(xlRange.Borders(xlEdgeBottom))
        End If
    End If

    '右边框,对于合并单元格的内部不转换
    If xlRange.Borders(xlEdgeRight).LineStyle <> xlNone And (xlRange.Column > = xlRange.
        MergeArea.Column + xlRange.MergeArea.Columns.Count-1) Then
        If OptionButton5.Value = True Then
            '创建直线
            sPt(0) = iPt(0) + rl + rw: sPt(1) = iPt(1)-rt * DirectionScale
            ePt(0) = iPt(0) + rl + rw: ePt(1) = iPt(1)-(rt + rh) * DirectionScale
            Set LineObj = BlockObj.AddLine(sPt,ePt)
            LineObj.color = LineColor(xlRange.Borders(xlEdgeLeft))
        Else
            '创建多段线
            pPt(0) = iPt(0) + rl + rw: pPt(1) = iPt(1)-rt * DirectionScale
            pPt(2) = iPt(0) + rl + rw: pPt(3) = iPt(1)-(rt + rh) * DirectionScale
            Set LWPLineObj = BlockObj.AddLightWeightPolyline(pPt)
            LWPLineObj.ConstantWidth = LineWidth(xlRange.Borders(xlEdgeRight))
            LWPLineObj.color = LineColor(xlRange.Borders(xlEdgeRight))
        End If
    End If

    '上边框,只有第一行才转换,避免重复,因为单元格的下边框为下方单元格的上边框
    If xlRange.Borders(xlEdgeTop).LineStyle <> xlNone And xlRange.Row = TableRange.Row Then
        If OptionButton5.Value = True Then
            '创建直线
            sPt(0) = iPt(0) + rl: sPt(1) = iPt(1)-rt * DirectionScale
            ePt(0) = iPt(0) + rl + rw: ePt(1) = iPt(1)-rt * DirectionScale
            Set LineObj = BlockObj.AddLine(sPt,ePt)
            LineObj.color = LineColor(xlRange.Borders(xlEdgeLeft))
        Else
            '创建多段线
            pPt(0) = iPt(0) + rl: pPt(1) = iPt(1)-rt * DirectionScale
            pPt(2) = iPt(0) + rl + rw: pPt(3) = iPt(1)-rt * DirectionScale
            Set LWPLineObj = BlockObj.AddLightWeightPolyline(pPt)
```

```
            LWPLineObj.ConstantWidth = LineWidth(xlRange.Borders(xlEdgeTop))
            LWPLineObj.color = LineColor(xlRange.Borders(xlEdgeTop))
        End If
    End If

    Set LineObj = Nothing
    Set LWPLineObj = Nothing
End Sub

'绘制单元格中的文字
Sub AddTableText(ByRef BlockObj As AcadBlock,ByVal iPt As Variant,ByVal xlRange As Range)
    '记录单元格的位置和尺寸
    Dim rl As Double
    Dim rt As Double
    Dim rw As Double
    Dim rh As Double

    Dim tiPt(0 To 2) As Double
    Dim TextObj As AcadText
    Dim MTextObj As AcadMText
    Dim tPt As Variant

    Dim i As Integer

    '单元格没有文字不做处理
    If xlRange.Text = "" Then Exit Sub

    If ColumnWidthUnitType = 1 Then
        '字符单位时的列宽
        For i = TableRange.Column To xlRange.Column-1
            rl = rl + TableRange.Columns(i).ColumnWidth * ColumnWidthScale
        Next
    ElseIf ColumnWidthUnitType = 2 Then
        '以磅为单位时的列宽
        rl = (xlRange.Left-TableRange.Left) * ColumnWidthScale
    End If
    rt = (xlRange.top-TableRange.top) * RowHeightScale
    If ColumnWidthUnitType = 1 Then
        rw = xlRange.ColumnWidth * ColumnWidthScale
    ElseIf ColumnWidthUnitType = 2 Then
        rw = xlRange.Width * ColumnWidthScale
    End If
    rh = xlRange.MergeArea.Height * RowHeightScale

    tiPt(0) = iPt(0) + rl: tiPt(1) = iPt(1)-rt * DirectionScale: tiPt(2) = 0
```

```
If OptionButton7.Value = True Then
    '创建单行文字
        Set TextObj = BlockObj.AddText(xlRange.Text, tiPt, xlRange.Font.Size *
            TextHeightScale)
    TextObj.color = TextColor(xlRange.Font)
Else
    '创建多行文字
    Set MTextObj = BlockObj.AddMText(tiPt,rw,xlRange.Text)
    MTextObj.Height = xlRange.Font.Size * TextHeightScale
    MTextObj.color = TextColor(xlRange.Font)
End If

tiPt(0) = tiPt(0) + rw/2: tiPt(1) = tiPt(1)-rh/2 * DirectionScale: tiPt(2) = 0
 If xlRange.VerticalAlignment = xlTop And (xlRange.HorizontalAlignment = xlLeft Or
    xlRange.HorizontalAlignment = xlGeneral) Then
    '左上对齐
    If OptionButton7.Value = True Then
        TextObj.Alignment = acAlignmentTopLeft
    Else
        MTextObj.AttachmentPoint = acAttachmentPointTopLeft
    End If
    tPt = ThisDrawing.Utility.PolarPoint(tiPt,3.1415926,rw/2)
    tPt = ThisDrawing.Utility.PolarPoint(tPt,1.5707963,rh/2)
ElseIf xlRange.VerticalAlignment = xlTop And xlRange.HorizontalAlignment = xlCenter Then
    '中上对齐
    If OptionButton7.Value = True Then
        TextObj.Alignment = acAlignmentTopCenter
    Else
        MTextObj.AttachmentPoint = acAttachmentPointTopCenter
    End If
    tPt = ThisDrawing.Utility.PolarPoint(tiPt,1.5707963,rh/2)
ElseIf xlRange.VerticalAlignment = xlTop And xlRange.HorizontalAlignment = xlRight Then
    '右上对齐
    If OptionButton7.Value = True Then
        TextObj.Alignment = acAlignmentTopRight
    Else
        MTextObj.AttachmentPoint = acAttachmentPointTopRight
    End If
    tPt = ThisDrawing.Utility.PolarPoint(tiPt,0,rw/2)
    tPt = ThisDrawing.Utility.PolarPoint(tPt,1.5707963,rh/2)
ElseIf xlRange.VerticalAlignment = xlCenter And (xlRange.HorizontalAlignment = xlLeft
        Or xlRange.HorizontalAlignment = xlGeneral) Then
    '左中对齐
    If OptionButton7.Value = True Then
        TextObj.Alignment = acAlignmentMiddleLeft
```

```
    Else
        MTextObj.AttachmentPoint = acAttachmentPointMiddleLeft
    End If
    tPt = ThisDrawing.Utility.PolarPoint(tiPt,3.1415926,rw/2)
  ElseIf xlRange.VerticalAlignment = xlCenter And xlRange.HorizontalAlignment =
    xlCenter Then
    '正中对齐
    If OptionButton7.Value = True Then
        TextObj.Alignment = acAlignmentMiddleCenter
    Else
        MTextObj.AttachmentPoint = acAttachmentPointMiddleCenter
    End If
    tPt = tiPt
  ElseIf xlRange.VerticalAlignment = xlCenter And xlRange.HorizontalAlignment =
    xlRight Then
    '右中对齐
    If OptionButton7.Value = True Then
        TextObj.Alignment = acAlignmentMiddleRight
    Else
        MTextObj.AttachmentPoint = acAttachmentPointMiddleRight
    End If
    tPt = ThisDrawing.Utility.PolarPoint(tiPt,0,rw/2)
ElseIf xlRange.VerticalAlignment = xlBottom And (xlRange.HorizontalAlignment = xlLeft _
        Or xlRange.HorizontalAlignment = xlGeneral) Then
    '左下对齐
    If OptionButton7.Value = True Then
        TextObj.Alignment = acAlignmentBottomLeft
    Else
        MTextObj.AttachmentPoint = acAttachmentPointBottomLeft
    End If
    tPt = ThisDrawing.Utility.PolarPoint(tiPt,3.1415926,rw/2)
    tPt = ThisDrawing.Utility.PolarPoint(tPt,-1.5707963,rh/2)
  ElseIf xlRange.VerticalAlignment = xlBottom And xlRange.HorizontalAlignment =
    xlCenter Then
    '中下对齐
    If OptionButton7.Value = True Then
        TextObj.Alignment = acAlignmentBottomCenter
    Else
        MTextObj.AttachmentPoint = acAttachmentPointBottomCenter
    End If
    tPt = ThisDrawing.Utility.PolarPoint(tiPt,-1.5707963,rh/2)
  ElseIf xlRange.VerticalAlignment = xlBottom And xlRange.HorizontalAlignment =
    xlRight Then
    '右下对齐
    If OptionButton7.Value = True Then
```

```
            TextObj.Alignment = acAlignmentBottomRight
        Else
            MTextObj.AttachmentPoint = acAttachmentPointBottomRight
        End If
        tPt = ThisDrawing.Utility.PolarPoint(tiPt,0,rw/2)
        tPt = ThisDrawing.Utility.PolarPoint(tPt,-1.5707963,rh/2)
    End If
    If OptionButton7.Value = True Then
        TextObj.TextAlignmentPoint = tPt
    Else
        MTextObj.InsertionPoint = tPt
    End If

    Set TextObj = Nothing
    Set MTextObj = Nothing
End Sub

'边框粗细
Function LineWidth(ByVal xlBorder As Border) As Double
    Select Case xlBorder.Weight
        Case xlThin
            LineWidth = 0
        Case xlMedium
            LineWidth = 0.35
        Case xlThick
            LineWidth = 0.7
        Case Else
            LineWidth = 0
    End Select
End Function

'边框颜色,处理的颜色不全,请自己添加
Function LineColor(ByVal xlBorder As Border) As Integer
    Select Case xlBorder.ColorIndex
        Case xlAutomatic
            LineColor = acByLayer
        Case 2
            LineColor = acWhite
        Case 3
            LineColor = acRed
        Case 5
            LineColor = acBlue
        Case 6
            LineColor = acYellow
         Case 7
```

```
            LineColor = acMagenta
        Case 10
            LineColor = acGreen
        Case 14
            LineColor = acCyan
        Case Else
            LineColor = acByLayer
    End Select
End Function

'文字颜色,处理的颜色不全,请自己添加
Function TextColor(ByVal xlFont As Excel.Font) As Integer
    Select Case xlFont.ColorIndex
        Case xlAutomatic
            TextColor = acByLayer
        Case 2
            TextColor = acWhite
        Case 3
            TextColor = acRed
        Case 5
            TextColor = acBlue
        Case 6
            TextColor = acYellow
        Case 7
            TextColor = acMagenta
        Case 10
            TextColor = acGreen
        Case 14
            TextColor = acCyan
        Case Else
            TextColor = acByLayer
    End Select
End Function
```

(6) 最后,在“取消”按钮的 Click 事件中添加结束程序的代码:

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

13.1.4　实例效果

(1) 在 VBA 集成开发环境中,按下 F5 键,在 AutoCAD 应用程序窗口中执行当前的程序,系统会弹出如图 13.3 所示的对话框。

(2) 在对话框中单击“插入点”按钮，对话框隐藏，AutoCAD 的命令行上会提示“指定插入点：”，拾取完成后，对话框重新显示，在文本框中给出了拾取的插入点结果，如图 13.4 所示。

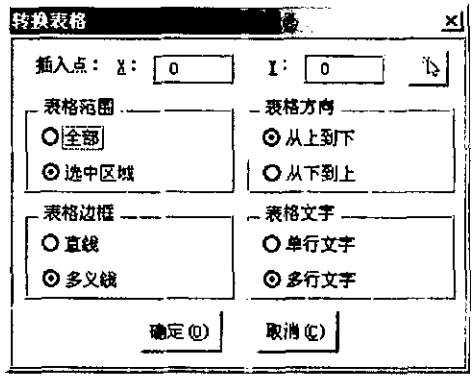


图 13.3 程序运行窗体

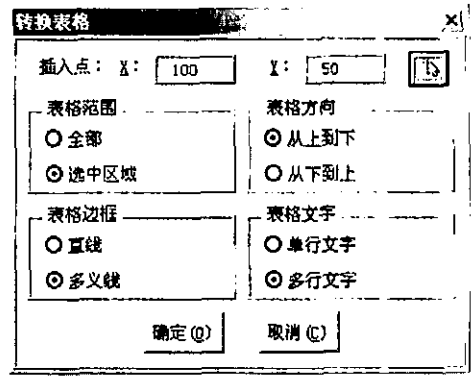


图 13.4 指定插入点结果的显示

(3) 分别选中“表格范围”、“表格方向”、“表格边框”、“表格文字”中的单选按钮，设置转换的参数。

(4) 单击“确定”按钮，运行转换 Excel 数据生成表格的代码，单击“取消”按钮，结束程序的运行，得到如图 13.5 所示的结果。

序号	代号	名称	数量	材料	单件	总计	备注
1	GB70-76	螺钉M6X12	4	A3			
2	GB119-76	销3X10	1	45			
3	MDE-51-1-101	滑阀	1	40Cr			
4	MDE-51-1-102	拨叉	1	HT15-33			
5	MDE-51-1-103	盖板	1	HT15-33			
6		垫片	1	纸板			无图
7	MDE-51-1-10	阀体	1	HT20-40			
8	HG4-333-76	O型密封圈25X24	1	耐油橡胶			
9	GB71-76	螺钉M6X10	1	A3			
10	MDE-55-1-105	手把	1	35			
11	MDE-51-1-106	胶木球	1	胶木			
12	GB308-77	钢球7.5	1	GCr6			
13	GB2089-80	弹簧1X7X15	1	碳素弹簧钢丝I组			
14	GB1099-79	键3X16	1	45			

图 13.5 数据导出结果

13.1.5 小结

表格是由直线和文字构成的，可以独立转换，也可以制作成块，再插入到图纸中，形成一个整体，缺点是编辑不方便。

13.2　使用 Excel 数据绘图

13.2.1　实例说明

很多情况下，用户习惯将采集的数据放置在 Excel 中，然后直接通过 Excel 的数据在 AutoCAD 中绘图，以便将 Excel 中的数据“导入”到 AutoCAD 中。

本节所完成的实例非常简单，从如图 13.6 所示的 Excel 中读取数据，以这些数据作为直线的起点和终点坐标，在 AutoCAD 中创建一条直线。

13.2.2　编程思路

Excel 中最重要的几个对象分别是 Application、Workbook、Worksheet 和 Range，其相互之间的层次关系如图 13.7 所示。

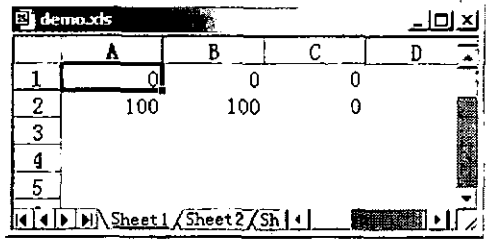


图 13.6　要使用的 Excel 数据

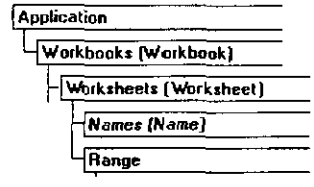


图 13.7　对象的层次关系

几个主要对象的含义如下：

- Application：代表 Excel 应用程序对象，是一个多用途的对象，包括应用程序范围内的设置和选项。
- Workbook：可以存储数据的工作薄，Workbooks 是所有工作薄的集合。一个工作薄相当于一个 Excel 文档。
- Worksheet：处理数据的原始文档，由排列成行和列的单元格组成，位于工作薄当中，如图 13.8 所示。Worksheets 是工作薄中所有工作表的集合。

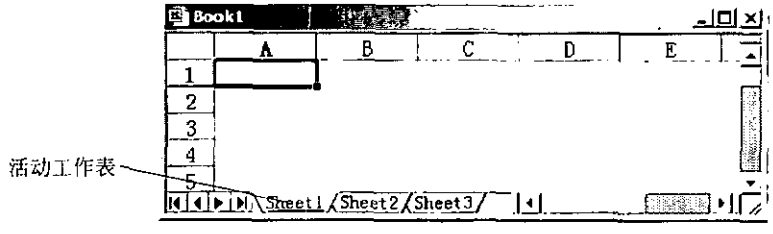


图 13.8　工作表的概念

- Range：工作表内选择的单元格。

Excel VBA 有很多东西值得一看，但是本书不再对此做更深入的探讨。下面，就将在

创建工程的过程中了解从 Excel 中读取数据的方法。

13.2.3 步骤

(1) 打开 Excel 应用程序，在其自动创建的新工作表中，输入如图 13.6 所示的数据，并且保存在适当的位置(文件名为 demo.xls)。

(2) 在 AutoCAD 2005 中，打开 VBA 管理器，创建一个新工程，将其保存在 demo.xls 所在的文件夹中。

(3) 进入 VBA 集成开发环境，双击“工程资源管理器”窗口中的 ThisDrawing 图标，打开其代码窗口。选择“工具”|“引用”菜单项，打开如图 13.9 所示的对话框，选中 Microsoft Excel 9.0 Object Library(Excel 类型库，其版本与计算机上安装的 Office 版本有关，9.0 是 Office 2000 对应的版本号)、Microsoft Visual Basic for Applications Extensibility 5.3 (VBA IDE 类型库)两项。

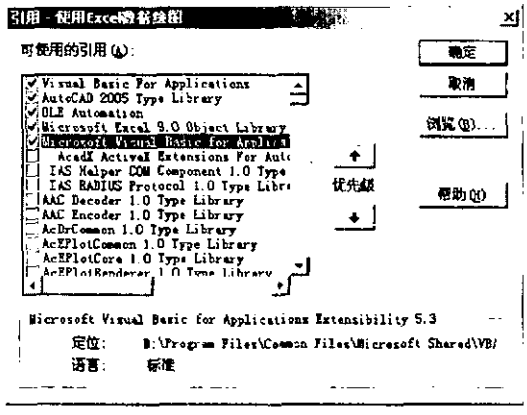


图 13.9 应用 Excel 类型库和 VBA IDE 类型库

引用类型库，实际上是向编译器(解释器)表示本程序要使用某个已经注册的组件(一般来说可能是动态链接库、ActiveX EXE 等组件)。引用类型库之后，就可以在对象浏览器(按下 F2 键即可显示)中观察类型库中的对象、方法和属性。

(4) 在 ThisDrawing 模块的代码窗口中，添加一个宏 UseExcelData，用于根据 Excel 中的数据创建一条直线，其实现代码为：

```
Public Sub UseExcelData( )
    Dim excelApp As Excel.Application
    Dim excelSheet As Excel.Worksheet

    '获得当前工程的路径
    Dim strFile As String
    strFile = ThisDrawing.Application.VBE.ActiveVBProject.FileName

    '运行 Excel 应用程序
    Set excelApp = CreateObject("Excel.Application")
    excelApp.Visible = True
```

```
    '打开指定的 Excel 文件,获得指定的页
    excelApp.Workbooks.Open Left$(strFile,Len(strFile)-15) & "demo.xls"
    Set excelSheet = excelApp.ActiveWorkbook.Sheets("Sheet1")

    '使用指定页的数据绘图
    Dim lineObj As AcadLine
    Dim startPoint(0 To 2) As Double
    Dim endPoint(0 To 2) As Double

    startPoint(0) = excelSheet.Cells(1,1).Value
    startPoint(1) = excelSheet.Cells(1,2).Value
    startPoint(2) = excelSheet.Cells(1,3).Value
    endPoint(0) = excelSheet.Cells(2,1).Value
    endPoint(1) = excelSheet.Cells(2,2).Value
    endPoint(2) = excelSheet.Cells(2,3).Value

    Set lineObj = ThisDrawing.ModelSpace.AddLine(startPoint,endPoint)
    ZoomAll

    '退出 Excel 应用程序
    excelApp.Quit
End Sub
```

ThisDrawing. Application. VBE. ActiveVBPproject. FileName 用于获得当前 DVB 文件的位置和名称,这是为了实现 Excel 文件位置的无关性(否则附带光盘中的这个程序必然不能在某些读者的计算机上直接运行)。只要 Excel 文件和当前的 DVB 文件在同一目录下,程序就可以正确无误地运行。

CreateObject 函数创建并返回一个对 ActiveX 对象的引用,在这里用于创建 Excel 应用程序的一个进程。关于该函数的详细说明,可以参见本节小结的内容。

Workbooks 集合的 Open 方法用于打开一个 Excel 文档,其参数为 Excel 文档的位置和名称。由于这里将 Excel 文档和当前的 DVB 文件保存在同一目录下,就可以根据 DVB 文件的路径来定位 Excel 文档。Left$(strFile,Len(strFile)-15)用于获得 DVB 文件所在的目录,再加上 demo. xls 就是 Excel 文档的具体位置。

由 DVB 文件的具体位置获得其所在目录,必须从具体位置中删去 DVB 文件的名称(例如,这里保存的名称是"使用 Excel 绘图. dvb")。为了判断 DVB 文件名称的长度,编写了下面的宏:

```
Public Sub GetLengthOfString( )
    MsgBox Len("使用 Excel 数据绘图.dvb")
End Sub
```

运行此宏所得的结果是 15,因此这里使用的是 Left$(strFile,Len(strFile)-15)。读者可以用自己的文件名称命名"使用 Excel 绘图. dvb",根据结果来修改 Left$(strFile,Len(strFile)-15)。

由工作薄获得工作表可以使用工作表的 Sheets 属性，并在返回的 Worksheets 集合中直接获得所需要的工作表 Sheet1。

获得某个单元格的内容可以使用下面的语句：

```
excelSheet.Cells(1,1).Value
```

最终，使用 excelApp.Quit 退出 Excel 应用程序。

13.2.4 实例效果

在 VBA 集成开发环境中，执行 UseExcelData 宏，能够得到如图 13.10 所示的结果。

13.2.5 小结

Excel VBA 的编程并不困难，如果需要获得更为详细的资料，可以参考相关的图书。

在本节的程序中，CreateObject("Excel.Application")用于创建 Excel 应用程序的一个进程，其中的 Excel.Application 是一个类标志符(CLSID)，为什么是 Excel.Application 而不是其他形式？这个标志符实际上是应用程序的创建者指定的，因此要获得一个应用程序的类标志符，必须参考一些相关的资料。

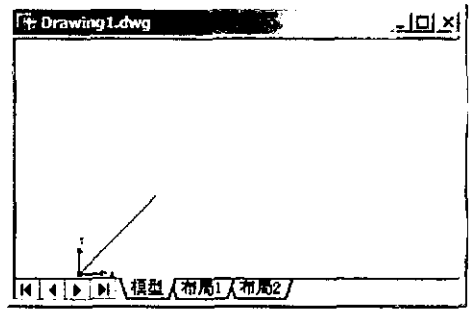


图 13.10 绘制直线的结果

如同下面的代码来加载 ObjectDBX 的版本：

```
If Left(Version,2) = "15" Then
    Set objDbx = CreateObject("ObjectDBX.AxDbDocument.1")
ElseIf Left(Version,2) = "16" Then
    Set objDbx = CreateObject("ObjectDBX.AxDbDocument.16")
End If
```

很多用户不理解为什么 AutoCAD 2000/2002 使用 ObjectDBX.AxDbDocument.1，而 AutoCAD 2004 则使用 ObjectDBX.AxDbDocument.16。这就是 AutoDesk 设计时的问题了，它本来想从 1 开始递增版本号，但是又想让这个版本号与 AutoCAD 的版本号一致，因此就出现了这个问题。

实际上，这些 CLSID 都保存在注册表中，可以在 Windows“开始”菜单中选择“运行”菜单项，在弹出的对话框中输入 regedit，并按下 Enter 键，打开注册表窗口，搜索 ObjectDBX.AxDbDocument.16，可以得到如图 13.11 所示的结果。

从注册表窗口中可以看出，想直接从注册表窗口中查找需要的 CLSID(ProgID 从某种意义上说是它的同义词)是非常困难的，最好的办法就是查看官方的资料。

上面的知识涉及到一种名为 COM 的技术。作为一般的编程者，不用关心 COM 的内部实现，只需掌握调用的方法即可。如果需要进一步研究这些内容，可以阅读 COM 方面的书籍。

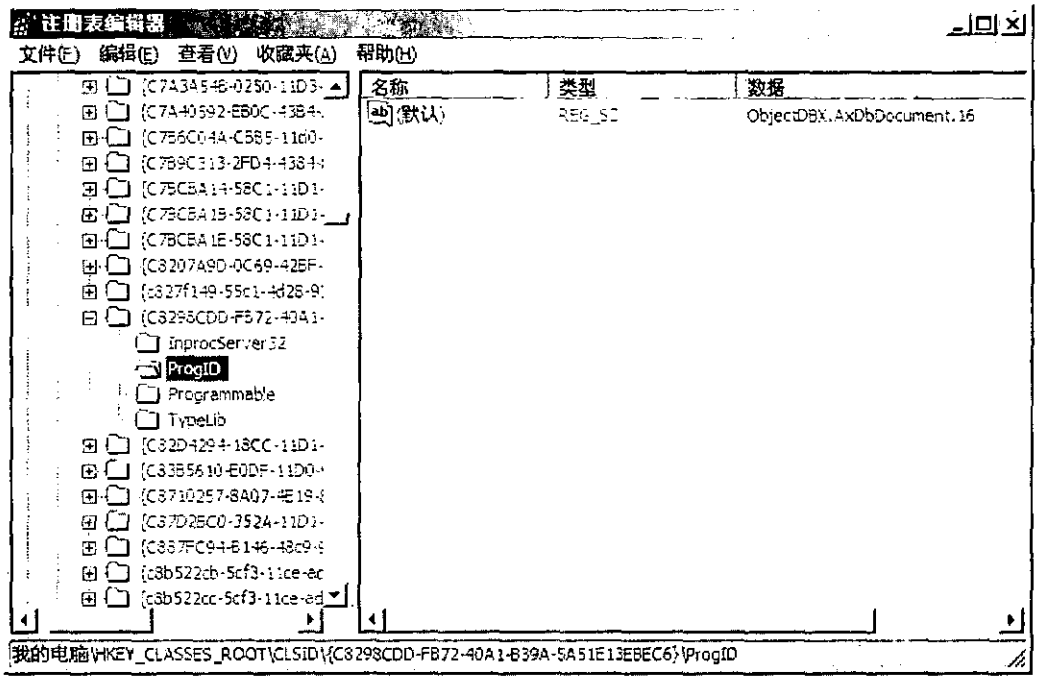
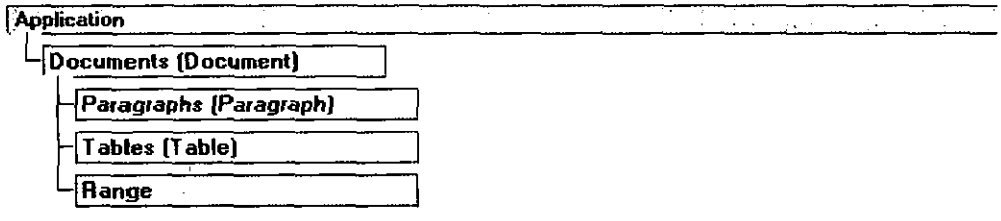


图 13.11　注册表中保存的 CLSID

13.3　转换 AutoCAD 数据到 Word 和 Excel 中

13.3.1　实例说明

Word 对象模型中的主要对象：



运行 Microsoft Word 应用程序，就生成了一个 Application(应用程序)对象的实例，选择“文件”|“新建”菜单项，创建一个新的文档，就在 Documents 集合中添加了一个 Document(文档)对象。

实际应用中，一个 Document 对象可以看成一篇文章，由若干个 Range(文字区域)对象和 Table(表格)对象组成，而 Range 对象可以表示文章中的段落(Paragraph)、句子(Sentence)、单词(Word)、字符(Character)等，也可以表示表格中的行(Row)、列(Column)和单元格(Cell)。

使用 AutoCAD 中的数据来生成 Word 中的文字和表格，必须了解如何与 Word 建立通信，如何写入数据的问题，这些都可以通过 Word 对象模型所提供的对象及其方法、属性来

实现。

本实例的运行界面如图 13.12 所示，其中包含了明细表表行块的选择、块中属性名称的列表，以及明细表转换后的调整和排序。

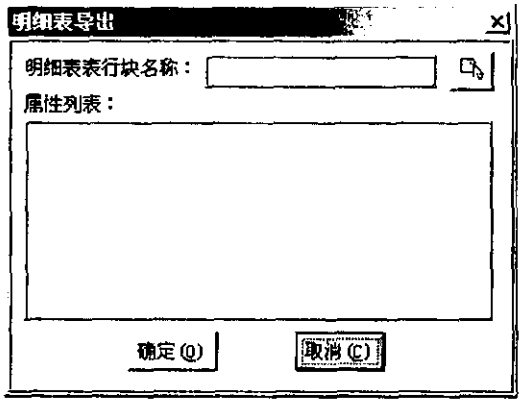


图 13.12 程序运行界面

13.3.2 编程思路

机械行业中的明细表是由表头和表行构成的，表头相当于表格的列标题，而表行则是表格的数据。要在 Word 中生成一张包含明细表数据的表格，首先明确本程序所要实现的以下几种主要功能：

- 表格的范围，即它包含的行数和列数，行数就是表行块的个数，列数是块中属性的个数。
- 表格的转换方向，明细表是按从下到上的方式生成，并有折行的现象，使用 Word 的排序功能就不需要考虑这些。
- 表格的行高、列宽，使用 Word 的自动调整功能就不需要考虑这些。
- 其他，按系统默认的方式转换。

13.3.3 步骤

(1) 在 AutoCAD 中，打开“VBA 管理器”，创建一个新工程，保存在适当的位置，进入 VBA 集成开发环境。

(2) 选择“插入”|“用户窗体”菜单项，向程序中添加一个用户窗体，并在其中放置如图 13.13所示的控件。

(3) 选择“插入”|“模块”菜单项，向程序中添加一个标准模块，并在其中添加下面的代码：

```
Public Sub OutputWord( )
    UserForm1.Show
End Sub
```

(4) 在对话框中双击“选择”按钮，进入代码窗口，在该按钮的 Click 事件中添加下面的代码：

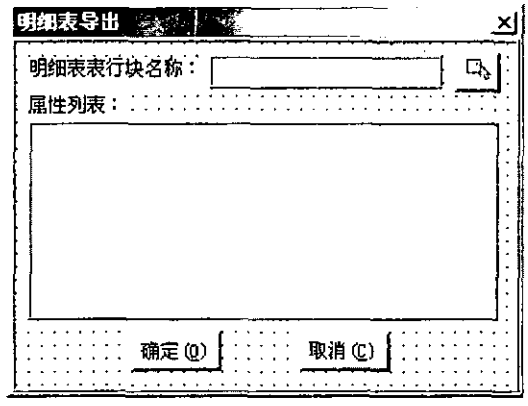


图 13.13　向窗体中添加控件

```
Private Sub cmdPickPt_Click( )
    Dim EntObj As AcadEntity
    Dim pPt As Variant

    On Error GoTo ErrTrap
    TextBox1.Text = ""
    ListBox1.Clear

    '隐藏窗体,注意与用户交互时必须先隐藏窗体,然后等待用户操作,最后显示窗体
    Me.Hide
    '返回选择的对象
    ThisDrawing.Utility.GetEntity EntObj,pPt,"选择明细表表行块："
    '选中的对象不是块
    If StrComp(EntObj.ObjectName,"AcDbBlockReference",vbTextCompare) <> 0 Then
        Me.Show
        Exit Sub
    End If

    '返回选择的块的名称
    TextBox1.Text = EntObj.Name

    Dim AttRefObjs As Variant
    Dim i As Integer
    '判断块是否包含属性
    If EntObj.HasAttributes Then
        '返回属性
        AttRefObjs = EntObj.GetAttributes
        '遍历属性
        For i = 0 To UBound(AttRefObjs)
            ListBox1.AddItem AttRefObjs(i).TagString
            ListBox1.Selected(ListBox1.ListCount-1) = True
```

```
        Next
    End If

    '释放对象
    Set EntObj = Nothing

    '显示窗体
    Me.Show
    Exit Sub

ErrTrap:
    '未选中对象时发生错误,错误号为 7,按右键取消选择操作
    If ThisDrawing.GetVariable("ERRNO") = 7 Then Resume
    Me.Show
    On Error GoTo 0
End Sub
```

(5) 窗体中“确定”按钮的单击事件,根据选中的属性,导出数据到 Word 中,其实现代码:

```
Private Sub cmdOK_Click( )
    Dim SSetObj As Object
    Dim fType(0 To 1) As Integer
    Dim fData(0 To 1) As Variant

    On Error Resume Next
    '创建选择集
    Set SSetObj = ThisDrawing.SelectionSets("MXB")
    If Err.Number <> 0 Then
        Err.Clear
        Set SSetObj = ThisDrawing.SelectionSets.Add("MXB")
    End If
    SSetObj.Clear

    On Error GoTo ErrTrap
    '创建过滤机制
    fType(0) = 0: fData(0) = "INSERT" '块
    fType(1) = 2: fData(1) = TextBox1.Text '块名

    SSetObj.Select acSelectionSetAll,,,fType,fData
    If SSetObj.Count = 0 Then Exit Sub

    '导出属性到 Word
    OutputToWord SSetObj,ListBox1
    '导出属性到 Excel
    OutputToExcel SSetObj,ListBox1
```

```
    '删除选择集,释放内存
    SSetObj.Delete
    Set SSetObj = Nothing
    Exit Sub

ErrTrap:
    On Error GoTo 0
End Sub
```

上面代码中使用了自定义函数 OutputToWord,用来导出数据到 Word 中；自定义函数 OutputToExcel,用来导出数据到 Excel 中。提供两种不同的转换方法,在于区别它们的不同之处,其实现代码为：

```
'导出到 Word 中
Public Sub OutputToWord(ByVal SSetObj As AcadSelectionSet,ByVal LBObj As ListBox)
    Dim wdApp As Word.Application
    Dim wdDoc As Word.Document
    Dim wdTable As Word.Table

    Dim EntObj As AcadEntity
    Dim AttRefObjs As Variant
    Dim n As Integer
    Dim i As Integer
    Dim j As Integer

    On Error Resume Next
    '连接 Word
    Set wdApp = GetObject(,"Word.Application")
    If Err Then
        Err.Clear
        Set wdApp = CreateObject("Word.Application")
        If Err Then
            Err.Clear
            MsgBox "无法启动 Word,请检查是否正确安装!"
            Exit Sub
        End If
    End If
    wdApp.Visible = True

    On Error GoTo ErrTrap
    '返回新创建的文档
    Set wdDoc = wdApp.Documents.Add
    '返回在段落一之后新创建的表格
    Set wdTable = wdDoc.Tables.Add(wdDoc.Paragraphs(1).Range,1,LBObj.ListCount)
```

```
    n = 0
    '遍历选择集
    For Each EntObj In SSetObj
        '增加行
        wdTable.Rows.Add
        '返回属性数据
        AttRefObjs = EntObj.GetAttributes
        n = n + 1
        For i = 0 To UBound(AttRefObjs)
            For j = 0 To LBObj.ListCount-1
                If AttRefObjs(i).TagString = LBObj.List(j) And LBObj.Selected(j) = True Then
                    If n = 1 Then
                        '首行,标签做为表格的列标题
                        wdTable.Cell(n,j + 1).Range.Text = AttRefObjs(i).TagString
                        wdTable.Cell(n + 1,j + 1).Range.Text = AttRefObjs(i).TextString
                    Else
                        wdTable.Cell(n + 1,j + 1).Range.Text = AttRefObjs(i).TextString
                    End If
                End If
            Next
        Next
    Next

    '删除表格中的空列
    For i = LBObj.ListCount-1 To 0 Step -1
        If wdTable.Cell(1,i + 1).Range.Text = vbCr + Chr(7) Then
            wdTable.Columns(i + 1).Delete
        End If
    Next

    '按序号排序
    wdTable.Sort True,"列 1"
    '自动调整列宽
    wdTable.AutoFitBehavior 1

    '释放 Word 对象
    Set wdTable = Nothing
    Set wdDoc = Nothing
    Set wdApp = Nothing
    Exit Sub

ErrTrap:
    On Error GoTo 0
End Sub
```

```
'导出到 Excel 中
Public Sub OutputToExcel(ByVal SSetObj As AcadSelectionSet,ByVal LBObj As ListBox)
    Dim xlApp As Excel.Application
    Dim xlBook As Excel.Workbook
    Dim xlSheet As Excel.Worksheet

    Dim EntObj As AcadEntity
    Dim AttRefObjs As Variant
    Dim n As Integer
    Dim i As Integer
    Dim j As Integer

    On Error Resume Next
    '连接 Excel
    Set xlApp = GetObject(,"Excel.Application")
    If Err Then
        Err.Clear
        Set xlApp = CreateObject("Excel.Application")
        If Err Then
            Err.Clear
            MsgBox "无法启动 Word,请检查是否正确安装!"
            Exit Sub
        End If
    End If
    xlApp.Visible = True

    On Error GoTo ErrTrap
    '返回新创建的工作簿
    Set xlBook = xlApp.Workbooks.Add
    '返回新增加的工作表,并移动到最后一个
    Set xlSheet = xlBook.Worksheets.Add
    xlSheet.Move,xlBook.Worksheets(xlBook.Worksheets.Count)

    n = 0
    '遍历选择集
    For Each EntObj In SSetObj
        '返回属性数据
        AttRefObjs = EntObj.GetAttributes
        n = n + 1
        For i = 0 To UBound(AttRefObjs)
            For j = 0 To LBObj.ListCount-1
                If AttRefObjs(i).TagString = LBObj.List(j) And LBObj.Selected(j) = True Then
                    If n = 1 Then
                        '首行,标签做为表格的列标题
                        xlSheet.Cells(n,j + 1).Value = AttRefObjs(i).TagString
```

```
                    xlSheet.Cells(n + 1,j + 1).Value = AttRefObjs(i).TextString
                Else
                    xlSheet.Cells(n + 1,j + 1).Value = AttRefObjs(i).TextString
                End If
            End If
        Next
    Next
Next

'删除表格中的空列
For i = LBObj.ListCount-1 To 0 Step -1
    If xlSheet.Cells(1,i + 1).Value = "" Then
        xlSheet.Columns(i + 1).Delete
    End If
Next

'按序号排序
xlSheet.UsedRange.Sort Key1: = Range("A2"),Header: = xlYes
'自动调整列宽
xlSheet.Columns.AutoFit

'释放 Exccel 对象
Set xlSheet = Nothing
Set xlBook = Nothing
Set xlApp = Nothing
Exit Sub

ErrTrap:
    On Error GoTo 0
End Sub
```

(6) 最后，在窗体中“取消”按钮的单击事件中，添加用于退出程序的代码：

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

13.3.4 实例效果

(1) 在 AutoCAD 2005 中，打开配套光盘中“\chap13 与其他应用程序和 Windows API 交互\转换 AutoCAD 数据到 Word 和 Excel 中－测试资料\”目录中的 demo.dwg 文件。在 VBA 集成开发环境中，按下 F5 键运行程序，在 AutoCAD 应用程序窗口中执行当前的程序，系统会弹出如图 13.14 所示的“明细表导出”对话框。

(2) 在对话框中单击“选择”按钮，对话框隐藏，AutoCAD 的命令行上会提示“选择明细表表行块：”，拾取完成后，对话框重新显示，在文字框中给出了拾取的明细表表行块的名称结果，如图 13.15 所示。

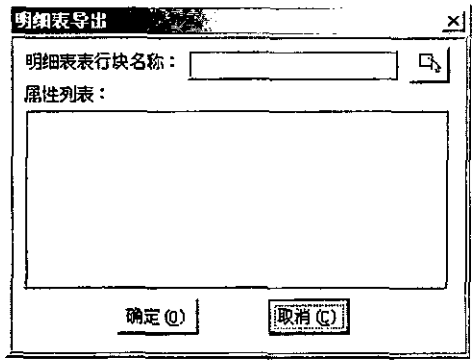


图 13.14　程序运行窗体

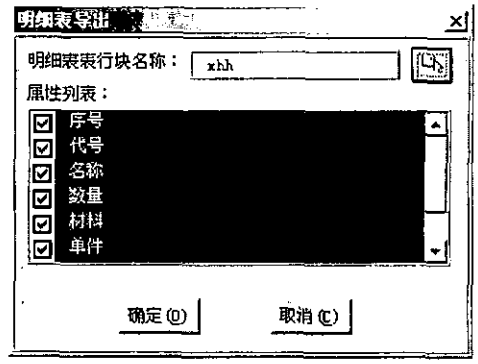


图 13.15　属性字段列表

(3) 选择“属性列表”列表框中要导出的属性字段，可以全部或者部分导出。

(4) 单击“确定”按钮，运行导出的代码，然后单击“取消”按钮，结束程序的运行，分别在 Word 和 Excel 得到输出的数据，如图 13.16 和图 13.17 所示。

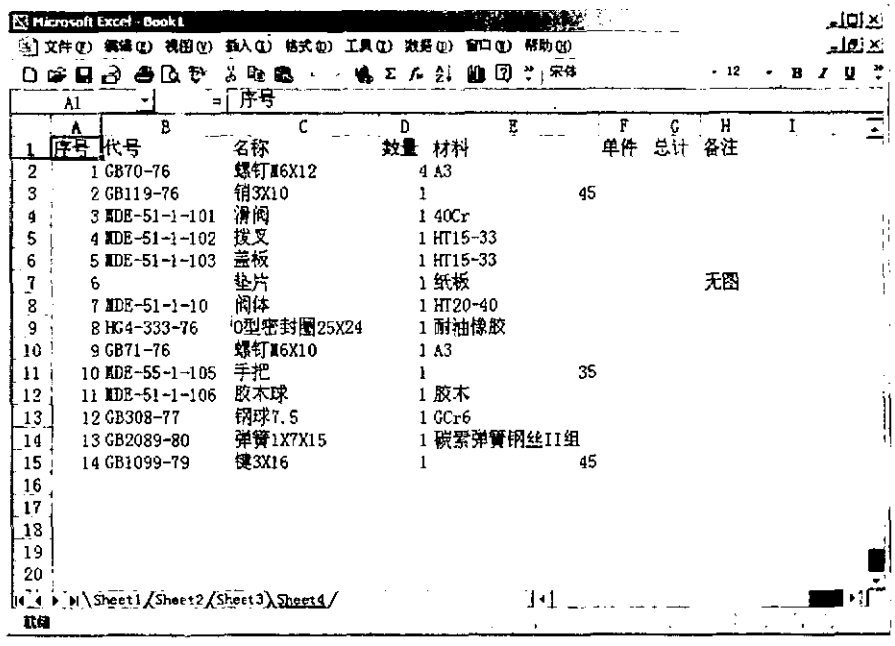


序号	代号	名称	数量	材料	单件	总计	备注
1	GB70-76	螺钉 M6X12	4	A3			
2	GB119-76	销 3X10	1	45			
3	MDE-51-1-101	滑阀	1	40Cr			
4	MDE-51-1-102	拔叉	1	HT15-33			
5	MDE-51-1-103	盖板	1	HT15-33			
6		垫片	1	纸板			无图
7	MDE-51-1-10	阀体	1	HT20-40			
8	HG4-333-76	O 型密封圈 25X24	1	耐油橡胶			
9	GB71-76	螺钉 M6X10	1	A3			
10	MDE-55-1-105	手把	1	35			
11	MDE-51-1-106	胶木球	1	胶木			
12	GB308-77	钢球 7.5	1	GCr6			
13	GB2089-80	弹簧 1X7X15	1	碳素弹簧钢丝 II 组			
14	GB1099-79	键 3X16	1	45			

图 13.16　Word 中导出数据的结果

13.3.5　小结

明细表是由表头和表行构成的，对于表行由块组成的方式，可以先转换数据，然后在 Word 或者 Excel 中利用它的排序功能，根据序号列进行排序，而行高和列宽则利用自动调整功能来缩放。



Microsoft Excel - Book1

文件(F) 编辑(E) 视图(V) 插入(I) 格式(O) 工具(T) 数据(D) 窗口(W) 帮助(H)

A1 = 序号

	A	B	C	D	E	F	G	H	I
1	序号	代号	名称	数量	材料	单件	总计	备注	
2	1	GB70-76	螺钉M6X12	4	A3				
3	2	GB119-76	销5X10	1	45				
4	3	MDE-51-1-101	滑阀	1	40Cr				
5	4	MDE-51-1-102	拨叉	1	HT15-33				
6	5	MDE-51-1-103	盖板	1	HT15-33				
7	6		垫片	1	纸板			无图	
8	7	MDE-51-1-10	阀体	1	HT20-40				
9	8	HG4-333-76	O型密封圈25X24	1	耐油橡胶				
10	9	GB71-76	螺钉M6X10	1	A3				
11	10	MDE-55-1-105	手把	1	35				
12	11	MDE-51-1-106	胶木球	1	胶木				
13	12	GB308-77	钢球7.5	1	GCr6				
14	13	GB2089-80	弹簧1X7X15	1	碳素弹簧钢丝II组				
15	14	GB1099-79	键3X16	1	45				
16									
17									
18									
19									
20									

Sheet1 / Sheet2 / Sheet3 / Sheet4 /

就绪

图 13.17 Excel 中导出数据的结果

13.4 去掉关闭按钮(X)和添加快捷菜单

13.4.1 实例说明

通过 API 函数对用户窗体的菜单进行设置，也可以实现窗体的一些特殊效果。

本实例的运行界面如图 13.18 所示，其中包含了去除 X 按钮、添加快捷菜单和回调函数的使用。

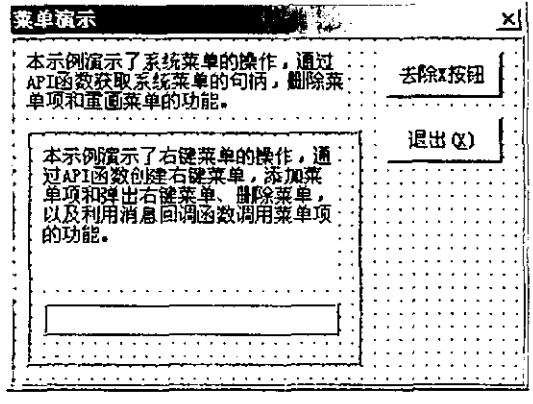


图 13.18 程序运行界面

13.4.2　编程思路

使用 GetActiveWindow 函数获取运行时的用户窗体口句柄，使用控件的 SetFocus 方法设置焦点，之后利用 GetFocus 函数可以返回控件的句柄。

使用 GetSystemMenu 函数返回用户窗体的系统菜单，使用 RemoveMenu 函数可以移除其中的某一菜单项。

使用 CreatePopupMenu 函数创建弹出菜单，使用 AppendMenu 函数向弹出菜单中添加菜单项。

使用 GetCursorPos 函数得到光标的位置，TrackPopupMenu 函数显示弹出菜单。

13.4.3　步骤

(1) 在 AutoCAD 中，打开“VBA 管理器”，创建一个新工程，保存在适当的位置，进入 VBA 集成开发环境。

(2) 选择“插入”|“用户窗体”菜单项，向程序中添加一个用户窗体，并在其中放置如图 13.19 所示的控件。

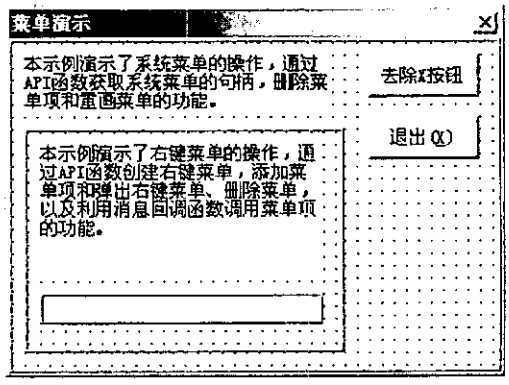


图 13.19　添加窗体控件

(3) 选择“插入”|“模块”菜单项，向程序中添加一个标准模块，其中添加 API 函数和常量的声明，以及宏的启动代码：

```
Public Declare Function GetActiveWindow Lib "user32" ( ) As Long
Public Declare Function GetFocus Lib "user32" ( ) As Long
Public UserFormHwnd As Long
Public TextBoxHwnd As Long

Public Declare Function GetSystemMenu Lib "user32" (ByVal hwnd As Long,ByVal bRevert As Long)
    As Long
Private Declare Function GetMenuItemCount Lib "user32" (ByVal hMenu As Long) As Long
Private Declare Function RemoveMenu Lib "user32" (ByVal hMenu As Long,ByVal nPosition As Long,
    ByVal wFlags As Long) As Long
Private Declare Function DrawMenuBar Lib "user32" (ByVal hwnd As Long) As Long
Private Const MF_REMOVE = &H1000&
```

```
Private Const MF_BYPOSITION = &H400&

Private Declare Function CreatePopupMenu Lib "user32" ( ) As Long
Private Declare Function AppendMenu Lib "user32" Alias "AppendMenuA" (ByVal hMenu As Long,ByVal
    wFlags As Long,ByVal wIDNewItem As Long,ByVal lpNewItem As Any) As Long
Public Declare Function DestroyMenu Lib "user32" (ByVal hMenu As Long) As Long
Public Declare Function TrackPopupMenu Lib "user32" (ByVal hMenu As Long,ByVal wFlags As Long,
    ByVal x As Long,ByVal y As Long,ByVal nReserved As Long,ByVal hwnd As Long,ByVal lprc As
    Any) As Long
Public Declare Function GetCursorPos Lib "user32" (lpPoint As POINTAPI) As Long
Private Const MF_BYCOMMAND = &H0&
Private Const MF_SEPARATOR = &H800&
Private Const MF_STRING = &H0&
Public Const WM_USER = &H400
'自定义菜单项的标识号偏移量
Public Const IDM_CUT = 1
Public Const IDM_COPY = 2
Public Const IDM_PASTE = 3

Public Const TPM_LEFTALIGN = &H0&
Public Type POINTAPI
    x As Long
    y As Long
End Type

Public hMenu As Long

Public Declare Function SetWindowLong Lib "user32" Alias "SetWindowLongA" (ByVal hwnd As Long,
    ByVal nIndex As Long,ByVal dwNewLong As Long) As Long
Public Declare Function CallWindowProc Lib " user32 " Alias " CallWindowProcA " ( ByVal
    lpPrevWndFunc As Long,ByVal hwnd As Long,ByVal Msg As Long,ByVal wParam As Long,ByVal
    lparam As Long) As Long
Public Declare Function DefWindowProc Lib "user32" Alias "DefWindowProcA" (ByVal hwnd As Long,
    ByVal wMsg As Long,ByVal wParam As Long,ByVal lparam As Long) As Long
'消息
Public Const GWL_WNDPROC = (-4)
Public Const WM_COMMAND = &H111

Public OldProc As Long

Sub CallMenu( )
    UserForm1.Show
End Sub
```

(4) 在窗体的初始化事件中,返回窗体和文本框的句柄,添加快捷菜单,其实现代码:

```
Private Sub UserForm_Initialize( )
    '获取用户窗体句柄
    UserFormHwnd = GetActiveWindow

    '获取文本框句柄
    TextBox1.SetFocus
    TextBoxHwnd = GetFocus
    UserFormAddMenu
End Sub
```

上面代码中使用了自定义函数 UserFormAddMenu，用来创建弹出式菜单，其实现代码：

```
Public Sub UserFormAddMenu( )
    hMenu = CreatePopupMenu( )
    AppendMenu hMenu,MF_BYCOMMAND Or MF_STRING,WM_USER + IDM_CUT,"前切(&T)"
    AppendMenu hMenu,MF_BYCOMMAND Or MF_STRING,WM_USER + IDM_COPY,"复制(&C)"
    AppendMenu hMenu,MF_BYCOMMAND Or MF_STRING,WM_USER + IDM_PASTE,"粘贴(&P)"
    OldProc = SetWindowLong(UserFormHwnd,GWL_WNDPROC,AddressOf AddCallBack)
End Sub
```

(5) 窗体中“去除 X 按钮”按钮的单击事件，用于变灰关闭按钮，其实现代码：

```
Private Sub CommandButton2_Click( )
    RemoveXButton GetActiveWindow( )
End Sub
```

上面代码中使用了自定义函数 RemoveXButton，用来去除关闭按钮，其实现代码：

```
Public Sub RemoveXButton(ByVal WindowHwnd As Long)

    Dim hMenu As Long
    Dim nCount As Long

    hMenu = GetSystemMenu(WindowHwnd,True)
    hMenu = GetSystemMenu(WindowHwnd,False)
    nCount = GetMenuItemCount(hMenu)

    Call RemoveMenu(hMenu,nCount-1,MF_REMOVE Or MF_BYPOSITION)
    Call RemoveMenu(hMenu,nCount-2,MF_REMOVE Or MF_BYPOSITION)

    DrawMenuBar WindowHwnd
End Sub
```

(6) 窗体中文本框的弹起事件，用于显示快捷菜单，其实现代码：

```
Private Sub TextBox1_MouseUp(ByVal Button As Integer,ByVal Shift As Integer,ByVal x As Single,
    ByVal y As Single)
    Dim Pt As POINTAPI
    GetCursorPos Pt
```

```
    If Button = 2 Then
        TrackPopupMenu hMenu,TPM_LEFTALIGN,Pt.x,Pt.y,0,UserFormHwnd,ByVal 0&
    End If
End Sub
```

(7) 窗体中回调的过程中,用于执行快捷菜单中的操作,其实现代码:

```
Public Function AddCallBack(ByVal hwnd As Long,ByVal wMsg As Long,ByVal wParam As Long,ByVal
    lparam As Long) As Long
    Select Case wMsg
        Case WM_COMMAND '消息
            Select Case wParam
                Case WM_USER + IDM_CUT
                    UserForm1.TextBox1.Cut
                Case WM_USER + IDM_COPY
                    UserForm1.TextBox1.Copy
                Case WM_USER + IDM_PASTE
                    UserForm1.TextBox1.Paste
            End Select
    End Select
    AddCallBack = CallWindowProc(OldProc,hwnd,wMsg,wParam,lparam)
End Function
```

(8) 在窗体的释放操作中,用于删除菜单,其实现代码:

```
Private Sub UserForm_Terminate( )
    DestroyMenu hMenu
    UserFormReleaseMenu
End Sub
```

上面代码中使用了自定义函数 UserFormReleaseMenu,用来删除弹出式菜单,其实现代码:

```
Public Sub UserFormReleaseMenu( )
  SetWindowLong UserFormHwnd,GWL_WNDPROC,OldProc
End Sub
```

(9) 最后,在窗体中"退出"按钮的单击事件中,添加用于退出程序的代码:

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

13.4.4 实例效果

(1) 在 VBA 集成开发环境中,按下 F5 键运行程序,系统会弹出如图 13.20 所示的"菜单演示"对话框。

(2) 单击"去除 X 按钮"按钮,窗体右上角的关闭按钮(X)变灰,如图 13.21 所示。

(3) 在文本框中右击,弹出快捷菜单,如图 13.22 所示。

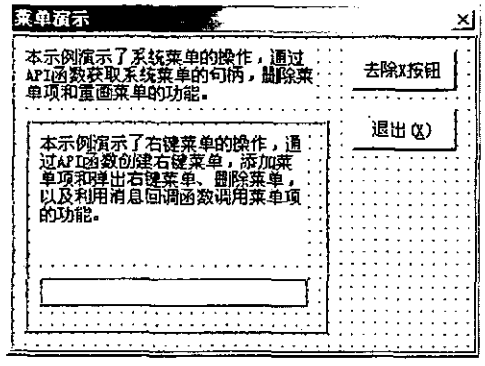


图 13.20　程序运行界面

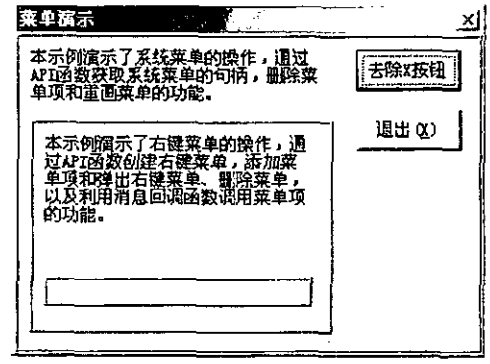


图 13.21　去除关闭按钮的效果

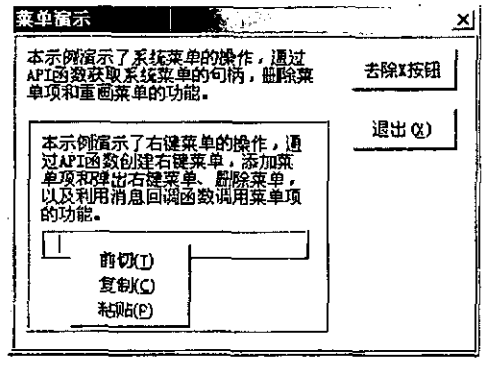


图 13.22　弹出快捷菜单

13.4.5　小结

本节介绍了获取用户窗体和控件句柄的方式，同时通过调用 Windows API 函数，实现用户窗体的特殊效果。

第 14 章　创建数据库应用程序

数据库一直是计算机软件开发的重要话题。在利用计算机辅助办公的时代，人们很难不和数据库打交道。事实上，AutoCAD 图形本身也是一个数据库。在本章中主要讨论 AutoCAD 图形与其他常用的关系型数据库连接的问题。

本章所涉及的主要型数据库对象如表 14.1 所示。

表 14.1　本章所涉及的主要型数据库对象

数据库对象	说明
ADODB. Connection	ADO 数据库连接对象，表示打开的与数据库之间的连接
ADODB. Command	ADO 数据库命令对象，可以通过命令对象执行 SQL 查询语句
ADODB. Recordset	ADO 数据库记录集对象，其中可包含多个数据库记录
ADODB. Recordset. Fields	ADO 数据库记录集的字段集合

ADO 是在 AutoCAD VBA 中开发数据库应用程序最好的技术，因此本章的内容将 ADO 的使用作为重点，在对数据操作方面，直接使用 SQL 语句操作数据库具有较高的灵活性、更好的效率和强大的功能，是推荐使用的方法。惟一的缺点是 SQL 语句的掌握需要一点时间。

GIS 目前是一个热门的技术，其核心思想就是数据记录和地理实体的对应，许多行业已经使用 AutoCAD 创建了地理实体，那么为其加上属性数据（保存在数据库中）就能模拟一个简单的 GIS 系统。本章分析了数据库记录和 AutoCAD 实体对应的方法，这是最为重要的一个问题。

14.1　创建数据库的 ADO 连接

14.1.1　实例说明

本节解答在 AutoCAD 的 VBA 中使用代码建立数据库 ADO 连接的问题，然后利用建立的连接创建一个数据库表。

14.1.2　编程思路

ADO（ActiveX Data Objects）提供一个能够在应用程序中使用数据库的编程模型，利用 ADO 编程模型可以完成访问和更新数据源的工作。ADO 2.5 编程模型主要由连接（Connection）、命令（Command）、记录集（Recordset）、字段（Field）、参数（Parameters）、错误（Error）和属性（Property）等 7 个对象及对象集合构成。可以通过访问这些对象或者对象集合来操作数据库。

Connection 对象代表打开的与数据源的连接。每一个成功的连接表示和数据源的一

次会话，包括打开数据源到关闭与数据源的连接之间的所有操作。建立与数据源的连接后，可以使用 Connection 对象的方法和属性执行各种操作，例如：

- 使用 ConnectionString、ConnectionTimeout 和 Mode 等属性对连接进行初始化。
- 使用 Provider 属性指定 OLE DB 提供者。
- 使用 Open 方法建立到数据源的连接，使用 Close 方法切断连接。
- 使用 Execute 方法执行对数据连接的命令。
- 使用 BeginTrans、CommitTrans 和 RollbackTrans 方法管理连接上的事务。
- 使用 Errors 集合检查数据源返回的错误等。

14.1.3　步骤

(1) 在 AutoCAD 2005 中，打开 VBA 管理器，创建一个新工程，保存在适当的位置，进入 VBA 集成开发环境。

(2) 要在 VBA 中使用 ADO，首先必须在 VBA 工程中引用 ADO 对象库。在 VBA 中，根据用户使用的数据源不同提供了不同的 ADO 版本。要添加对 ADO 对象的引用，可在 VBA 中选择“工具”|“引用”命令，打开“引用”对话框，在“可使用的引用”列表框中选择要引用的 ADO 对象库。这里为方便起见，直接引用了 ADO 2.5 的类型库，如图 14.1 所示。

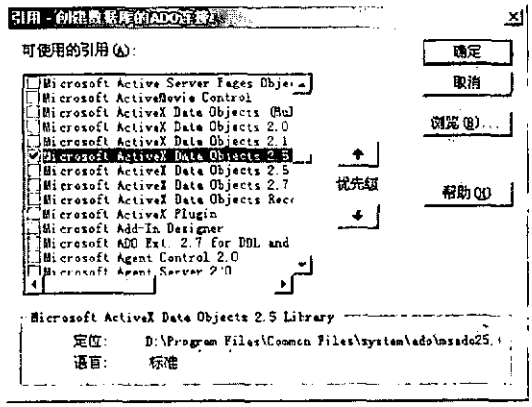


图 14.1　引用 ADO 2.5 类型库

若要使用 Access 97 数据库，可以引用 Microsoft ActiveX Data Objects 2.0 Library 对象库，使用 Access 2000 或者 Access XP 数据库，建议引用 Microsoft ActiveX Data Objects 2.1 Library 对象库。

(3) 为了本章讨论方便，下面创建一种空白的 Access 数据库。当然如果只是想在 AutoCAD 中使用已经创建好的 Access 数据库，可以跳过这一步。

启动 Microsoft Access 应用程序，从启动向导中选择空白数据库选项，在“保存新数据库”对话框中指定好数据库保存路径及文件名，在这假设为 D:\VBASample.mdb。也可以在 Access 应用程序菜单上选择“文件”|“新建数据库”菜单项来创建空白数据库。

(4) 创建好空白数据库后，可以单击 Access 应用程序窗口的关闭按钮，或使用文件菜单的“关闭”命令关闭数据库。另外，还可以在 VBA 中使用代码来创建空白数据库。在 VBA 中创建空白数据库，首先要用类似步骤(1)的方法引用合适的 Access 对象库，然后使

用以下代码创建新的空白数据库：

```
Sub CreateAccessDB( )
    Dim AccessApp As Access.Application      '首先声明一个 Access 应用程序对象
    Set AccessApp = CreateObject("Access.Application.9")  '创建 Access 应用程序对象

    On Error Resume Next
    AccessApp.NewCurrentDatabase "D:\VBASample.mdb"    '创建新的空白数据库并设置为当前数据库

    If Err.Number = 7865 Then      '若已经存在同名数据库文件
        MsgBox "已经存在名称为 D:\VBASample.mdb 的文件",,"明经通道 VBA 示例"
        Err.Clear
        AccessApp.OpenCurrentDatabase "D:\VBASample.mdb"  '直接打开数据库并设置为当前数据库
    End If

    If Not Err Then
      MsgBox "已经创建了名为 D:\VBASample.mdb 的 Aceess 数据库文件",,"明经通道 VBA 示例"
    End If

    On Error GoTo 0     '恢复 VBA 错误处理方式

    AccessApp.CloseCurrentDatabase     '关闭数据库
    Set AccessApp = Nothing      '回收 AccessApp 对象占用的内存
End Sub
```

程序中使用 CreateObject 方法创建 Access 应用程序对象时要注意创建对象的 Access 版本。表 14.2 列出了常用 Access 版本应该引用的对象库和创建应用程序方法，本例程序中引用的是 Microsoft Access 9.0 Object Library，引用的方法可以参考步骤(2)。

表 14.2 各常用版本 Access 应用程序对应的对象库和创建方法

Access 应用程序版本	需引用的对象库	创建应用程序对象的方法
MS Access 97	Microsoft Access 8.0 Object Library	CreateObject("Access.Application.8")
MS Access 2000	Microsoft Access 9.0 Object Library	CreateObject("Access.Application.9")
MS Access XP	Microsoft Access 10.0 Object Library	CreateObject("Access.Application.10")

程序使用了一种比较简单的方法处理已经存在同名数据库的错误，其中的错误代码 Err.Number 可能与 VBA 的版本等有关。

注意：及时关闭不再使用的数据库对象和 Access 应用程序对象。

(5) 要建立 ADO 连接，首先要声明一个新的连接(Connection)对象，然后调用它的 Open 方法。该对象的 Open 方法完整语法如下：

```
Connection.Open [ConnectionString As String],[UserID As String],[Password As String],
    [Options As Long = -1]
```

其中，ConnectionString 为连接字符串，用于提示数据源提供者名称和数据库路径。UserID 和 PassWord 分别为打开数据库所需的用户名和密码。

OpenOptions 为数据库打开选项。如果设置为 adConnectAsync，则异步打开连接。当连接可用时将产生 ConnectComplete 事件。

使用 Connection 对象的 Open 方法可建立到数据源的物理连接。在该方法成功完成后连接是活跃的，可以对它发出命令并且处理结果。在程序中添加下面的宏用于创建一个 ADO 连接，并且打开前面创建的 Access 文件：

```
'因为多个程序都要用过 ADO 连接对象，声明一个模块级变量
Dim cn As ADODB.Connection

'创建 ADO 连接并打开
Sub CreateConnection( )
    Dim ConStr As String     '连接字符串
    Set cn = New ADODB.Connection

    cn.CursorLocation = adUseClient
    cn.Provider = "Microsoft.Jet.OLEDB.4.0;"
    ConStr = "Data Source = d:\VBASample.mdb;"
    cn.Open ConStr

    MsgBox "已经创建了一个到 D:\VBASample.mdb 的 ADO 连接",,"明经通道 VBA 示例"
End Sub
```

程序中首先在内存中新建一个 ADODB.Connection 连接对象，然后设置连接对象的游标位置和数据源提供者。程序中连接对象的 Provider 提供者属性设置为 Microsoft Aceess 2000 数据库，若使用 Microsoft Access 97 数据库，提供者属性应设置为 Microsoft.Jet.OLEDB.3.51。建立的连接对象在其他程序中还要使用，因此以上程序没有关闭 ADODB.Connection 连接对象。

14.1.4　实例效果

在 VBA 集成开发环境中，执行 CreateAccessDB 宏，程序就自动在 D 盘根目录下创建了名称为 VBASample.mdb 的数据库文件。

14.1.5　小结

打开 Access 的 ADO 连接对象，也可以在连接字符串中指定数据源的提供者，例如：

```
cn.Open "Provider = Microsoft.Jet.OLEDB.3.51;Data Source = D:\VBASample.mdb"
```

ADO 是“ActiveX 数据对象”的英文缩写，它建立在 OLE DB 之上。ADO 是一个 OLE DB 使用程序，使用 ADO 的应用程序都要间接地使用 OLE DB。直接使用 OLE DB 必须使用复杂的 COM 接口，而 ADO 简化了 OLE DB，在 VBA 中，能够利用 ADO 提供的自治接口连接并使用数据库。

本节讲述了建立到 Access 数据库的 ADO 连接的方法，同时讲述了借助 Access 的 ActiveX 对象模型创建空白 Access 数据库的方法。建立 ADO 连接的目的是对数据库进行读写操作，这样就要使用 ADO 连接的 Execute 方法或者 ADO 的 Command 和 Recordset 对象。

14.2 创建数据表

14.2.1 实例说明

本节介绍在指定数据库文件中创建数据表的方法，然后将 AutoCAD 图形信息保存到指定数据表中。

14.2.2 编程思路

建立 Access 数据表可以使用 ADO 编程模型的 Command 对象执行 SQL 语句 CREATE TABLE 来实现。将 AutoCAD 图形数据库信息写入数据库，可以执行 SQL 语句 INSERT INTO 向指定数据表添加记录。

问题的关键是，若数据库中已经存在相同名称的数据表时，SQL 语句的执行会出错。要避免这个错误，执行 SQL 语句前需要判断数据库中是否存在同名数据表。

另外，执行 SQL 语句也可以使用 ADODB. Connection. Execute 方法。

14.2.3 步骤

(1) 在 AutoCAD 2005 中，打开 VBA 管理器，创建一个新工程，保存在适当的位置。进入 VBA 集成开发环境，按照上一节介绍的方法，创建与 Access 数据库的连接，以下的步骤中假设该连接对象名为 cn。

(2) 判断 Access 数据库中是否存在要创建的同名数据表。ADODB. Connection 对象的 OpenSchema 方法可以从数据库中获取其摘要信息。该方法返回一个包含数据库摘要信息的 Recordset 记录集对象。以下 VBA 函数判断连接对象 cn 所连接的数据库中是否存在与输入字符串 InputStr 同名的数据表：

```
'函数 HasTable,判断数据库中是否存在某数据表
'输入参数：cn--ADODB.Connection 类型,ADO 数据库连接对象
        'InputStr--字符串,要判断是否存在的数据表名称,忽略大小写区别
'返回值：若数据库中存在与 InputStr 同名的数据表,返回 True,否则返回 False
Function HasTable(cn As ADODB.Connection,InputStr As String) As Boolean
    Dim rst As New ADODB.Recordset  '新建记录集对象
    Set rst = cn.OpenSchema(adSchemaTables) '获取数据库全部数据表名称

    Dim i As Integer
    For i = 0 To rst.RecordCount-1
        If UCase(rst.Fields("TABLE_NAME")) = UCase(InputStr) Then  '比较数据表名称与输入字符串
            HasTable = True  '存在与 InputStr 同名的数据表
            rst.Close      '及时关闭记录集对象
            Exit Function
        End If
        rst.MoveNext
```

```
    Next i

    HasTable = False  '不存在与 InputStr 同名的数据表
    rst.Close  '及时关闭记录集对象
End Function
```

(3) 创建 ADODB. Command 命令对象执行 SQL 语句。建立与 ADO 数据源的连接后,可以先声明一个 ADODB. Command 类型的对象变量,然后设置该对象变量的 ActiveConnection 属性和 CommandText 属性,指定命令使用的连接和命令文本字符串。命令文本字符串就是完整的 SQL 语句串,新建 Access 表使用 SQL 语句 CREATE TABLE,其完整语法为:

```
CREATE TABLE 表名 (字段名 1　类型[,字段名 2　类型,……][其他参数]);
```

其中可选项[其他参数]是与物理存储有关的参数,由具体数据库系统决定,在 Access 数据库中无须指定。以上参数表中,每个字段的类型后加 SQL 关键字 NOT NULL,指定某些字段不允许使用 NULL 值。

例如,以下代码段创建一个基于连接 cn 的命令对象 cmd,并指定其命令文本为建立数据表 line,该表 4 个字段分别存储 AutoCAD 直线对象的起点和终点坐标值,字段数据类型为 float 浮点数。

```
Set cmd = New ADODB.Command
Set cmd.ActiveConnection = cn
If Ch17_3_HasTable(cn,"line") = False Then
  cmd.CommandText = "CREATE TABLE line(X1 float,Y1 float,X2 float,Y2 float);"
  cmd.Execute
Else
  MsgBox "数据库中已经存在数据表 line,不能创建",,"明经通道 VBA 示例"
End If
```

实际上,执行 SQL 语句,也可以不新建 Command 对象而直接使用 Connection 对象的 Execute 方法:

```
cn.Execute "CREATE TABLE line(X1 float,Y1 float,X2 float,Y2 float);"
```

(4) 读取 AutoCAD 图形信息并写入数据库。在以上步骤创建的数据表中写入 ACAD 图形信息,也就是在数据表中填写记录,可以使用 SQL 语句 INSERT INTO,该语句的完整语法为:

```
INSERT INTO 表名 (字段名 1 [,字段名 2,……] values(常数 1 [,常数 2,……]);
```

该语句在指定数据表中插入一条记录,且指定字段具有给定的值,即第 i 个字段对应第 i 个常数。例如某直线对象起点为(10,15),终点为(20,10),在以上创建的数据表 line 中插入表示该直线对象的记录,SQL 语句可以这样写:

```
INSERT INTO line(X1,Y1,X2,Y2) values(10,15,20,10);
```

注意: INSERT INTO 语句可以省略字段名,字段取值对应表原始定义的字段名,从而以上 SQL 语句可以这样写:

```
INSERT INTO line values(10,15,20,10);
```

这种方法固然方便、省事，但潜在一个危险，当数据表结构被修改，比如增加一个字段，就要出问题。

(5) 添加一个函数 FindObj，用于在指定的记录集中查找某个对象 ID，其相关代码为：

```
Private Function FindObj(rst As ADODB.Recordset,Id As String) As Boolean
    If rst.RecordCount > 0 Then
        rst.MoveFirst

        While Not rst.EOF
            If Trim(rst.Fields("id")) = Id Then
                FindObj = True
                Exit Function
            End If
            rst.MoveNext
        Wend

        FindObj = False
    Else
        FindObj = False
    End If
End Function
```

(6) 创建一个宏 AcadWriteToDb，将 ACAD 图形中的直线、圆、圆弧对象信息写入相应的数据表中：

```
Dim cmd As ADODB.Command      'ADO 命令对象
Dim rst As ADODB.Recordset

'将 ACAD 当前图形中直线和圆、圆弧对象写入数据库
Public Sub AcadWriteToDB( )
    Call CreateConnection        '建立 ADO 连接
    Set cmd = New ADODB.Command
    Set cmd.ActiveConnection = cn

    '创建 line,circle,arc 三个数据表
    If HasTable(cn,"line") = False Then
        cmd.CommandText = "CREATE TABLE line(id char(10),X1 float,Y1 float,X2 float,Y2 float);"
        cmd.Execute
    End If
    If HasTable(cn,"circle") = False Then
        cmd.CommandText = "CREATE TABLE circle(id char(10),CenX float,CenY float,Rad float);"
        cmd.Execute
    End If
    If HasTable(cn,"arc") = False Then
```

```
    cmd.CommandText = "CREATE TABLE arc(id char(10),CenX float,CenY float,Rad float," & _
                      "StartAng float,EndAng float);"
    cmd.Execute
End If

'读取 ACAD 图形信息并写入数据库
Set rst = New ADODB.Recordset
rst.CursorLocation = adUseClient
Dim pt1 As Variant,pt2 As Variant
Dim obj As AcadEntity

For Each obj In ThisDrawing.ModelSpace
    Select Case obj.ObjectName
        Case "AcDbLine":
            rst.Open "SELECT id FROM line",cn,adOpenForwardOnly,adLockReadOnly,adCmdText
            pt1 = obj.StartPoint
            pt2 = obj.EndPoint

            If Not FindObj(rst,obj.ObjectID) Then      '新建的直线的处理
                cmd.CommandText = "INSERT INTO line(id,X1,Y1,X2,Y2) VALUES(" & _
                    "'" & obj.ObjectID & "'," & pt1(0) & "," & pt1(1) & "," & pt2(0) & "," & pt2(1) & ");"
            Else                                       '修改的直线的处理
                cmd.CommandText = "UPDATE line SET X1 = " & pt1(0) & ",Y1 = " & pt1(1) & ",X2 = " &
                    pt2(0) & ",Y2 = " & pt2(1) & " WHERE id = '" & obj.ObjectID & "';"
            End If

            cmd.Execute
            rst.Close
        Case "AcDbCircle":
            rst.Open "SELECT id FROM circle",cn,adOpenForwardOnly,adLockReadOnly,adCmdText
            pt1 = obj.Center

            If Not FindObj(rst,obj.ObjectID) Then
                cmd.CommandText = "INSERT INTO circle(id,CenX,CenY,Rad) VALUES(" & _
                    "'" & obj.ObjectID & "'," & pt1(0) & "," & pt1(1) & "," & obj.Radius & ");"
            Else
                cmd.CommandText = "UPDATE circle SET CenX = " & pt1(0) & ",CenY = " & pt1(1) & _
                    ",Rad = " & obj.Radius & " WHERE id = '" & obj.ObjectID & "';"
            End If

            cmd.Execute
            rst.Close
        Case "AcDbArc":
```

```
                rst.Open "SELECT id FROM arc",cn,adOpenForwardOnly,adLockReadOnly,adCmdText
                pt1 = obj.Center

                If Not FindObj(rst,obj.ObjectID) Then
                    cmd.CommandText = "INSERT INTO arc(id,CenX,CenY,Rad,StartAng,EndAng) VALUES(" & _
                        "'" & obj.ObjectID & "'," & pt1(0) & "," & pt1(1) & "," & obj.Radius & "," & _
                        obj.StartAngle & "," & obj.EndAngle & ");"
                Else
                    cmd.CommandText = "UPDATE arc SET CenX = " & pt1(0) & ",CenY = " & pt1(1) & _
                        ",Rad = " & obj.Radius & ",StartAng = " & obj.StartAngle & ",EndAng = " & _
                        obj.EndAngle & " WHERE id = '" & obj.ObjectID & "';"
                End If

                cmd.Execute
                rst.Close
            End Select
        Next obj

        Set cmd = Nothing
        cn.Close
        Set cn = Nothing
    End Sub
```

以上代码所建立的数据表存在一些问题，比如说，由于数据没有设置主键，当多次运行VBA宏时，就会在数据表中插入大量重复的数据。

14.2.4 实例效果

(1) 在AutoCAD 2005中，使用LINE、CIRCLE和ARC命令创建如图14.2所示的实体。

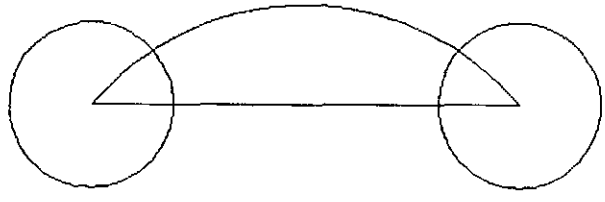


图14.2 创建测试用的实体

(2) 在VBA集成开发环境中，执行AcadWriteToDB宏，系统会弹出如图14.3所示的对话框。单击“确定”按钮，继续执行程序。

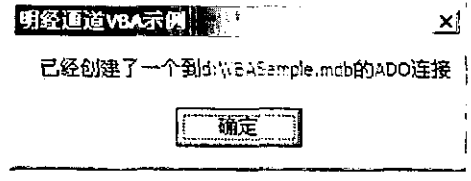


图14.3 创建ADO连接的提示

(3) 打开 D 盘根目录下的 VBASample. mdb 文件，其中已经包含了 line、arc 和 circle 三个表，可以打开某个表查看其中的内容，如图 14.4 所示。

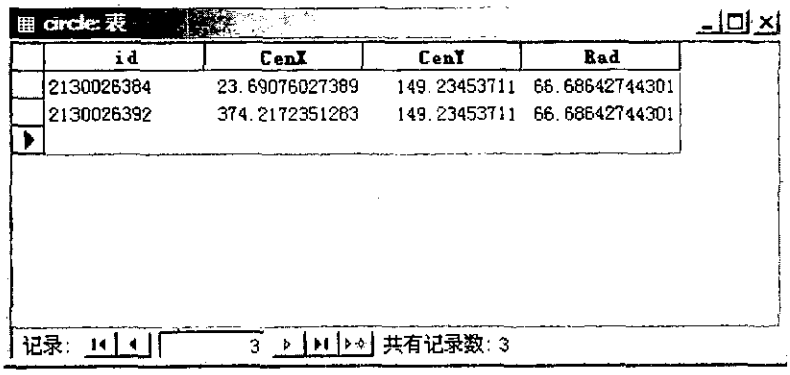


图 14.4　查看 circle 表的内容

14.2.5　小结

本节通过 ADO 编程模型的 Command 命令对象执行 SQL 语句，在数据库中创建数据表并向数据表中填写记录。在实际编程过程中，应该注意 SQL 语句的正确书写，因为 VBA 编程环境不能自动检查 SQL 语句的正确性。

在向数据库创建表之前，应该检查数据库中是否存在同名数据表。这一点可以通过浏览数据库连接的 OpenSchema 方法返回的记录集来判断。

14.3　使用记录集从数据表中读取数据

14.3.1　实例说明

本节的实例利用数据库记录集对象，从指定数据表中查询信息，并根据查询结果在 AutoCAD 窗口中添加图形对象。

14.3.2　编程思路

使用 ADO 编程模型 Command 对象的 Execute 方法执行 SQL 语句时，实际上会返回一个 Recordset 记录集对象。当然也可以直接使用记录集对象的 Open 方法，记录集打开后就可以使用 Move 等方法移动记录指针，从而浏览记录集对象所包含的全部数据。

14.3.3　步骤

(1) 在 AutoCAD 2005 中，打开 VBA 管理器，加载上一节创建的工程，然后将其另存为新的工程。本节的工程将在上一节工程的基础上继续添加功能。

(2) 建立 ADO 连接及其 Command 对象。使用前两节的方法建立 ADO 连接和 Command 对象：

```
Dim cn As ADODB.Connection      'ADO 连接对象
Dim cmd As ADODB.Command        'ADO 命令对象
Ch17_2_CreateConnection         '建立 ADO 连接
Set cmd = New ADODB.Command
Set cmd.ActiveConnection = cn
```

(3) 建立 SQL 查询语句。从数据库中读取数据,使用 SQL 的 SELECT 语句。例如以下 SQL 语句选择数据表 line 记录中所有起点信息:

```
SELECT X1,Y1 FROM line;
```

要查询数据表中所有字段的信息,可以使用带星号"*"的 SELECT 语句:

```
SELECT * FROM line;
```

在 SELECT 语句中可以带有 ORDER BY 子句,对查询结果进行排序,也可以使用 WHERE 子句给出查询条件。例如以下 SQL 语句查询所有起点的纵坐标值大于 100.0 的直线对象的全部信息,且按起点横坐标值升序排序:

```
SELECT * FROM line WHERE YI>100.0 ORDER BY X1;
```

由于上节建立的数据可能包含有重复记录,可以使用 SQL 语言的 DISTINCT 关键字来消除查询结果中的重复记录,例如:

```
SELECT DISTINCT * FROM line;
```

提示: SQL 的数据操纵功能十分强大,以上仅仅介绍了最简单的 SELECT 语句的编写方法,要了解 SELECT 语句的详细语法及使用,可参考讲述数据库原理的教程或者 SQL 语言标准文本。

(4) 建立并打开记录集对象。记录集对象可以理解为一个记录的集合,它是由字段(或列)和记录(行)的形式构成的二维表,用来显示或查询数据。

要使用 ADO 的记录集,首先要声明一个新的记录集(Recordset)对象,然后调用它的 Open 方法。该对象的 Open 方法比 ADODB.Connection 的 Open 方法更为灵活,其完整语法如下:

```
Recordset.Open [Source],[ActiveConnection],[CursorType],[LockType],[Options As Long]
```

其中,Source 为要打开记录集对象的数据源,它可以是有效的 ADODB.Command 对象或者 SQL 语句、数据表名称等。

ActiveConnection 为要打开的记录集对象使用的活动连接,它可以是有效的 Connection 连接对象或者是一个连接字符串,字符串的内容与 Connection 对象的 ConnectionString 属性,例如 CreateConnection 宏中的 ConStr 变量。

CursorType 为打开的记录集对象使用的游标类型。这个参数多用于共享的多用户数据库系统,对于本地数据库,直接使用其默认值 adOpenForwardOnly 即可。

LockType 为记录集的锁定类型,即更新记录集内容的方法,该参数可设置如表 14.3 所示的某个 LockTypeEnum 的值。

表 14.3　Recordset 对象的 LockType 参数值

LockType 参数值	说明
adLockReadOnly	默认值，只读。无法更改数据
adLockPessimistic	保守式记录锁定(逐条)。数据源提供程序通常在编辑记录时立即更新数据源的记录，以确保成功编辑记录。从而在 VBA 编程时，只要对记录集内容有任何编辑操作就将立即更新数据源
adLockOptimistic	开放式记录锁定(逐条)。编程时可以在程序中先编辑某条记录，然后调用 Update 方法，数据源提供程序在此时更新记录
adLockBatchOptimistic	开放式批更新。编程时可以在程序中编辑整个记录集的各条记录，然后调用 UpdateBatch 方法更新记录集

Options 参数为长整型，该参数用于指示数据源提供程序如何确定 Source 参数的类型(如果该参数不是 Command 类型)。若编程时确知 Source 参数的类型，建议指定该参数以提高程序打开记录集对象的效率。如表 14.4 所示，为常用的几个 Options 参数值及说明。

表 14.4　Recordset 对象的 Options 参数值

Options 参数值	Source 参数类型及说明
adCmdText	Source 参数为命令文本。记录集对象打开时将此参数作为命令或者存储过程定义的文本(字符串)
adCmdTable	Source 参数为数据表名称。记录集对象打开时，ADO 将生成 SQL 查询，并返回 Source 参数命名的数据表
adCmdTableDirect	Source 参数为数据表名称。记录集对象打开时，ADO 将返回该数据表的全部记录
adCmdStoredProc	Source 参数为数据库存储过程名称
adCmdUnkown	默认值。Source 参数为未知类型

注意：Options 参数指定与 Source 参数类型不匹配时，打开记录集对象时将产生错误。

以下代码建立一个记录集对象，包括数据表 line 中的全部记录并消除重复内容。

```
Set rst = New ADODB.Recordset
rst.CursorLocation = adUseClient
rst.Open "SELECT DISTINCT * FROM line",cn,adOpenForwardOnly,_
         adLockBatchOptimistic,adCmdText
```

若已经创建并设置了 ADODB.Command 对象后，记录集也可以使用以下代码打开：

```
rst.Open cmd,,adOpenForwardOnly,adLockBatchOptimistic
```

注意：打开记录集对象时，若 Source 参数指定为 Command 对象，不要同时使用 ActiveConnection 参数。

(5) 读取记录集数据，并在 AutoCAD 中绘图。记录集对象的 Fields 属性可以返回记录集对象当前记录指定字段的内容。该属性有一个参数，它可以是一个表示字段顺序号的整数，也可以是用字符串表示的字段名。

修改记录集对象的当前记录，可使用记录集对象的 MoveFirst、MoveLast、MoveNext 或 MovePrevious 方法，它们分别移动记录指针到记录集对象中的第一个、最后一个、下一个或上一个记录，并使该记录成为当前记录。

以下是读取记录集数据并在 AutoCAD 中绘图的完整代码：

```
Public Sub ReadFromDB( )
    '创建数据库连接
    Call CreateConnection

    '在 line 表中查询所有的记录
    Set rst = New ADODB.Recordset
    rst.CursorLocation = adUseClient
    rst.Open "SELECT DISTINCT * FROM line",cn,adOpenForwardOnly,_
            adLockBatchOptimistic,adCmdText
    rst.MoveFirst

    Dim i As Integer
    Dim pt1(0 To 2) As Double,pt2(0 To 2) As Double

    '使用查询得到的数据创建直线
    For i = 0 To rst.RecordCount-1
        pt1(0) = rst.Fields("x1")
        pt1(1) = rst.Fields("y1")
        pt1(2) = 0
        pt2(0) = rst.Fields("x2")
        pt2(1) = rst.Fields("y2")
        pt2(2) = 0

        ThisDrawing.ModelSpace.AddLine pt1,pt2
        rst.MoveNext
    Next i

    rst.Close       '关闭记录集，准备下一次使用

    '在 circle 表中查询所有记录
    rst.Open "SELECT DISTINCT * FROM circle;",cn,adOpenForwardOnly,_
            adLockBatchOptimistic,adCmdText
    rst.MoveFirst

    Dim Rad As Double

    '使用表中的数据绘制圆
```

```
    For i = 0 To rst.RecordCount-1
        pt1(0) = rst.Fields("CenX")
        pt1(1) = rst.Fields("CenY")
        pt1(2) = 0
        Rad = rst.Fields("Rad")
        ThisDrawing.ModelSpace.AddCircle pt1,Rad
        rst.MoveNext
    Next i

    rst.Close

    '在 arc 表中查询所有记录
    rst.Open "SELECT DISTINCT * FROM arc;",cn,adOpenForwardOnly,_
            adLockBatchOptimistic,adCmdText
    rst.MoveFirst

    Dim StartAng As Double,EndAng As Double

    '使用表中的数据绘制圆弧
    For i = 0 To rst.RecordCount-1
        pt1(0) = rst.Fields("CenX")
        pt1(1) = rst.Fields("CenY")
        pt1(2) = 0
        Rad = rst.Fields("Rad")
        StartAng = rst.Fields("StartAng")
        EndAng = rst.Fields("EndAng")
        ThisDrawing.ModelSpace.AddArc pt1,Rad,StartAng,EndAng
        rst.MoveNext
    Next i

    ThisDrawing.Application.Update
End Sub
```

14.3.4　实例效果

在 VBA 集成开发环境中，执行 ReadFromDB 宏，能够在图形窗口得到如图 14.5 所示的结果，这与上一节在图形窗口中创建的图形对象是完全一致的。

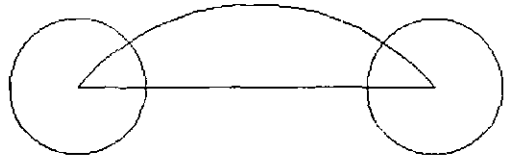


图 14.5　使用数据库中的数据绘图的结果

14.3.5 小结

由于多数数据库系统采用了高效算法来解决记录集的条件查询、排序等问题，若没有特别的原因，通常将按条件构造记录集、对记录集排序等工作都使用 SQL 查询语句来完成，而不是在 VBA 程序中完成。若 VBA 程序使用的是远程数据库，为减轻数据库服务器的负担，有时使用 VBA 程序的客户机(Client)来完成记录集查询或排序工作。

批量读取数据库中的数据，最为方便的方法是使用记录集对象。合理构造记录集对象，最重要的问题是正确编写 SQL 查询语句。记录集对象正确打开后，通过读取其 Fields 属性获得数据库中的数据，或者通过使用 MoveFirst、MoveLast、MoveNext 或 MovePrevious 方法移动记录指针。

14.4 使用 ADODC 控件

14.4.1 实例说明

本节的实例将在用户窗体中使用 ADODC 控件，设计一个可实现用户选择性地向数据库添加图形信息，或者从数据库中读取信息、添加 AutoCAD 图形对象的完整 VBA 程序。

14.4.2 编程思路

解决选择性添加或保存图形的问题，比较方便的方法是使用用户窗体。在用户窗体中，可以使用 ADODC 控件，从而方便地操作数据库。

14.4.3 步骤

(1) 在 AutoCAD 2005 中，打开 VBA 管理器，创建一个新工程，保存在适当的位置。进入 VBA 集成开发环境，选择“插入”|“用户窗体”菜单项，向工程中添加一个对话框，并在其中添加如图 14.6 所示的控件。

其中，添加 ADODC 控件之前，必须在工具箱上右击，从弹出的快捷菜单中选择“附加控件”菜单项，然后在弹出的对话框中选中“Microsoft ADO Data Control, version 6.0 (OLE DB)”选项，关闭“附加控件”对话框之后，就能从工具箱中添加 ADODC 控件了。

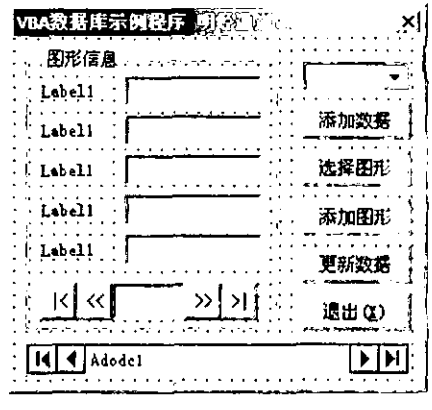


图 14.6 设计窗体

窗体中各控件的属性如表 14.5 所示。

(2) 设置 ADODC 数据库控件连接字符串属性。与指定 ADODB.Connection 对象的 ConnectionString 属性相比，设置 ADODB 数据库控件的连接字符串属性可以通过对话框来完成，从而更为方便、更不易出错。

表 14.5　VBA 数据库示例程序窗体控件说明

控件名称	控件类型	Caption 属性	说　明	备　注
Label1-5	Label	(代码指定)	图形各种不同信息名称,在代码中指定为数据库字段名	ComboBox1 选择圆时,Label5 不可见
TextBox1-5	Text Box	(代码指定)	图形各种不同信息值,在代码中指定为数据库字段值	ComboBox1 选择圆时,TextBox5 不可见
TextBox6	Text Box	(代码指定)	显示用户选择的图形数目和当前操作的图形 Index 值	
ComboBox1	Combo Box		用户选择要操作的图形类型	示例程序仅演示两种类型:线和圆
Adodc1	Adodc	(代码指定)	数据库控件。用户可移动记录指针	
cmdSelFirst	Command Button	|<	用户选择的第一个图形对象	仿照数据库控件,用4个按钮和一个文本框实现在用户选择集中移动指针
cmdSelPrev	Command Button	<<	用户选择的上一个图形对象	
cmdSelNext	Command Button	>>	用户选择的下一个图形对象	
cmdSelLast	Command Button	>|	用户选择的最后一个图形	
cmdAdd2Db	Command Button	添加数据	将用户选择的指定图形信息添加到数据库中	
cmdSel	Command Button	选择图形	用户选择要操作的 ACAD 图形	
cmdAdd 2Acad	Command Button	添加图形	将用户指定的数据库记录添加到 ACAD 图形中	
cmdUpdate	Command Button	更新数据	将用户对图形信息的修改保存到数据库中	
cmdExit	Command Button	退出	退出系统	

在 VBA 属性窗口中选择 ConnectionString 属性,单击"…"按钮,在弹出的对话框中可以选择数据库连接方式,本例使用连接字符串,单击"Build..."按钮。根据数据源的不同,也可以选择数据连接文件或 ODBC 连接。

(3) 在弹出的"数据链接属性"对话框中,首先确定数据库提供程序名称,在本例中选择 Microsoft Jet 4.0 OLE DB Provider,然后单击"下一步"按钮,进行数据库连接设置,如图 14.7 所示。

(4) 选择或者输入数据库名称后,若数据库要求用户名和密码,输入登录数据库的信息。在输入数据库连接属性后,可以单击"测试连接"按钮,检查数据库连接属性是否设置正确。数据库连接设置正确后,单击"确定"按钮,返回 VBA 属性窗口。此时,ADODC 控件的连接字符串属性已经正确填写好。

(5) 本例程序需要使用两个模块级局部变量。sel 变量为 ACAD 选择集对象,用于保存用户从 AutoCAD 图形界面中选择的图元对象。IsUserOprt 为布尔型变量,当用户在窗体中操作移动数据库记录时,该变量值为 True。用户在窗体中移动数据库记录时,需要在窗体中显示数据库记录内容,而由于编程需要,对数据库记录操作时不必影响窗体显示。

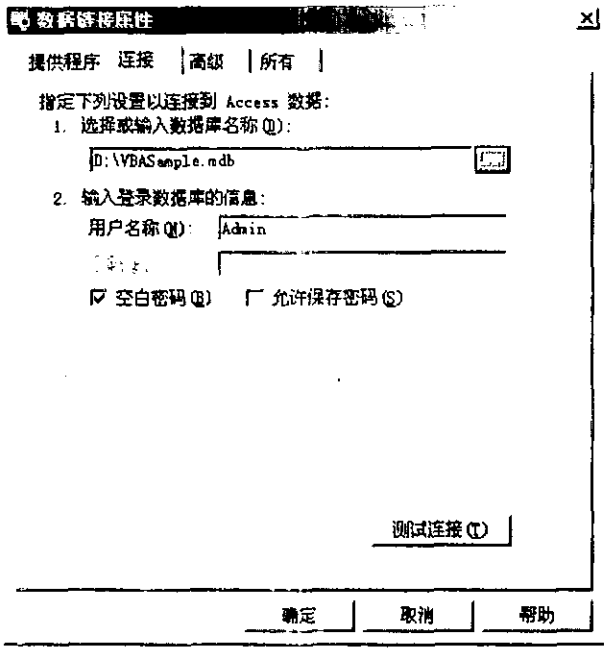


图 14.7 选择 OLE DB 提供程序

在窗体代码窗口的通用区,添加两个模块级变量的声明:

```
Dim sel As AcadSelectionSet
Dim IsUserOprt As Boolean
```

(6) 本例为了简化代码,仅仅处理直线和圆两种 AutoCAD 对象。还可根据编程需要添加对其他 AutoCAD 对象类型的支持代码。双击用户窗体,编写 UserForm_Initialize 事件代码:

```
Private Sub UserForm_Initialize( )
    IsUserOprt = False      '在本事件中,操作数据库记录不必影响窗体显示
    ComboBox1.AddItem "圆"
    ComboBox1.AddItem "线"
    ComboBox1.Text = "线"
    Adodc1.RecordSource = "select * from line;"     '窗体初始化时,先打开 line 数据表
    Adodc1.Refresh      '设置数据源后,需刷新 ADODC 控件

    With Adodc1.Recordset
        Call ShowDb 'ShowDb 过程用于在窗体中显示数据库当前记录内容

        Label1.Caption = .Fields(0).Name      '提取数据库记录字段名,并在窗体中显示
        Label2.Caption = .Fields(1).Name
        Label3.Caption = .Fields(2).Name
        Label4.Caption = .Fields(3).Name
        Label5.Caption = .Fields(4).Name
```

```
        '在 ADODC 控件中填写数据库记录集信息
        Adodc1.Caption = "第" & .AbsolutePosition & "/共" & .RecordCount & "条记录"
    End With

    cmdAdd2Db.Enabled = False      '用户尚未选择 ACAD 图形对象,不能向数据库添加记录
    IsUserOprt = True

    '初始化 AutoCAD 选择集对象
    On Error Resume Next

    Set sel = ThisDrawing.SelectionSets.Add("Sel")
    If Err Then
        Err.Clear
        ThisDrawing.SelectionSets.Item("Sel").Delete
        Set sel = ThisDrawing.SelectionSets.Add("Sel")
    End If

    On Error GoTo 0
End Sub
```

代码中 ShowDb 过程的实现代码如下：

```
Private Sub ShowDb( )
    With Adodc1.Recordset
        If Not .EOF And Not .BOF Then
            TextBox1.Text = .Fields(0)
            TextBox2.Text = .Fields(1)
            TextBox3.Text = .Fields(2)
            TextBox4.Text = .Fields(3)

            If ComboBox1.Text = "线" Then TextBox5.Text = .Fields(4)
        End If
    End With
End Sub
```

(7) 编写 ADODC 控件的事件处理代码：

```
Private Sub Adodc1_MoveComplete(ByVal adReason As ADODB.EventReasonEnum, ByVal pError As
    ADODB.Error,adStatus As ADODB.EventStatusEnum,ByVal pRecordset As ADODB.Recordset)
    If IsUserOprt Then
        Call ShowDb
        Adodc1.Caption = "第" & pRecordset.AbsolutePosition & "/共" & _
                         pRecordset.RecordCount & "条记录"
    End If
End Sub
```

ADODC 控件的 MoveComplete 事件在移动数据库记录后触发，参数表中 pRecordset 表示移动记录指针后的当前记录。当用户在窗体中操作 ADODC 控件移动数据库记录指针

时触发该事件，在窗体中显示移动后的当前记录内容。

(8) 设计用户修改 ComboBox1 内容后的事件代码：

```
Private Sub ComboBox1_Click( )
    Select Case ComboBox1.Text
        Case "圆":
            Adodc1.RecordSource = "select * from circle"      '修改数据源
            Adodc1.Refresh          '刷新 ADODC 数据控件

            With Adodc1.Recordset
                Call ShowDb              '显示新数据源的数据库记录
                Label1.Caption = .Fields(0).Name     '修改数据字段名称
                Label2.Caption = .Fields(1).Name
                Label3.Caption = .Fields(2).Name
                Label4.Caption = .Fields(3).Name
                Label5.Visible = False   '由于圆对象信息比线对象少，在窗体暂时隐藏两个控件
                TextBox5.Visible = False
                Adodc1.Caption = "第" & .AbsolutePosition & "/共" & .RecordCount & "条记录"
            End With
        Case "线":
            Adodc1.RecordSource = "select * from Line"     '修改数据源
            Adodc1.Refresh              '刷新 ADODC 数据控件

            With Adodc1.Recordset
                Call ShowDb          '显示新数据源的数据库记录
                Label1.Caption = .Fields(0).Name     '修改数据字段名称
                Label2.Caption = .Fields(1).Name
                Label3.Caption = .Fields(2).Name
                Label4.Caption = .Fields(3).Name
                Label5.Visible = True  '由于线对象信息比圆对象多，在窗体中显示可能被隐藏的控件
                TextBox5.Visible = True
                Label5.Caption = .Fields(4).Name
                Adodc1.Caption = "第" & .AbsolutePosition & "/共" & .RecordCount & "条记录"
            End With
    End Select

    '由于用户修改了要操作的图形对象类型，ACAD 选择集需要更新，先隐藏与之相关的控件
    cmdSelFirst.Visible = False
    cmdSelPrev.Visible = False
    cmdSelNext.Visible = False
    cmdSelLast.Visible = False
    TextBox6.Visible = False
    cmdAdd2Db.Enabled = False
End Sub
```

(9) “添加图形”按钮用于将数据库中指定参数的图形添加到图形窗口中，其实现代

码为：

```
Private Sub cmdAdd2Acad_Click( )
    Dim pt1(0 To 2) As Double
    Dim pt2(0 To 2) As Double

    Select Case ComboBox1.Text
        Case "圆":
            With Adodc1.Recordset
                pt1(0) = .Fields("cenx")     '读取当前记录内容
                pt1(1) = .Fields("ceny")
                Dim rad As Double
                rad = .Fields("rad")

                ThisDrawing.ModelSpace.AddCircle pt1,rad     '在 ACAD 中添加圆对象
                ThisDrawing.Application.Update     '刷新 ACAD 窗口,显示添加的圆对象
            End With
        Case "线":
            With Adodc1.Recordset
                pt1(0) = .Fields("x1")
                pt1(1) = .Fields("y1")
                pt2(0) = .Fields("x2")
                pt2(1) = .Fields("y2")

                ThisDrawing.ModelSpace.AddLine pt1,pt2     '在 ACAD 中添加直线对象
                ThisDrawing.Application.Update     '刷新 ACAD 窗口,显示添加的直线对象
            End With
    End Select
End Sub
```

(10)“更新数据”按钮用于保存用户对数据库当前记录的修改，其实现代码为：

```
Private Sub cmdUpdate_Click( )
    With Adodc1.Recordset
        .Fields(0) = TextBox1.Text
        .Fields(1) = TextBox2.Text
        .Fields(2) = TextBox3.Text
        .Fields(3) = TextBox4.Text

        If ComboBox1.Text = "线" Then .Fields(4) = TextBox5.Text

        .Update
    End With
End Sub
```

(11)“退出”按钮用于删除选择集，结束程序的执行，其实现代码为：

```
Private Sub cmdExit_Click( )
```

```
    sel.Delete                                    '删除 ACAD 选择集对象
    Unload Me                                     '卸载窗体
End Sub
```

(12) 在“选择图形”按钮的单击事件中，添加与用户选择 ACAD 图形对象有关的控件事件代码：

```
Private Sub cmdSel_Click( )
    Dim gpCode(0) As Integer
    Dim dataValue(0) As Variant

    Select Case ComboBox1.Text     '根据用户要操作的图形类型设置选择集过滤条件
        Case "圆":
            gpCode(0) = 0
            dataValue(0) = "Circle"
        Case "线":
            gpCode(0) = 0
            dataValue(0) = "line"
    End Select

    Me.Hide     '暂时隐藏用户窗体，从 ACAD 窗口中选择对象

    sel.SelectOnScreen gpCode,dataValue     '用户从 ACAD 窗口中选择要操作类型的图形对象

    '根据用户选择集包含有效图形对象数目，设置某些控件的可见性和可操作属性
    If sel.Count > 0 Then              '若用户选择了有效的图形对象
        cndAdd2Db.Enabled = True       '允许用户向数据库添加新记录
        TextBox6.Visible = True        '使操作和显示 ACAD 选择集指针的控件可见
        cmdSelFirst.Visible = True
        cmdSelNext.Visible = True
        cmdSelPrev.Visible = True
        cmdSelLast.Visible = True
        cmdSelPrev.Enabled = False     '初始状态处于选择集中第一个对象，不允许向前移动指针
        cmdSelFirst.Enabled = False

        If sel.Count > 1 Then      '若选择集包含的有效图形对象多于一个，允许向后移动指针
            cmdSelNext.Enabled = True
            cmdSelLast.Enabled = True
        Else                       '若选择集仅包含一个有效图形对象，不允许移动指针
            cmdSelNext.Enabled = False
            cmdSelLast.Enabled = False
        End If

        Call ShowSel(0)      '在用户窗体中显示选择集中第一个有效图形对象
    End If
```

```
    Me.Show      '恢复用户窗体可见
End Sub
```

其中，ShowSel 过程在用户窗体中显示选择集对象中图形对象的信息，其实现代码为：

```
Private Sub ShowSel(i As Integer)     '变量 i 表示图形对象在选择集中从 0 开始的索引号
    Dim pt1,pt2
    TextBox6.Text = i + 1 & "/" & sel.Count

    Select Case ComboBox1.Text     '根据用户操作的对象类型显示图形信息
        Case "圆":
            TextBox1.Text = sel.Item(i).ObjectID
            pt1 = sel.Item(i).Center
            TextBox2.Text = pt1(0)
            TextBox3.Text = pt1(1)
            TextBox4.Text = sel.Item(i).Radius
        Case "线":
            pt1 = sel.Item(i).StartPoint
            pt2 = sel.Item(i).EndPoint
            TextBox1.Text = sel.Item(i).ObjectID
            TextBox2.Text = pt1(0)
            TextBox3.Text = pt1(1)
            TextBox4.Text = pt2(0)
            TextBox5.Text = pt2(1)
    End Select
End Sub
```

(13) “|<”、“<<”、“>>”和“>|”4 个按钮用于在选择集内的多个实体之间切换，“|<”按钮的实现代码为：

```
Private Sub cmdSelFirst_Click( )
    Call ShowSel(0)                    '在窗体中显示第一个图形对象信息
    cmdSelFirst.Enabled = False        '设置移动指针命令按钮的可操作属性
    cmdSelPrev.Enabled = False

    If sel.Count > 1 Then
        cmdSelNext.Enabled = True
        cmdSelLast.Enabled = True
    End If
End Sub
```

“>|”按钮用于定位到最后一个实体，其实现代码为：

```
Private Sub cmdSelLast_Click( )
    Call ShowSel(sel.Count-1)
    cmdSelNext.Enabled = False   '设置移动指针命令按钮的可操作属性
    cmdSelLast.Enabled = False
```

```
    If sel.Count > 1 Then
        cmdSelFirst.Enabled = True
        cmdSelPrev.Enabled = True
    End If
End Sub
```

“>>”按钮用于定位到下一个实体，其实现代码为：

```
Private Sub cmdSelNext_Click( )
    Dim i As Integer
    i = CInt(Mid(TextBox6.Text,2,1))  '从 TextBox6 控件中提取移动前的图形对象序号
    Call ShowSel(i)  '显示下一图形对象信息，序号从 1 开始，从而选择集索引号为 i
    cmdSelFirst.Enabled = True     '设置移动指针命令按钮的可操作属性
    cmdSelPrev.Enabled = True

    If sel.Count > i + 1 Then
        cmdSelNext.Enabled = True
        cmdSelLast.Enabled = True
    Else
        cmdSelNext.Enabled = False
        cmdSelLast.Enabled = False
    End If
End Sub
```

“<<”按钮用于返回上一个实体，其实现代码为：

```
Private Sub cmdSelPrev_Click( )
    Dim i As Integer
    i = CInt(Mid(TextBox6.Text,2,1))     '从 TextBox6 控件中提取移动前的图形对象序号
    Call ShowSel(i-2)     '显示上一图形对象信息，序号从 1 开始，从而选择集索引号为 i-2
    cmdSelNext.Enabled = True     '设置移动指针命令按钮的可操作属性
    cmdSelLast.Enabled = True

    If i > 2 Then
        cmdSelFirst.Enabled = True
        cmdSelPrev.Enabled = True
    Else
        cmdSelFirst.Enabled = False
        cmdSelPrev.Enabled = False
    End If
End Sub
```

(14) “添加数据”按钮用于将选择集中的当前图形对象信息添加到数据库中，其实现代码为：

```
Private Sub cmdAdd2Db_Click( )
    Dim HasRecord As Boolean     '该变量表示数据库中是否已经存在当前图形信息
    Dim i As Integer
```

```
    IsUserOprt = False      '以下在数据库中查找需移动指针,但不希望影响用户窗体显示

    With Adodc1.Recordset      '在数据库中查找有无当前图形信息
        .MoveFirst

        For i = 0 To .RecordCount-1
            If .Fields(0) = TextBox1.Text Then
                HasRecord = True
                Exit For
            End If

            .MoveNext
        Next i

        If HasRecord Then      '若数据库中已存在当前图形信息,仅更新图形记录内容
            Call cmdUpdate_Click
        Else
            .AddNew      '若数据库中没有当前图形信息,需在数据库中添加新记录
            .Fields(0) = TextBox1.Text
            .Fields(1) = TextBox2.Text
            .Fields(2) = TextBox3.Text
            .Fields(3) = TextBox4.Text

            If ComboBox1.Text = "线" Then .Fields(4) = TextBox5.Text

            .Update      '刷新数据库记录
            IsUserOprt = True      '移动数据库记录指针到新添加的记录时,改变窗体显示
            .MoveLast
        End If
    End With

    IsUserOprt = True
End Sub
```

(15) 选择“插入”|“模块”菜单项,向工程中添加一个标准模块,并在其中添加启动宏的代码:

```
Public Sub Main( )
    UserForm1.Show
End Sub
```

14.4.4　实例效果

(1) 在 VBA 集成开发环境中,执行 Main 宏,系统会弹出如图 14.8 所示的对话框。

(2) 单击“添加图形”按钮,能将当前指定的直线(圆)(下面的 ADODC 控件中显示了当

前指定的直线的序号)添加到图形窗口中。

(3) 单击"选择图形"按钮,可以在图形窗口中选择直线(圆),完成选择之后,系统会在对话框中显示选择的结果,如图 14.9 所示。"图形信息"选项区域的下方显示了当前选择集中包含直线(圆)的数量,可以单击 4 个导航按钮在各个实体之间切换。在这种状态下,可以单击"添加数据"按钮将指定索引号的实体的参数添加到数据库中,以便在其他图形中创建该实体。

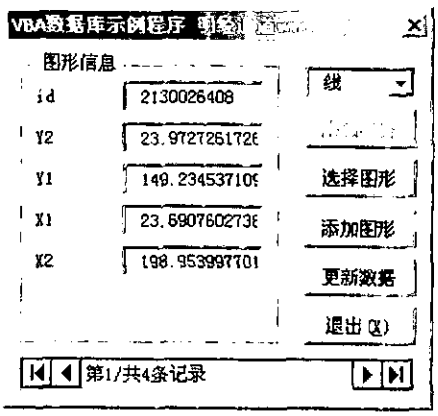


图 14.8 程序运行主界面

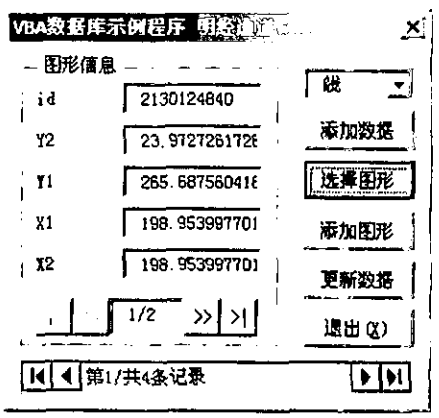


图 14.9 显示选择集添加实体的结果

14.4.5 小结

本节是一个比较综合的实例,其中涉及到了数据库的连接、数据表的读取、添加记录等内容,读者可以自行扩展该程序所支持实体(目前仅支持直线和圆)的种类,并实现一些数据库搜索的操作。

14.5 数据记录和 AutoCAD 实体的对应

14.5.1 实例说明

GIS 是目前比较热门的一个行业,而 GIS 与 CAD 又有着非常密切的关系。有人说,GIS=CAD+MIS,这个式子很形象地揭示了 GIS 和 CAD 的关系。

GIS 中一个基本的要求就是地理数据和属性的关联,在 AutoCAD 二次开发的程序中,有时也需要这种关联,本节的实例就创建了一个简单的 GIS 程序,用户可以添加和修改实体的属性数据。

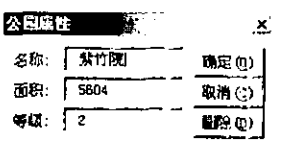


图 14.10 显示公园的属性

例如,图形中"公园"图层中的闭合多段线代表了公园的外轮廓,可以为其添加公园名称、面积和等级等属性,如图 14.10 所示。

14.5.2　编程思路

1. ObjectID、Handle 和对象本身

学习 VBA 编程的很长时间，我们都一直在和对象本身打交道，通过对象获得它的各种属性，例如，objCircle. Center 能够获得圆心的坐标。

在 VBA 编程中，用到 ObjectID 的机会不多，但是有必要了解一下。在 AutoCAD 的一个进程中，会为所有的对象分配一个惟一的 ObjectID，在一次 AutoCAD 的任务中，所有的 ObjectID 都不相同。换而言之，对于同一对象，这一次和下一次在 AutoCAD 中的 ObjectID 可能是不一样的。

Handle(句柄)是 Windows 编程中一个常用的概念，究竟什么是句柄呢？句柄是一个长整型的数，用于管理大量的同类对象，例如，在学校里用学生证号码来区分每一个学生，国家用身份证号码来区别每一个公民。AutoCAD 用句柄来区别同一个图形中的不同对象，同一图形中所有对象的句柄都不相同，但是不同图形中的对象有可能拥有相同的句柄。

从对象可以通过 Handle 属性来获得其句柄，使用 ObjectID 属性来获得对象的 ID 号。反过来，可以使用 HandleToObject 方法从句柄获得一个对象，使用 ObjectIDToObject 方法从 ID 号获得对应的对象。

2. 记录和实体对应的惟一性

数据库中的记录通过设置主键可以保证其惟一性，AutoCAD 中实体的惟一性则可以通过图形名称和句柄来共同确定(ObjectID 在不同的 AutoCAD 任务中是可变的，无法作为永久的关联方式)。

本节的实例中，为数据表设置了两个主键 DwgName 和 EntHandle，一起用于确定一个实体，这样就确保了数据库中记录和 AutoCAD 中实体的一一对应。

14.5.3　步骤

(1) 在 AutoCAD 2005 中，打开 VBA 管理器，创建一个新工程，保存在适当的位置。进入 VBA 集成开发环境，选择“插入”|“模块”菜单项，向工程中添加一个标准模块，将其名称修改为 mdlPubVariable，并在其中添加下面的全局变量：

```
Public addMode As Boolean          '显示属性时是否是添加模式

Public strDbName As String         '数据库的名称，在 frmConnectDB 中赋值
Public adoCon As New ADODB.Connection        '所有的记录集共用一个数据库连接对象

Public entHandle As String         '当前选择的图形的句柄
Public area As Double              '当前选择的多段线的面积
Public length As Double            '当前选择的多段线的长度
```

这些全局变量都有一个特点，需要同时在启动宏和对话框中使用，因此只能将其定义成全局变量。在设计该程序的过程中，这些全局变量是根据需要逐个添加的。

(2) 选择“工具”|“引用”菜单项，在弹出的对话框中选中 *Microsoft Visual Basic for*

Applications Extensibility 5.3 和 Microsoft ActiveX Data Objects 2.5 Library，如图 14.11 所示。

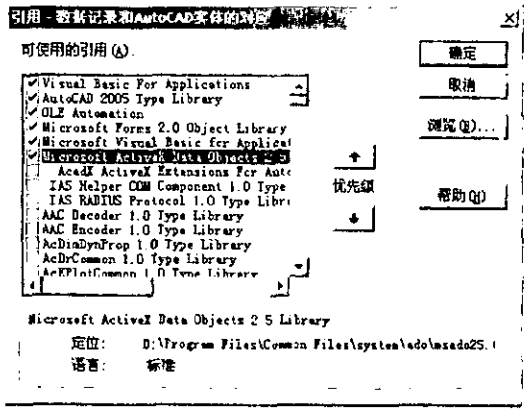


图 14.11 引用类型库

(3) 选择“插入”|“模块”菜单项，插入一个标准模块，将其名称修改为 mdlConnectDB，并在其中添加用于连接和关闭数据库的代码：

```
'功能：打开指定的数据库(在 frmConnectDB 中指定)
'输入：无
'调用：无
'返回：如果完成连接，返回 True；否则返回 False
'示例：OpenDB
Public Function OpenDB( ) As Boolean
    OpenDB = True

    '如果数据库已打开，不执行任何操作
    If adoCon.State <> 0 Then Exit Function

    adoCon.CursorLocation = adUseClient

    '获得数据库文件的位置
    Dim strDbName As String
    Dim strProject As String
    strProject = Left(ThisDrawing.Application.VBE.activevbProject.FileName,_
                      Len(ThisDrawing.Application.VBE.activevbProject.FileName)-21)
    strDbName = strProject & "Attribute.mdb"

    adoCon.Open "Provider = Microsoft.Jet.OLEDB.4.0;Data Source = " & _
        strDbName & ";"
End Function

'功能：关闭指定的数据库(在 frmConnectDB 中指定)
'输入：无
'调用：无
'返回：如果数据库处于打开状态，就关闭它
```

```
'示例：CloseDB
Public Function CloseDB( )
    If adoCon.State <> 0 Then
        adoCon.Close
    End If
End Function
```

(4) 双击“工程资源管理器”窗口中的 ThisDrawing 图标，打开其代码窗口，在其中添加一个过程 FindObject，用于查找数据表中是否已经包含指定的实体，并根据查询结果设置变量 addMode 的值，以此来决定添加还是修改选中的实体的属性。其实现代码为：

```
Private Sub FindObject(ByVal strDwgName As String,ByVal entHandle As String,ByVal strTable As String)
    mdlPubVariable.addMode = False

    Dim rs As New ADODB.Recordset
    rs.Open "SELECT EntHandle FROM " & strTable & " WHERE DwgName = '" & strDwgName & "' AND
        EntHandle = '" & entHandle & "';",_
            adoCon,adOpenDynamic,adLockOptimistic

    If rs.RecordCount <= 0 Then
        mdlPubVariable.addMode = True
    End If

    rs.Close
End Sub
```

(5) 在 ThisDrawing 模块中，添加一个宏 AddAttribute，用于为图形中的实体添加或修改属性，其中包含了 GetEntity 方法的错误处理，相关代码为：

```
Private Const VK_ESCAPE = &H1B      '代表 Esc 键
Private Declare Function GetAsyncKeyState Lib "user32" (ByVal vKey As Long) As Integer

'功能：判断用户是否按下某一个键
'输入：代表键的常量(从 API Viewer 中获得)
'调用：API 函数 GetAsyncKeyState
'返回：如果用户按下了指定的键，返回 True；否则返回 False
'示例：If CheckKey(&H1B) = True Then do sth
Private Function CheckKey(lngKey As Long) As Boolean
If GetAsyncKeyState(lngKey) Then
    CheckKey = True
Else
    CheckKey = False
End If
End Function

'为图形中的实体添加属性
Public Sub AddAttribute( )
    '连接数据库
```

```
    If mdlConnectDB.OpenDB = False Then
        Exit Sub
    End If

    Dim objSelect As AcadEntity
    Dim basePnt As Variant
    '选择对象并判断类型
    On Error Resume Next

Retry:
    ThisDrawing.Utility.GetEntity objSelect,basePnt,vbCrLf & "选择所要添加属性的对象："

    '处理按下 Esc 键的错误
    If objSelect Is Nothing Then
        If CheckKey(VK_ESCAPE) = True Then
            Exit Sub
        Else
            GoTo Retry
        End If
    End If

    '处理未选择到实体的错误
    If Err <> 0 Then
        Err.Clear
        GoTo Retry
    End If

    '只对轻量多段线进行处理
    If Not TypeOf objSelect Is AcadLWPolyline Then Exit Sub

    '保存当前选择对象的句柄
    mdlPubVariable.entHandle = objSelect.Handle

    '获得多段线的长度和面积
    mdlPubVariable.area = objSelect.area
    mdlPubVariable.length = objSelect.length

    '根据图层来进行不同的处理
    Select Case objSelect.Layer
        Case "公园"
            Call FindObject(ThisDrawing.FullName,mdlPubVariable.entHandle,"Park")
            frmPark.Show

        Case "学校"
            Call FindObject(ThisDrawing.FullName,mdlPubVariable.entHandle,"School")
            frmSchool.Show
```

```
        Case "街道"
            Call FindObject(ThisDrawing.FullName,mdlPubVariable.entHandle,"Road")
            frmRoad.Show
    End Select

    '关闭数据库的连接
    CloseDB
End Sub
```

用户选择一个实体后，如果该实体不是多段线，便退出程序，不对其进行操作。如果选择了轻量多段线，并且其在几个特定的图层上，在相应的数据表中查找该实体的属性，如果找到便弹出一个对话框显示其属性，如果未找到其属性便弹出一个对话框供用户添加属性。

(6) 选择“插入”|“用户窗体”菜单项，向工程中添加一个对话框，名称修改为 frmPark，并在其中放置如图 14.12 所示的控件。其中，三个文本框的名称分别修改为 txtName、txtArea 和 txtClass，三个命令按钮的名称修改为 cmdOk、cmdCancel 和 cmdDelete。

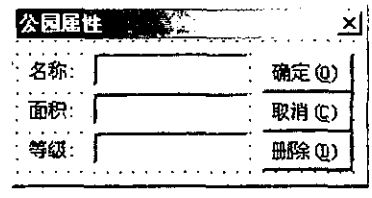


图 14.12　设计对话框

(7) 在窗体初始化的事件中，判断当前操作的模式，如果该实体已经包含了属性，就在弹出的对话框中显示其属性，否则仅计算其面积显示出来，其他属性可供用户添加。其相关代码为：

```
Private Sub UserForm_Initialize( )
    '自动计算面积
    txtArea.Text = mdlPubVariable.area

    '添加模式
    If mdlPubVariable.addMode = True Then
        cmdDelete.Enabled = False

    ElseIf mdlPubVariable.addMode = False Then        '修改模式
        '防止 mapID 和 eleID 未被初始化
        If Len(mdlPubVariable.entHandle) = 0 Then
            MsgBox "entHandle 未被初始化!检查对话框调用之前的语句。",vbCritical
            Exit Sub
        End If

        '显示数据库中指定的记录
        Dim adoRs As New ADODB.Recordset
        'Park 中的记录
        adoRs.Open "SELECT * FROM Park " & _
                   "WHERE DwgName = '" & ThisDrawing.FullName & "' AND EntHandle = '" & _
                   mdlPubVariable.entHandle & "';",adoCon,adOpenDynamic,adLockOptimistic

        If adoRs.RecordCount <= 0 Then
            MsgBox "未找到相应的记录!",vbCritical
            Exit Sub
```

```
        End If

        adoRs.MoveFirst

        txtName.Text = adoRs.Fields("ParkName")
        txtArea.Text = adoRs.Fields("Area")
        txtClass.Text = adoRs.Fields("Rank")

        If adoRs.State <> 0 Then adoRs.Close
    End If
End Sub
```

(8) 为了防止用户在文本框中输入无效的内容，分别对其 KeyPress 和 BeforeUpdate 事件进行处理，其结果是“面积”文本框仅能输入数字和小数点，“等级”文本框仅能输入 1、2 和 3 三个数字。为了进行这些限制，使用了下面的代码：

```
Private Sub txtArea_KeyPress(ByVal KeyAscii As MSForms.ReturnInteger)
    If (KeyAscii < Asc("0") Or KeyAscii > Asc("9")) And KeyAscii <> Asc(".") Then
        KeyAscii = 0
    End If
End Sub

Private Sub txtClass_BeforeUpdate(ByVal Cancel As MSForms.ReturnBoolean)
    If CInt(txtClass.Text) < 1 Or CInt(txtClass.Text) > 3 Then
        MsgBox "等级取值范围：1～3。"
        txtClass.Text = ""
    End If
End Sub

Private Sub txtClass_KeyPress(ByVal KeyAscii As MSForms.ReturnInteger)
    If (KeyAscii < Asc("0") Or KeyAscii > Asc("9")) Then
        KeyAscii = 0
    End If
End Sub
```

(9) 根据不同的操作模式，“确定”按钮实现了添加属性和修改属性的操作，其相关代码为：

```
Private Sub cmdOk_Click( )
    '确保所有的文本框不为空
    Dim itemControl As Control
    For Each itemControl In Me.Controls
        If StrComp(itemControl.Name,"txtRemark") <> 0 Then
            If TypeOf itemControl Is TextBox Then
                If Len(itemControl.Text) < = 0 Then
                    MsgBox "不能空缺参数，请继续输入。",vbInformation
                    Exit Sub
                End If
            End If
        End If
```

```
    Next itemControl

    Dim cmd As New ADODB.Command
    Set cmd.ActiveConnection = adoCon

    If mdlPubVariable.addMode = True Then
        '向 Park 表中添加记录
        cmd.CommandText = "INSERT INTO Park(DwgName,EntHandle,ParkName,Area,Rank) " & _
                          "VALUES('" & ThisDrawing.FullName & "','" & mdlPubVariable.
                              entHandle & "','" & _
                          txtName.Text & "'," & txtArea.Text & "," & txtClass.Text & ")"
        cmd.Execute

    ElseIf mdlPubVariable.addMode = False Then
        '修改 Park 表中的记录
        cmd.CommandText = "UPDATE Park SET ParkName = '" & txtName.Text & "',Area = " & _
                          txtArea.Text & ",Rank = " & txtClass.Text & " WHERE DwgName = '" & _
                          ThisDrawing.FullName & "'AND EntHandle = '" & mdlPubVariable.entHandle
                              & "';"
        cmd.Execute

    End If

    Unload Me
End Sub
```

(10) 如果是在修改属性的模式,“删除”按钮用于删除实体上已经包含的属性,该按钮单击事件的实现代码为:

```
Private Sub cmdDelete_Click( )
    Dim cmd As New ADODB.Command
    Set cmd.ActiveConnection = adoCon

    If mdlPubVariable.addMode = False Then
        '删除这条记录
        cmd.CommandText = "DELETE FROM Park WHERE DwgName = '" & ThisDrawing.FullName & "'AND
            EntHandle = '" & _
            mdlPubVariable.entHandle & "';"
        cmd.Execute
    End If

    Unload Me
End Sub
```

(11) “取消”按钮不对属性进行任何操作,其单击事件的处理代码为:

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

(12) 按照类似的方法添加 frmSchool 和 frmRoad 对话框，并对其中的控件进行适当的事件处理。由于另外两个对话框代码的总体结构与 frmPark 对话框相同，这里不再浪费宝贵的篇幅来列举另外两个对话框的代码，读者可以从配套光盘中获取完整的代码。

14.5.4 实例效果

(1) 在 AutoCAD 2005 中，打开配套光盘中本节的文件夹下的 Demo. dwg 文件。该图形文件中包含 3 个新建的图层“公园”、“学校”和“街道”，并在其中添加了若干多段线，如图 14.13 所示。

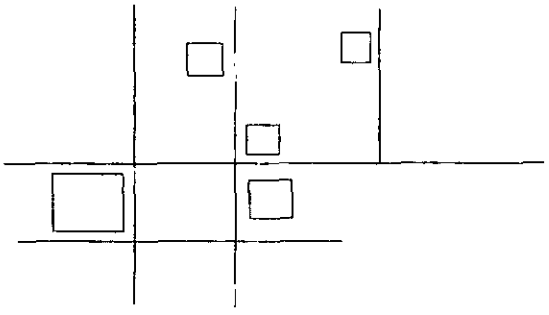


图 14.13 测试图形

(2) VBA 集成开发环境中，执行 AddAttribute 宏，在图形窗口中选择一个图形对象，如果该实体位于程序所要处理的 3 个层上，并且并未包含任何的属性，就会弹出如图 14.14 所示的对话框。可以在其中输入街道的名称和等级，可以修改自动计算的街道长度，然后单击“确定”按钮完成属性的保存。

(3) 再次执行 AddAttribute 宏，在图形窗口中选择一个图形对象，如果该实体位于程序所要处理的 3 个层上，并且已经包含某些属性，就会弹出如图 14.15 所示的对话框。修改其属性内容之后，可以单击“确定”按钮保存属性内容，或者单击“删除”按钮从数据库中删除此属性。

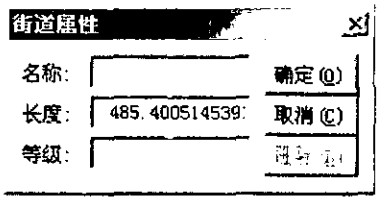


图 14.14 添加属性

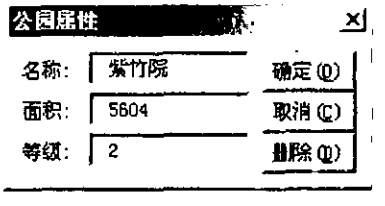


图 14.15 修改属性的内容

14.5.5 小结

通过本节所介绍的实例，读者可以掌握在 AutoCAD 系统上扩展 GIS 功能一个最基本的问题，就是实体和属性之间的关联的实现。在此基础上，还可以添加一些更复杂的内容，例如属性内容的批量查看，根据属性内容在图形中定位实体等。

第 15 章　VBA 程序的发布

程序的发布对于一个成熟的软件来说非常重要，对于 AutoCAD 二次开发的应用程序更是如此；AutoCAD 二次开发的应用程序发布比一般的应用程序困难，但这方面的知识在国内其他的图书中根本就没有见到过，而且很多商业软件的发布方式也很不规范。

“菜单宏”和在“工具栏中使用自定义图标”两节首先介绍了菜单栏和工具栏的创建方法，“基本图库的实现”部分则提供了动态修改菜单的一种技术。

在命令行或菜单中执行 VBA 程序，无疑是一个大的进步，这让 VBA 程序在执行的方便性上不再比 Visual LISP 和 ObjectARX“低人一等”（无法直接定义命令）；先选择后操作的编辑模式在 AutoCAD 中十分常见，但是 VBA 中特殊的实现方式导致无法在菜单中实现先选择后操作，本章专门讨论了这个问题，提出了合理的解决方案。

更改 AutoCAD 应用程序的标题栏和图标会让二次开发的程序显得更专业，使用 InstallShield 工具来发布 VBA 应用程序，提供了专业和完善的程序发布模板，直接仿照书中的程序，短时间之内就可以完成自己的程序打包发布！

15.1　菜　单　宏

15.1.1　实例说明

本节的实例创建了一个可在 AutoCAD 中加载的菜单，该菜单使用了多种菜单宏的语法，包括在菜单中调用 VBA 的宏、打开外部的应用程序，以及显示帮助文件。

15.1.2　编程思路

菜单宏用于指定某个菜单项所要执行的任务，可以在菜单宏中调用一个 AutoCAD 内部命令，和用户交互，或者完成其他任务，下面是一个典型的菜单宏示例（用于绘制一个半径为 1 的圆）：

```
Circle-1 [Radius-1]^C^C_circle \1
```

- 特殊字符^C^C 用于取消所有正在运行的命令。
- 特殊字符下划线（_）自动将其后的命令翻译为其他语言。
- 条目 *circle* 用于启动 CIRCLE 命令。
- 特殊字符反斜杠（\）用于创建停顿，以便用户指定中心点。
- 条目 1 用于响应半径提示。

关于菜单宏的详细语法，可以参考“AutoCAD 2005 帮助：开发人员文档”中“自定义手册/菜单文件/创建菜单宏”部分的内容。

15.1.3 步骤

(1) 打开记事 7 本应用程序,在其中添加菜单文件的基本框架:

```
//菜单组名称
*** MENUGROUP = MenuMacro

//菜单
*** POP1
** MnuMac

ID_MN [菜单宏(&M)]
```

最后一项代表当前的菜单的标题。

(2) 添加一个基本的菜单项,在菜单宏中调用 AutoCAD 内部命令 LINE:

```
ID_MNLine [直线(&L)]^C^C_line;
```

菜单宏中的空格字符相当于在 AutoCAD 中按下空格键,为了显示的效果更好,全部用分号(相当于按下回车键)来代替。

(3) 添加一个菜单项,调用 CIRCLE 命令,提示用户输入圆心,创建半径为 1 的圆:

```
ID_MNCircle [半径为 1 的圆(&C)]^C^C_circle;\1
```

(4) 添加一个菜单项,使用以 *^C^C 开头的菜单宏,能够重复执行 POINT 命令,直到用户按下 Esc 键为止:

```
ID_MNPoint [连续创建点(&P)] *^C^C_point;
```

(5) 添加一个菜单项,在系统提示选择对象时使用 Single 关键字,只允许用户选择一个实体,并且将其颜色修改为红色:

```
ID_MNChangeColor [修改一个实体的颜色(&M)]^C^C_change;single;\p;c;red ;
```

(6) 添加一个菜单项,用于打开外部应用程序记事本:

```
ID_MNLaunchExe [打开记事本(&N)]^c^c(startapp "notepad.exe");
```

(7) 添加一个菜单项,用于调用一个 DVB 文件中的宏(如果宏在标准模块中,需要指定模块的名称):

```
ID_MNMacro [执行 VBA 宏(&V)]^C^C-vbarun;c:/Line.dvb!AddLine;
```

(8) 添加一个菜单项,用于显示 AutoCAD 中的帮助文件 acadauto.chm:

```
ID_MNHelp [帮助(&H)]^C^C^P(help "acadauto" "") ^P
```

为了防止该菜单项对其他项产生一些未知的影响,可以将其放在菜单的最后。

(9) 最后,将文件保存为 MenuMacro.mnu。

15.1.4 实例效果

(1) 在 AutoCAD 2005 中,选择“工具”|“自定义”|“菜单”菜单项,系统会弹出如图 15.1 所示的“菜单自定义”对话框。

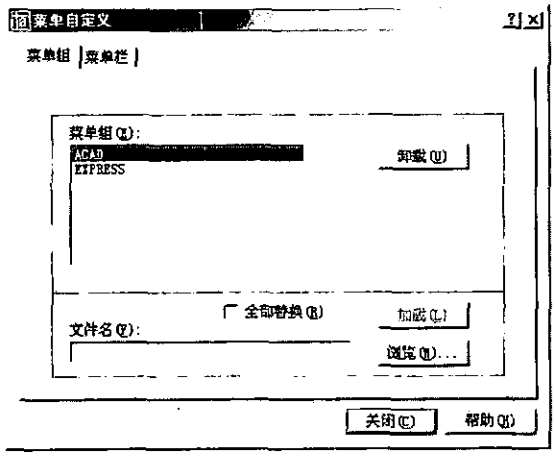


图 15.1　显示“菜单自定义”对话框

(2) 单击“浏览”按钮，在弹出的“选择菜单文件”对话框中选择 MenuMacro. mnu 文件，单击“打开”按钮，返回“菜单自定义”对话框。单击“加载”按钮，系统会弹出如图 15.2 所示的提示对话框，单击“是”按钮，继续加载菜单文件。

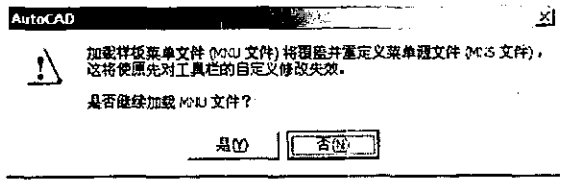


图 15.2　继续加载 MNU 菜单文件

(3) 切换到“菜单自定义”对话框的“菜单栏”选项卡中，从“菜单组”组合框中选择 MenuMacro 选项，在“菜单”列表框中选择“菜单宏”选项，在右侧的“菜单栏”列表框中选择“帮助”选项，单击“插入”按钮，将其插入到“帮助”菜单的前面，如图 15.3 所示。

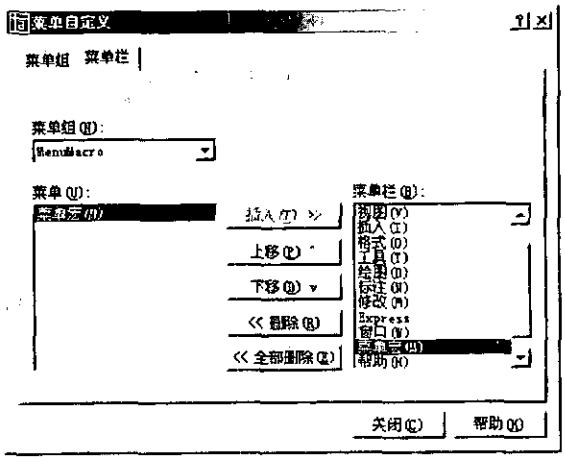


图 15.3　插入菜单栏

(4) 单击“关闭”按钮，关闭“菜单自定义”对话框，在 AutoCAD 的下拉菜单中就出现了“菜单宏”菜单栏，如图 15.4 所示。

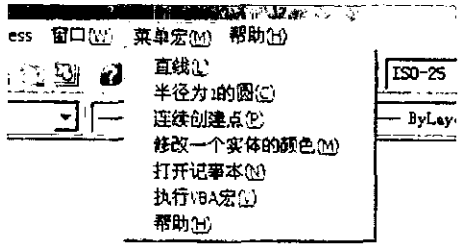


图 15.4 显示的“菜单宏”菜单栏

(5) 选择“菜单宏”|“直线”菜单项，系统会执行 LINE 命令。

(6) 选择“菜单宏”|“半径为 1 的圆”菜单项，系统会提示用户输入圆心，指定圆心的位置后，创建一个半径为 1 的圆。

(7) 选择“菜单宏”|“连续创建点”菜单项，系统会提示用户输入点的位置，创建一个点实体，并且在完成命令后继续提示用户输入点的位置，直到用户按下 Esc 键为止。

(8) 选择“菜单宏”|“修改一个实体的颜色”菜单项，系统会提示用户选择图形对象，在图形窗口中选择一个实体之后，系统会将该实体的颜色变为红色。

(9) 选择“菜单宏”|“打开记事本”菜单项，系统会自动打开记事本应用程序。

(10) 选择“菜单宏”|“执行 VBA 宏”菜单项，系统会自动加载 C: \Line. dvb 文件，并且执行其中的 AddLine 宏。

(11) 选择“菜单宏”|“帮助”菜单项，系统会打开 AutoCAD ActiveX and VBA reference 帮助文件。

15.1.5 小结

菜单和工具栏是用户调用某个功能的最友好的手段，利用本节介绍的知识，可以创建自定义的菜单文件，在其中自由调用外部程序、帮助文件和 VBA 中创建的宏，并且可以修改命令的实现方式。值得一提的是，工具栏中的按钮同样使用菜单宏来实现一定的功能，本节介绍的内容完全适用于工具栏。

15.2 在工具栏中使用自定义图标

15.2.1 实例说明

在发布应用程序时，一般都会创建几个工具栏和菜单来调用应用程序的功能，在创建工具栏时使用自定义的图标就很有必要。本节介绍的程序，将自定义的图标保存在一个 DLL 中，然后从菜单文件中为每个工具按钮指定图标，从而在工具栏中显示自定义的图标，如图 15.5所示。

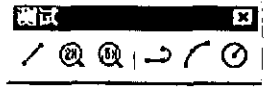


图 15.5 在工具栏中显示自定义图标

15.2.2 编程思路

在 AutoCAD 中，可以在菜单文件中定义工具栏，定义一个工具栏的语法为：

```
TAG1 [Toolbar ("tbarname",orient,visible,xval,yval,rows)]
```

其中，TAG1 是工具栏的标识符；tbarname 是命名工具栏的字符串，除了连字符(-)和下划线(_)之外不能使用其他的符号；orient 指定工具栏的方向，包括 floating、top、bottom、left 和 right；visible 指定工具栏的可见性，取值可为 show 和 hide；xval 指定从屏幕左端到工具栏左端的距离，以像素为单位；yval 指定从屏幕顶端到工具栏顶端的距离，以像素为单位；rows 指定工具栏的行数。

定义一个工具栏按钮的语法为：

```
TAG2 [Button ("btnname",id_small,id_large)]macro
```

其中，TAG2 是工具按钮的标识符；btnname 是命名按钮的字符串，该字符串会成为按钮的提示字符串；id_small 指定了小图像资源(16×15)的 ID 字符串；id_large 指定了大图像资源(32×30)的 ID 字符串；macro 是菜单宏，定义了该工具按钮所要执行的动作。从 AutoCAD 2004 开始，可以不指定大图像资源，而用小图像资源来代替，在需要大图标的场合，系统会自动将小图标拉伸来获得大图标。

下面是 acad. mnu 菜单文件中缩放工具栏的开始几行：

```
** TB_ZOOM
ID_TbZoom [_Toolbar("Zoom",_Floating,_Hide,100,380,1)]
ID_ZoomWindo [Button("Zoom Window",ICON_16_ZOOWIN,ICON_16_ZOOWIN,)]'_zoom_w
ID_ZoomDynam [Button("Zoom Dynamic",ICON_16_ZOODYN,ICON_16_ZOODYN,)]'_zoom_d
ID_ZoomScale [Button("Zoom Scale",ICON_16_ZOOSCA,ICON_16_ZOOSCA,)]'_zoom_s
```

要在工具栏中使用自定义的图标，可以创建一个与菜单文件同名的 DLL，将需要使用的图标封装在 DLL 中，然后在菜单文件中指定所要使用的图标 ID 字符串即可。

15.2.3　步骤

(1) 运行 Visual Basic 6.0，在系统弹出的 New Project 对话框中，选择 ActiveX DLL 项目，如图 15.6 所示，单击“打开”按钮。

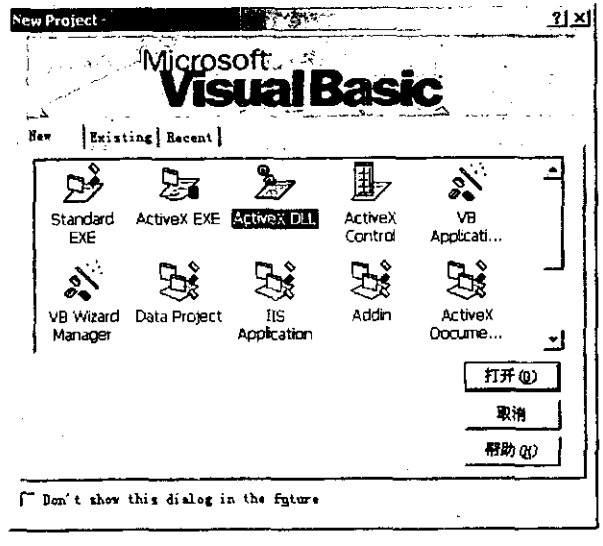


图 15.6　选择 ActiveX DLL 项目

(2) 在“工程浏览器”窗口中,右击 Project1 节点,从弹出的快捷菜单中选择 Project1 Properties 菜单项,系统会弹出如图 15.7 所示的 Project1 Properties 对话框。将 Project Name 文本框中的名称修改为 mjtd(该名称必须与所要使用自定义图标的菜单文件同名,这样在后面可以省却一些操作),单击“确定”按钮关闭该对话框。

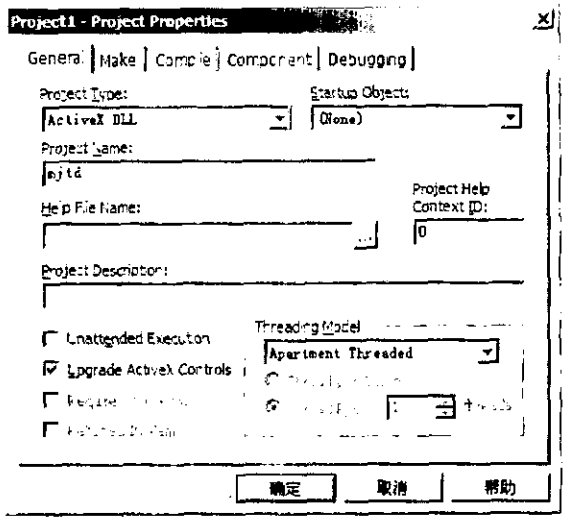


图 15.7 修改工程名称

(3) 选择“Add-Ins”|“Add-In Manager”菜单项,系统会弹出如图 15.8 所示的 Add-In Manager 对话框。从可用的外接程序列表中选择 VB 6 Resource Editor 选项,在 Load Behavior 选项区域中选择 Loaded/Unloaded 复选框,将该程序加载到 VB 6.0 中,单击 OK 按钮。

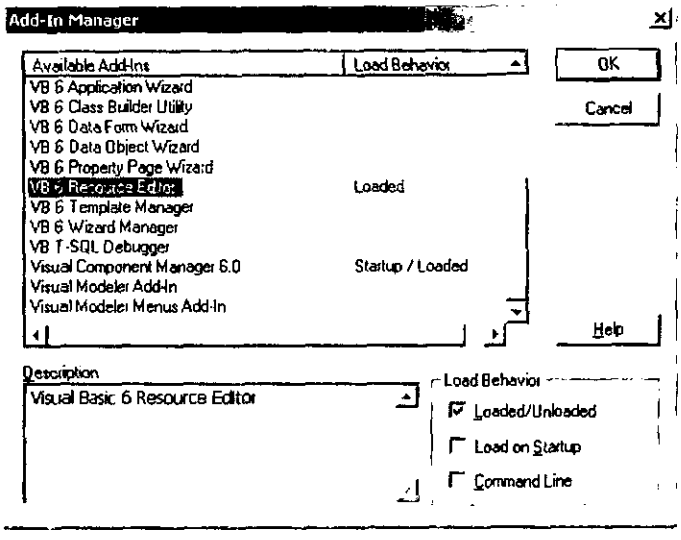


图 15.8 加载资源编辑器

(4) 选择 Tools|Resource Editor 菜单项,系统会弹出如图 15.9 所示的 VB Resource Editor 窗口。

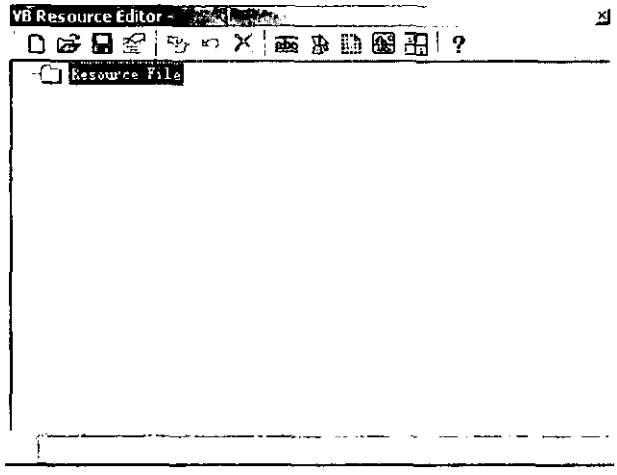


图 15.9　打开资源编辑器

(5) 修改资源位图的尺寸。在“Windows 资源管理器”窗口中，用 Microsoft Photo Editor 应用程序打开所要使用的位图，如图 15.10 所示。如果未安装该程序，可以使用 PhotoShop、ACDSee 这些常用的软件来完成同样的操作。注意，配套光盘中提供的图标文件已经修改成 16×15 像素，可以直接使用。

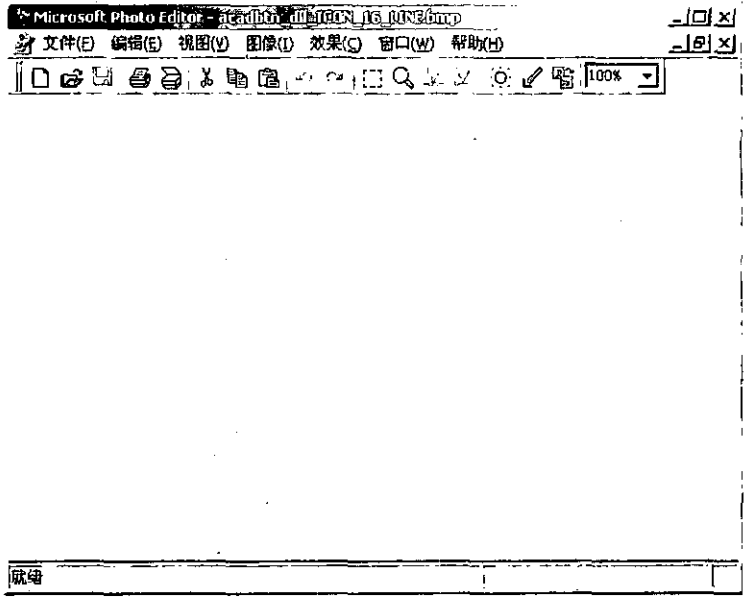


图 15.10　打开位图

(6) 在 Microsoft Photo Editor 中，选择“图像”|“调整尺寸”菜单项，系统会弹出如图 15.11 所示的“调整尺寸”对话框。从“单位”下拉列表框中选择“像素”选项，选中“允许失真”复选框，将位图的大小调整为 16×15 像素。单击“确定”按钮。

(7) 在 Microsoft Photo Editor 中，保存对位图的修改。使用同样的方法，修改在 DLL 中所要使用的其他几个位图。

(8) 切换到 Visual Basic 6.0 中，在“资源编辑器”窗口中单击 Add Bitmap 按钮，系统会弹出如图 15.12 所示的 Open A Bitmap File 对话框。选择所要添加到 DLL 中的多个位图，单击“打开”按钮。

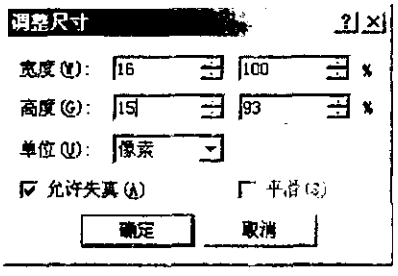


图 15.11 调整位图尺寸

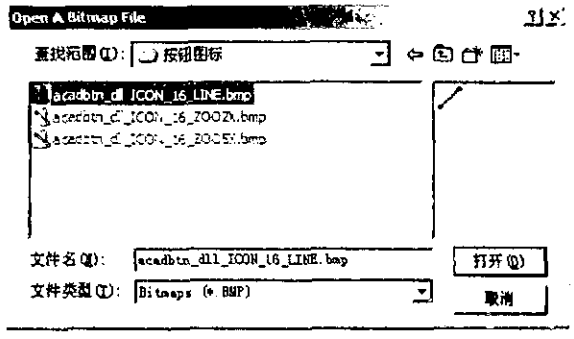


图 15.12 选择所要加载的位图

(9) 返回“资源编辑器”窗口，添加的 3 个位图被自动命名为 101、102 和 103。在一个图标上双击，系统弹出如图 15.13 所示的 Edit Bitmap Properties 对话框。在 Id 文本框中输入 ZOOMOUT 作为新的 Id，单击 OK 按钮。

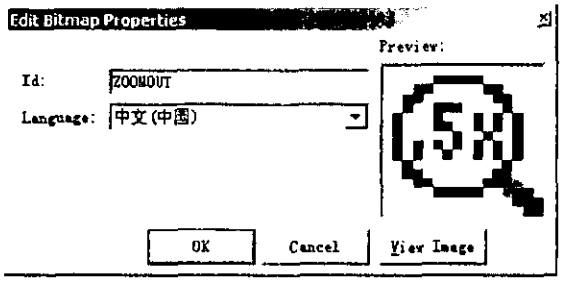


图 15.13 修改位图的 Id

(10) 使用同样的方法，将其他两个位图的 Id 修改为 ZOOMIN 和 LINE。在“资源编辑器”窗口中，得到如图 15.14 所示的结果。

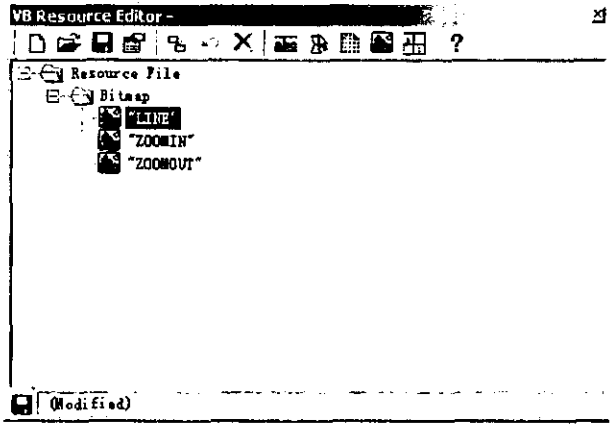


图 15.14 编辑位图 Id 之后的资源编辑器

(11) 在“资源编辑器”窗口中单击保存按钮，将资源文件与程序文件保存在同一文件夹中。关闭“资源编辑器”窗口，选择 File|Make mjtd.dll 菜单项，系统会弹出如图 15.15 所示的 Make Project 对话框，指定适当的保存位置，单击 OK 按钮。

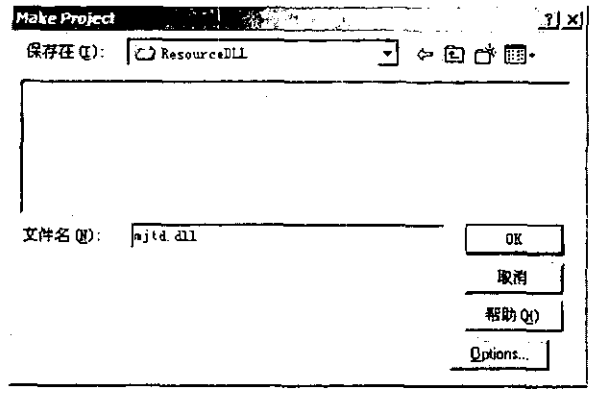


图 15.15　指定输出 DLL 的位置

至此为止，资源 DLL 的创建已经结束，下面的操作是修改 AutoCAD 的菜单文件。

(12) 用记事本新建一个文件，在其中添加如下的定义代码：

```
//菜单组名称
*** MENUGROUP = mjtd

//工具栏
*** TOOLBARS

** TB_TEST
** 测试
ID_TbTest     [_Toolbar("测试",_Floating,_Show,100,240,1)]
ID_Test01     [_Button("直线","LINE","LINE")]^C^C_line
ID_Test02     [_Button("放大","ZOOMIN","RZOOMIN")]^C^C_zoom;2x;
ID_Test03     [_Button("缩小","ZOOMOUT","ZOOMOUT")]^C^C_zoom;0.5x;
```

(13) 步骤(12)用于创建自定义图标的工具栏，还可以使用系统已经提供的图标。下面的代码使用了三个系统已经存在的图标：

```
              [--]
ID_Test04     [_Button("多段线","RCDATA_16_PLINE","RCDATA_16_PLINE")]^C^C_pline
ID_Test04     [_Button("圆弧","RCDATA_16_ARC3PT","RCDATA_16_ARC3PT")]^C^C_arc
ID_Test04     [_Button("圆","RCDATA_16_CIRRAD","RCDATA_16_CIRRAD")]^C^C_circle
```

其中[--]创建了一个分隔符。

(14) 在记事本应用程序中选择“文件”|“保存”菜单项，系统会弹出如图 15.16 所示的“另存为”对话框，在其中输入 mjtd.mnu 为文件名，选择“保存类型”为“所有文件”，单击“保存”按钮，完成菜单文件的保存。

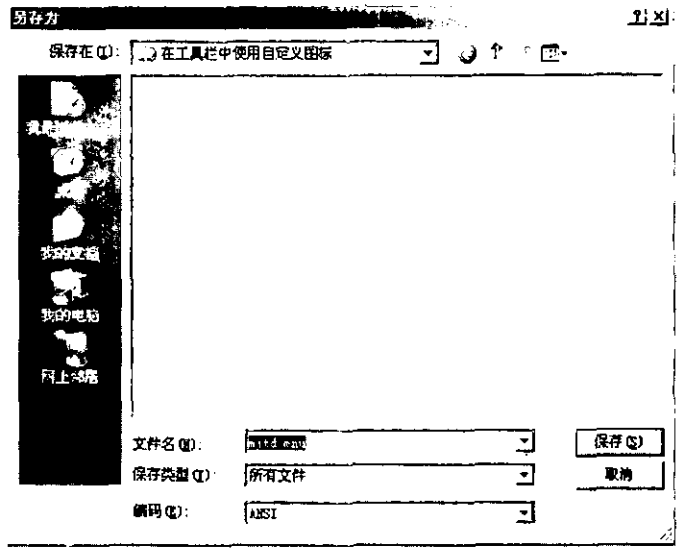


图 15.16 保存菜单文件

15.2.4 实例效果

(1) 将 mjtd.dll 复制到 mjtd.mnu 所在的目录下，在 AutoCAD 2005 中选择“工具”|“自定义”|“菜单”菜单项，系统会弹出如图 15.17 所示的“菜单自定义”对话框。

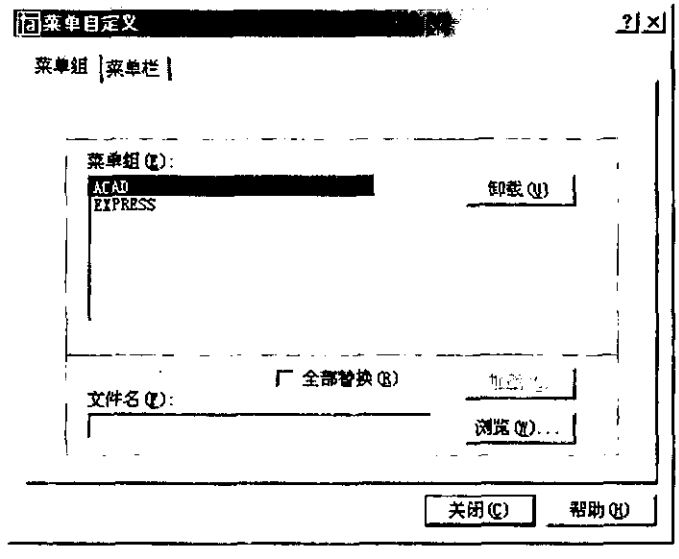


图 15.17 打开“菜单自定义”对话框

(2) 单击“浏览”按钮，在系统弹出的“选择菜单文件”对话框中，选择所要加载的菜单文件，单击“打开”按钮，如图 15.18 所示。

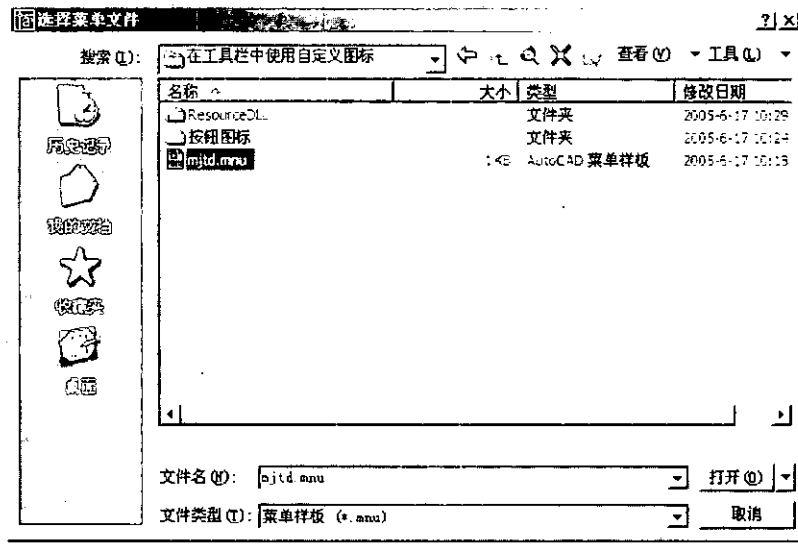


图 15.18　选择菜单文件

(3) 在“菜单自定义”对话框中单击“加载”按钮，在弹出的对话框中单击“是”按钮，确定继续加载菜单文件，完成加载。单击“关闭”按钮，关闭“菜单自定义”对话框，在图形窗口中就得到了如图 15.19 所示的工具栏。

单击某个工具按钮，就能执行相关的命令。

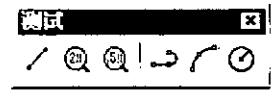


图 15.19　自定义工具栏

15.2.5　小结

在工具栏中使用自定义的图标，有两个要点：

- DLL 文件必须与菜单文件同名。
- DLL 文件必须与菜单文件放置在同一个文件夹中。

需要注意的一点是，在手工创建的菜单文件中可以使用 DLL 中的图标，但是通过 VBA 程序自动创建的工具栏无法使用。

15.3　基本图库的实现

15.3.1　实例说明

本节的实例将实现一个带有预览功能的图库，其运行界面如图 15.20 所示，用户选择一个预览图形之后，就可以将其对应的块插入到图形中。

如果仅仅包含预览的功能，这个程序没有多少令人感兴趣的东西，于是这里还为其添加了用户自定义图库内容的功能。在这个程序中，用户可以在图形窗口中绘制自己需要的图块，然后将其保存到图库中，并显示在预览界面中。

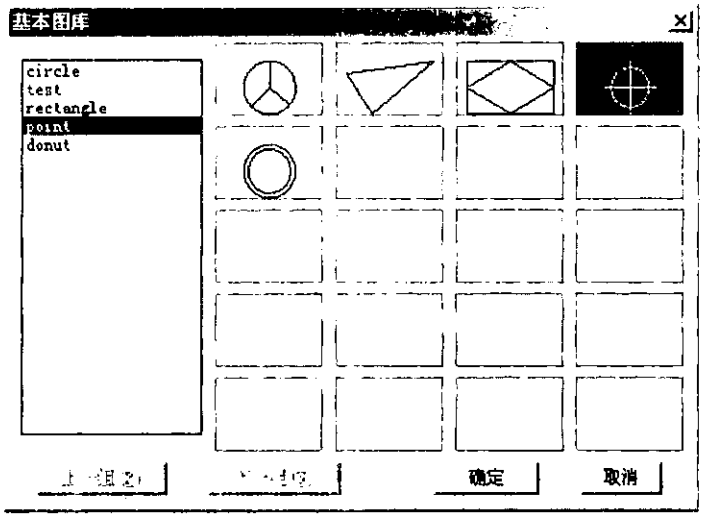


图 15.20 预览图库的界面

15.3.2 编程思路

为了实现图库的预览，在程序所在的文件夹中包含了两个重要的文件夹，Blocks(保存了图库中的 DWG 文件)和 Slides(保存了 DWG 文件对应的幻灯片，用于预览时显示)。Blocks 和 Slides 文件夹中同时包含若相同名称的 DWG 文件和 SLD 文件，这样用户在预览界面上选择某个幻灯片文件之后，就可以从 Blocks 文件夹中找到与其对应的 DWG 文件，将其插入到当前图形中。

图库的预览界面实际上是一个图像控件菜单(从外表来看，将其称为一个菜单确实有些勉强，但是 AutoCAD 帮助文档中确实是用这个名称来表示，后面会发现这个名称还是有一定道理的)，其中的标签定义包含了多种语法，本节的实例选择了下面的语法(之所以选择这种语法，与整个系统的设计有关)：

```
[sldname,labeltext] macro
```

其中，sldname 指定了标签所要显示的幻灯片名称；labeltext 是在列表框中的名称，作为幻灯片的说明；macro 是一个菜单宏，定义了此标签所要执行的任务。

关于图像控件菜单的详细介绍，请参考“AutoCAD 2005 帮助：开发人员文档”中的“自定义手册/菜单文件/创建图像控件菜单”部分。

用户自定义图库的实现步骤如下：

(1) 选择“创建块”菜单项之后，启用文档层事件，并在 ObjectAdded 事件中将此后创建的所有实体添加到选择集 SSet 中。

(2) 选择“保存块”菜单项之后，停止文档层事件，提示用户输入块的基点和名称，使用 Wblock 方法将其导出为一个 DWG 文件，保存在 Blocks 文件夹中。

(3) 打开步骤(2)生成的 DWG 文件，为其制作一张幻灯片，保存在 Slides 文件夹中，文件名称与 DWG 文件一致，然后关闭 DWG 文件。

(4) 在本节实例专门使用的菜单文件 BlockLib.mnu 中，写入图像控件菜单的一个标签

项定义语句。

(5) 卸载、重新加载 BlockLib. mnu 菜单，确保更新的 BlockLib. mnu 文件已经被 AutoCAD 加载，图像控件菜单中的标签项已经得到更新。

(6) 删除 SSet 选择集中所有的实体，然后删除 SSet 选择集自身。

15.3.3 步骤

(1) 打开记事本应用程序，在其中输入下面的内容：

```
//菜单组名称
*** MENUGROUP = BlockLib

//菜单
*** POP1
** BlkLib

ID_BL [图库(&L)]
ID_BLInsert [插入块(&R)] $ I = BlockLib. IMAGE_BLKLIB $ I = BlockLib. *
ID_BLAddBlk [创建块(&D)]^C^C-VBARUN 基本图库的实现. dvb!mdlStart. InitializeEvents
ID_BLSaveBlk [保存块(&S)]^C^C-VBARUN 基本图库的实现. dvb!mdlStart. StopEvent

//工具栏
*** TOOLBARS

//图像控件菜单
*** IMAGE

** IMAGE_BLKLIB
[基本图库]
```

务必在最后有一个空行(非菜单文件夹内容)

之所以在文件的最后保留一个空行，是要确保在程序中写入菜单文件内容时不会紧跟在[基本图库]之后，而是从新的一行开始。关于菜单文件的详细语法，可以参考 AutoCAD 帮助系统中的说明。

将文件保存为 BlockLib. mnu，可放置在与程序文件相同的文件夹中。

(2) 在 BlockLib. mnu 文件所在的目录中，创建两个文件夹，其名称分别为 Blocks 和 Slides。

(3) 在 AutoCAD 2005 中，打开 VBA 管理器，创建一个新工程，保存在 BlockLib. mnu 文件所在的文件夹中。进入 VBA 集成开发环境，选择“插入”|“模块”菜单项，向工程中添加一个标准模块，将其名称修改为 mdlStart，在其中添加一个全局变量：

```
Public SSet As AcadSelectionSet      '保存用户创建的组成块的实体
```

(4) 选择“插入”|“类模块”菜单项，向工程中添加一个类模块，将其名称修改为 EventClassModule，在其中添加下面的代码：

```
Public WithEvents Doc As AcadDocument

Private Sub Doc_ObjectAdded(ByVal Object As Object)
    If TypeOf Object Is AcadEntity Then
        If Not TypeOf Object Is AcadBlock Then
            Dim ent(0) As AcadEntity
            Set ent(0) = Object
            SSet.AddItems ent
        End If
    End If
End Sub
```

上面的代码捕获文档层事件 ObjectAdded，在其处理过程中将用户创建的所有实体添加到选择集 SSet 中。

(5) 在 mdlStart 模块中，添加一个全局变量，用于启用和停止文档层事件，其相关代码为：

```
Dim X As New EventClassModule        '类模块 EventClassModule 的实例，用于启动和停止文档事件
```

(6) 选择“工具”|“引用”菜单项，在弹出的对话框中引用 Microsoft Visual Basic for Applications Extensibility 5.3 类型库。添加一个宏 LoadMenu，用于加载本节的程序所要使用的菜单文件 BlockLib.mnu，方便用户的使用，其实现代码为：

```
Public Sub LoadMenu( )
    '获得相对路径
    Dim strPath As String        '当前工程所在的路径
    strPath = ThisDrawing.Application.VBE.ActiveVBProject.FileName
    strPath = Left$(strPath,Len(strPath)-Len("基本图库的实现.dvb"))

    '加载并且显示菜单
    ThisDrawing.Application.MenuGroups.Load strPath & "BlockLib.mnu"
    ThisDrawing.Application.MenuGroups.Item("BlockLib").Menus.Item(0). _
            InsertInMenuBar (ThisDrawing.Application.MenuBar.Count + 1)
End Sub
```

(7) 添加一个宏 InitializeEvents，用于启用文档层事件，并且在图形中添加一个名为 Example 的选择集，其实现代码为：

```
Public Sub InitializeEvents( )
    '启用文档事件
    Set X.Doc = ThisDrawing

    '安全创建选择集
    On Error Resume Next
    If Not IsNull(ThisDrawing.SelectionSets.Item("Example")) Then
        Set SSet = ThisDrawing.SelectionSets.Item("Example")
        SSet.Delete
    End If
    Set SSet = ThisDrawing.SelectionSets.Add("Example")
```

```
End Sub
```

(8) 添加一个宏 StopEvent,用于停止文档层事件,并且完成图块的创建和图库的更新,其实现代码为:

```
Public Sub StopEvent( )
    On Error GoTo ERRHANDLE
    '停止文档事件
    Set X = Nothing

    '提示用户输入块定义的基点
    Dim ptBase As Variant
    ptBase = ThisDrawing.Utility.GetPoint(,vbCrLf & "拾取基点: ")

    '提示用户输入块定义的名称
    Dim strName As String
    strName = ThisDrawing.Utility.GetString(True,vbCrLf & "输入块的名称: ")

    '获得相对路径
    Dim strPath As String
    strPath = ThisDrawing.Application.VBE.ActiveVBProject.FileName
    strPath = Left$(strPath,Len(strPath)-Len("基本图库的实现.dvb"))

    '将所有的实体移动到原点附近,确保块定义的插入点无误
    Dim ptOrigin(0 To 2) As Double
    ptOrigin(0) = 0: ptOrigin(1) = 0: ptOrigin(2) = 0
    Dim ent As AcadEntity
    For Each ent In SSet
        ent.Move ptBase,ptOrigin
    Next ent

    '将块定义导出
    ThisDrawing.Wblock strPath & "Blocks\" & strName & ".dwg",SSet

    '设置系统变量,确保不弹出对话框
    ThisDrawing.SetVariable "FILEDIA",0

    '生成幻灯片
    ThisDrawing.Application.Documents.Open strPath & "Blocks\" & strName & ".dwg"
    ThisDrawing.Application.ZoomExtents
    ThisDrawing.SendCommand "MSLIDE" & vbCr & strPath & "Slides\" & strName & ".sld" & vbCr
    ThisDrawing.Close

    '还原系统变量
    ThisDrawing.SetVariable "FILEDIA",1
```

```
    '将新的块名称写入菜单文件
    Open strPath & "BlockLib.mnu" For Append As #1
    Print #1,"[" & strName & "," & strName & "]^C^C" & "-Insert " & strName & ".dwg" & " \"
    Close 1

    '卸载要更新的菜单文件
    ThisDrawing.Application.MenuGroups.Item("BlockLib").Unload
    If Len(Dir(strPath & "BlockLib.mns")) > 0 Then
        Kill strPath & "BlockLib.mns"
    End If
    If Len(Dir(strPath & "BlockLib.mnc")) > 0 Then
        Kill strPath & "BlockLib.mnc"
    End If

    '更新菜单文件
    ThisDrawing.Application.MenuGroups.Load strPath & "BlockLib.mnu"
    ThisDrawing.Application.MenuGroups.Item("BlockLib").Menus.Item(0).InsertInMenuBar
        (ThisDrawing.Application.MenuBar.Count + 1)

    '删除图形中绘制的所有对象
    SSet.Erase

    '删除选择集
    SSet.Delete

ERRHANDLE:
    End
End Sub
```

使用 Set X=Nothing 语句即可停止处理文档层事件，但是如果在 ThisDrawing 模块中处理文档层事件，就无法在需要的时候停止文档层事件了。

之所以将选择集中所有的实体按照从 ptBase 到 ptOrigin 的矢量进行移动，是为了保证生成的 DWG 文件具有正确的基点。Wblock 方法无法指定块的基点，系统会自动将当前图形的原点作为基点，如果不在执行 Wblock 方法之前将选择集中的所有对象按照此矢量进行移动，那么用户输入的基点 ptBase 就毫无用武之地。

生成幻灯片使用的是 AutoCAD 中的 MSLIDE 命令，但是为了不让该命令执行过程中弹出对话框，就必须将系统变量 FILEDIA 的值设置为 0，在执行 MSLIDE 命令之后再将其设置为 1。

写入菜单文件使用了下面的代码：

```
Open strPath & "BlockLib.mnu" For Append As #1
Print #1,"[" & strName & "," & strName & "]^C^C" & "-Insert " & strName & ".dwg" & " \"
Close 1
```

菜单文件的默认编辑器就是记事本，当然可以使用 Open 语句打开以顺序文件的形式进行读写。注意 Open 语句中使用的关键字是 Append，确保原来菜单文件中的内容被保留下来，

在菜单文件的末尾写入新的菜单项。写入图像控件菜单的标签项时，strName 同时被作为幻灯片文件名称和显示的字符串名称，而后面的菜单宏则可以确保完成插入块中选择块的步骤。

可以发现，在卸载和重新加载菜单文件之间，删除了 BlockLib. mns 和 BlockLib. mnc 两个文件，这是因为，AutoCAD 按照下面的原则来加载菜单文件：

- 如果找到一个 MNS 文件，AutoCAD 会在同一个目录中查找编译后的同名菜单(MNC)文件。如果 AutoCAD 找到一个匹配的 MNC 文件，并且该文件日期和时间与 MNS 文件日期和时间相同或较之稍后，则加载该 MNC 文件。否则，AutoCAD 编译该 MNS 文件，在同一个目录中生成新的 MNC 文件，然后加载该 MNC 文件。
- 如果没有找到 MNS 文件，AutoCAD 按照库搜索步骤查找具有给定名称的编译后菜单(MNC)文件。如果 AutoCAD 找到该 MNC 文件，将加载该文件。
- 如果 MNS 和 MNC 文件均未找到，AutoCAD 将搜索库路径，查找具有给定名称的菜单样板(MNU)文件。如果找到此文件，AutoCAD 将编译 MNC 文件，生成 MNS 文件，然后加载 MNC 文件。
- 如果未找到具有给定名称的文件，AutoCAD 将显示错误信息，并提示用户输入其他菜单文件名。

在 BlockLib. mns 和 BlockLib. mnc 两个文件存在的情况下，AutoCAD 永远不会加载更新的 BlockLib. mnu 文件，也就无法更新图库的预览界面。因此，只能在程序中删除这两个文件，强迫 AutoCAD 加载更新后的 BlockLib. mnu 文件，实现图库预览界面的更新。

(9) 在 AutoCAD 2005 中，选择“工具”|“选项”菜单项，在弹出的“选项”对话框中，切换到“文件”选项卡，向“搜索路径、文件名和文件位置”列表框中添加三个路径，也就是程序所在的文件夹、Blocks 文件夹和 Slides 文件夹，如图 15.21 所示。

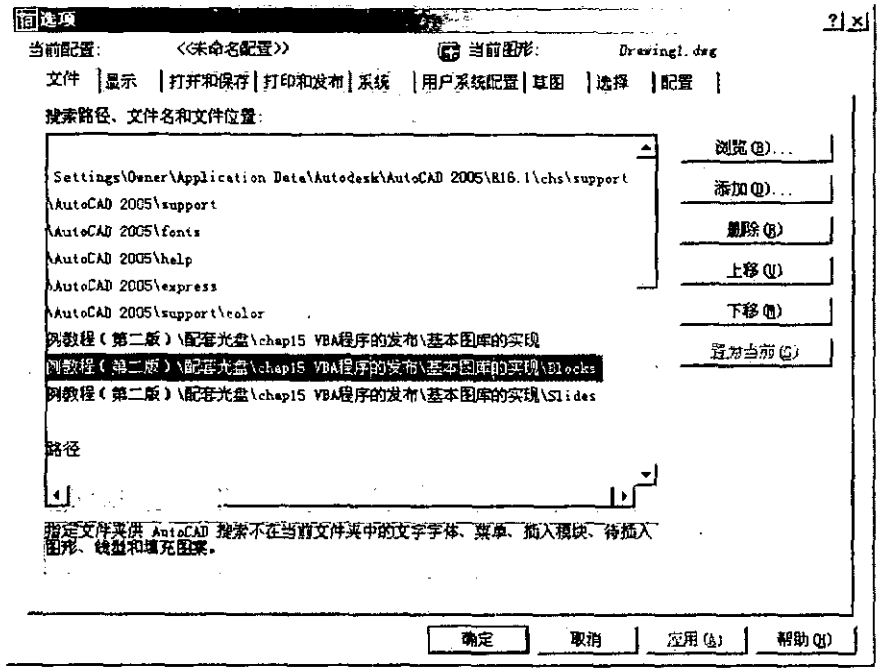


图 15.21　添加支持文件的搜索路径

15.3.4 实例效果

(1) 在 VBA 集成开发环境中，执行 LoadMenu 宏，将 BlockLib. mnu 菜单加载到 AutoCAD 2005 中，在 AutoCAD 菜单栏的最后，就出现了“图库”菜单组，如图 15.22 所示。

(2) 选择“图库”|“创建块”菜单项，然后在图形窗口中添加组成块的图形对象，如图 15.23 所示。

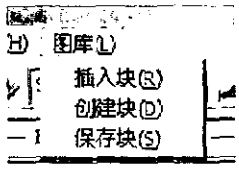


图 15.22 显示加载的菜单

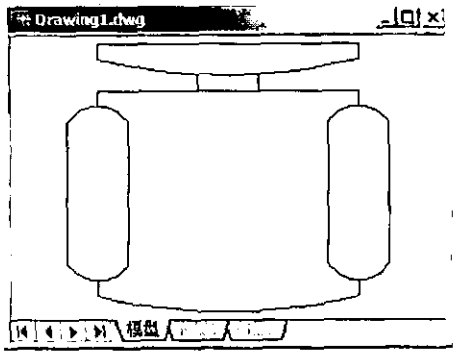


图 15.23 创建组成块的实体

(3) 选择“图库”|“保存块”菜单项，按照系统的提示，指定块的基点，输入“桌椅”作为块的名称，完成块的定义。

(4) 选择“图库”|“插入块”菜单项，系统会弹出如图 15.24 所示的“基本图库”对话框。

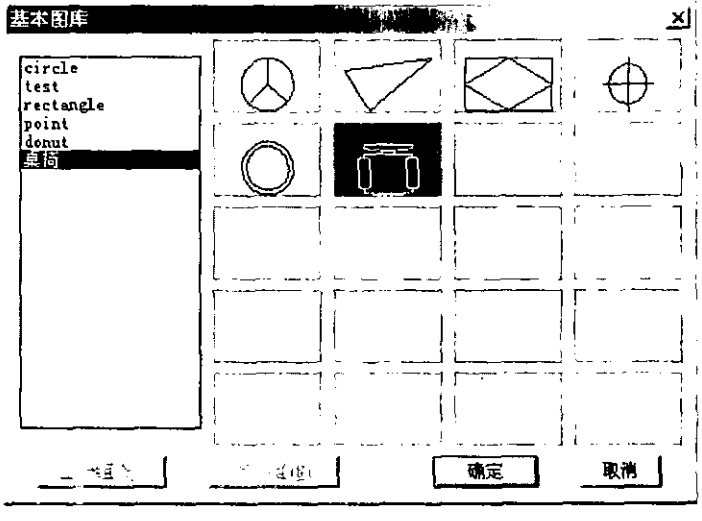


图 15.24 已经更新的预览界面

(5) 选择“桌椅”块，单击“确定”按钮，在 AutoCAD 图形窗口中完成块的插入，其命令提示为：

```
命令：-Insert 输入块名或 [?] <A$C7AEC068D>：桌椅.dwg 指定插入点或
[比例(S)/X/Y/Z/旋转(R)/预览比例(PS)/PX/PY/PZ/预览旋转(PR)]：        【从这里开始执行，用
    户可以移动光标，选择块参照的插入点】
```

```
输入 X 比例因子,指定对角点,或 [角点(C)/XYZ] <1>:
输入 Y 比例因子或 <使用 X 比例因子>:

指定旋转角度 <0>:
```

15.3.5　小结

学习本节内容之后,需要掌握下面的内容:

- 菜单文件的加载和卸载。
- 在程序中修改菜单文件。
- 文档事件的启用和停止。
- 使用 Wblock 方法导出块时基点的调整。
- 在 VBA 中创建幻灯片文件。

实际上,很多情况下都可以将 FILEDIA 系统变量设置为 0,然后使用 SendCommand 方法调用 AutoCAD 内部命令完成一些特殊的工作。例如,下面的代码能将当前图形窗口中已经着色的实体导出为 BMP 位图:

```
Public Sub ExportBMP( )
    '设置系统变量,确保不弹出对话框
    ThisDrawing.SetVariable "FILEDIA",0

    '导出当前图形窗口显示的图像
    ThisDrawing.SendCommand "BMPOUT" & vbCr & "C:\test.bmp" & vbCr & vbCr

    '还原系统变量
    ThisDrawing.SetVariable "FILEDIA",1
End Sub
```

15.4　在命令行或菜单中执行 VBA 程序

15.4.1　实例说明

AutoCAD 的二次开发,为的是执行一些复杂并且重复的工作。一般来说,AutoLISP 编制的程序加载后就可以直接在命令行中输入程序命令来执行。但对于 VBA 程序,如果不通过 LISP 程序来调用,则只能在宏对话框中加载和执行了。这样的程序不够专业,也不利于分发给用户使用。

为了让 VBA 程序可以在命令行中执行,开发者还必须了解 AutoLISP 的一点编程知识。

15.4.2　编程思路

可以说不利用 LISP 程序而直接在命令行中简单加载运行 VBA 程序的可能性是没有的,所以本节所有的例子都是 LISP 程序。

通过 LISP 程序，可以在需要时直接加载并运行 VBA 程序；也可以先加载，并在需要时运行 VBA 程序；还可以使用专用的自定义 LISP 函数定义一个通用的运行函数，来调用各种不同 VBA 程序。此外，还可以使用事件触发的方法来运行 VBA 程序。

本节中还介绍了怎样把 VBA 程序的运行直接写到菜单文件中，在选择菜单项或单击工具栏按钮时运行 VBA 程序。

LISP 程序可以通过 VisualLISP 编辑器来编制，它可以通过选择 AutoCAD 中的“工具”|AutoLISP|“VisualLISP 编辑器”菜单项来打开。由于本节中的 LISP 程序都非常简单，所以只需要使用 Windows 系统的记事本来编辑保存。

15.4.3 步骤

为了演示 VBA 程序的运行过程，首先编制一个简单的 VBA 程序。

(1) 在 AutoCAD 2004 中，打开 VBA 管理器，创建一个新工程，进入 VBA 集成开发环境。

(2) 在“工程资源管理器”窗口中，双击 ThisDrawing 图标，打开其代码窗口，添加宏的定义代码：

```
Sub Draw2Line( )
'这是一个绘制两段不同颜色线段的程序
    Dim FromPnt As Variant
    Dim MidPnt As Variant
    Dim ToPnt As Variant
    Dim Line1 As AcadLine
    Dim Line2 As AcadLine

    ThisDrawing.Utility.Prompt vbCr & "绘制两段线段程序" & vbCrLf
    FromPnt = ThisDrawing.Utility.GetPoint(,vbCr & "指定起点：")
    MidPnt = ThisDrawing.Utility.GetPoint(FromPnt,vbCr & "指定下一点：")
    Set Line1 = ThisDrawing.ModelSpace.AddLine(FromPnt,MidPnt)
    Line1.color = acRed
    ToPnt = ThisDrawing.Utility.GetPoint(MidPnt,vbCr & "指定终点：")
    Set Line2 = ThisDrawing.ModelSpace.AddLine(MidPnt,ToPnt)
    Line2.color = acYellow

End Sub
```

(3) 选择“插入”|“模块”菜单项，向程序中添加一个标准模块，将模块的名称修改为 Draw，在其中添加宏的代码：

```
Sub Draw3Line( )
'这是一个绘制三段不同颜色线段的程序
    Dim Pnt1 As Variant
    Dim Pnt2 As Variant
    Dim Pnt3 As Variant
    Dim Pnt4 As Variant
```

```
    Dim Line1 As AcadLine
    Dim Line2 As AcadLine
    Dim Line3 As AcadLine

    ThisDrawing.Utility.Prompt vbCr & "绘制三段线段程序" & vbCrLf
    Pnt1 = ThisDrawing.Utility.GetPoint(,vbCr & "指定第 1 点: ")
    Pnt2 = ThisDrawing.Utility.GetPoint(Pnt1,vbCr & "指定第 2 点: ")
    Set Line1 = ThisDrawing.ModelSpace.AddLine(Pnt1,Pnt2)
    Line1.color = acRed
    Pnt3 = ThisDrawing.Utility.GetPoint(Pnt2,vbCr & "指定第 3 点: ")
    Set Line2 = ThisDrawing.ModelSpace.AddLine(Pnt2,Pnt3)
    Line2.color = acGreen
    Pnt4 = ThisDrawing.Utility.GetPoint(Pnt3,vbCr & "指定第 4 点: ")
    Set Line3 = ThisDrawing.ModelSpace.AddLine(Pnt3,Pnt4)
    Line3.color = acYellow

End Sub
```

完成以上程序后，将文件保存在 AutoCAD 的支持目录下，命名为 DrawLine.dvb。这样，文件名为 DrawLine.dvb 的 VBA 文件中有两个过程，一个是在 Thisdrawng 模块中的 Draw2Line 程序，它用于绘制两段不同颜色的线段，另一个是在 Draw 模块中的 Draw3Line 程序，它用于绘制三段不同颜色的线段。

1. 通过-VbaRun 命令来定义执行命令

(1) 打开记事本，输入以下内容：

```
(defun c: Draw2Line( )
    (setvar "cmdecho" 0)
    (command "-vbarun" "DrawLine.dvb!Draw2Line")
    (setvar "cmdecho" 1)
(princ)
)
(defun c: Draw3Line( )
    (setvar "cmdecho" 0)
    (command "-vbarun" "DrawLine.dvb!Draw.Draw3Line")
    (setvar "cmdecho" 1)
(princ)
)
```

输入后保存为 DrawLine1.lsp 文件名。

提示：要将记事本中创建的文件保存为其他格式，可选择“文件”|“另存为”菜单项，在弹出的“另存为”对话框中，将“保存类型”修改为“所有文件”，然后输入一个带有 LSP 扩展名的文件名，如图 15.25 所示，单击“保存”按钮，即可将当前的文件保存为 LSP 文件。

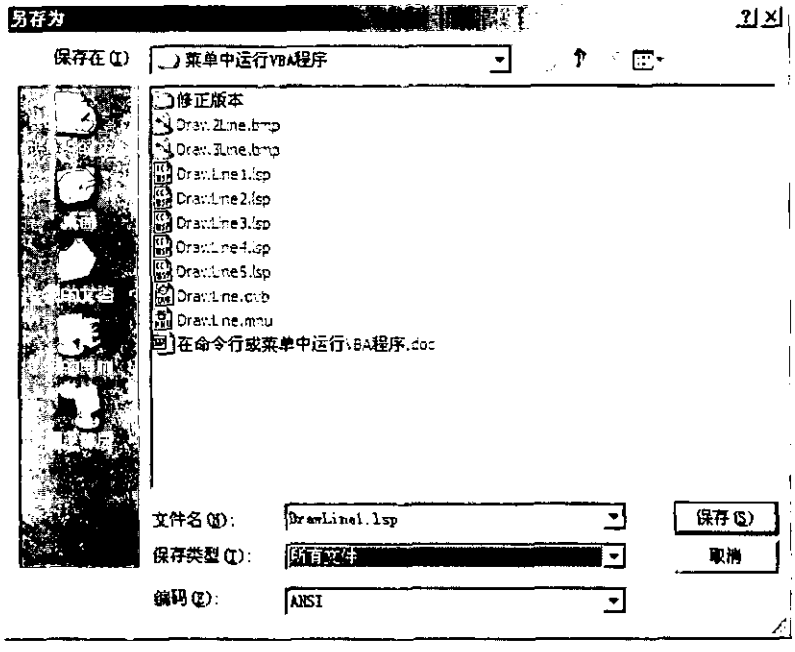


图 15.25 在记事本中保存 LSP 文件

(2) 在 AutoCAD 2005 中,选择"工具"|"加载应用程序"菜单项,在弹出的"加载"|"卸载应用程序"对话框中,选择刚保存的 DrawLine1.lsp 文件,单击"加载"按钮,然后单击"关闭"按钮,关闭该对话框。

2. 通过 vl-vbaload 和 vl-vbarun 函数来定义执行命令

(1) 打开记事本,并输入以下内容:

```
(vl-load-com)
(vl-vbaload (findfile "DrawLine.dvb"))
(defun C: Draw2Line( )
  (vl-vbarun "Draw2Line")
  (princ)
)
(defun C: Draw3Line( )
    (vl-vbarun "Draw.Draw3Line")
    (princ)
)
```

输入后保存为 DrawLine2.lsp 文件。

(2) 在 AutoCAD 2005 中,选择"工具"|"加载应用程序"菜单项,在弹出的"加载"|"卸载应用程序"对话框中,选择刚保存的 DrawLine2.lsp 文件,单击"加载"按钮,然后单击"关闭"按钮,关闭该对话框。

3. 通过定义通用运行函数来执行程序

(1) 打开记事本,并输入以下内容:

```
(vl-load-com)
'自动加载 VBA 程序的函数
(defun AutoVBALoad (cmdname project macro)
  (eval
   (list 'defun
         (read (strcat "C: " cmdname))
         nil
         (list  'vl-vbarun
             (strcat    project    "!"
                 (if macro macro cmdname )
             )
         )
         (princ)
       )
   )
)
'自动加载 VBA 程序
(AutoVBALoad "Draw2Line" "DrawLine.dvb" "Draw2Line")
(AutoVBALoad "Draw3Line" "DrawLine.dvb" "Draw.Draw3Line")
```

输入后保存为 DrawLine3.lsp 文件。

(2) 在 AutoCAD 2005 中，选择“工具”|“加载应用程序”菜单项，在弹出的“加载”|“卸载应用程序”对话框中，选择刚保存的 DrawLine3.lsp 文件，单击“加载”按钮，然后单击“关闭”按钮，关闭该对话框。

该自定义函数使用格式为：

```
(AutoVBALoad "命令名" "VBA 文件名" "过程名")
```

4. 使用功能更强大的自定义通用运行函数来执行程序

(1) 打开记事本，并输入以下内容：

```
(vl-load-com)
'自动加载 VBA 程序的函数
(defun AutoVBALoad (app cmdliste/qapp)
(setq qapp (strcat "\"" app "\""))
(mapcar
    '(lambda (cmd/nom_cmd dot nodotcmd)
      (progn
        (setq dot (vl-string-search "." cmd))
        (if dot
          (setq nodotcmd (substr cmd ( + dot 2)))
          (setq nodotcmd cmd)
        )
        (setq nom_cmd (strcat "C: " nodotcmd))
        (eval
```

```
            (read (strcat
                    "(defun " nom_cmd "(/app)"
                      "(if (setq app(fdvbfile " qapp "))"
                        "(vl-vbarun (strcat app \"!" cmd "\"))"
                        "(nodvbfile " qapp "))"
                    "(princ))"
        )))))
      cmdliste
    )
    nil
)

(defun fdvbfile (app)
  (if (not (findfile app))
    (if (not (findfile (strcat app ".dvb"))) nil (strcat app ".dvb")) app)
)

(defun nodvbfile (filename)
  (princ (strcat "\n文件 " filename "(.dvb) 在搜索路径文件夹中未找到。"))
  (princ "\n请检查支持文件的安装,然后重试。")
  (princ)
)
'自动加载 VBA 程序
(AutoVBALoad "DrawLine" '("Draw2Line" "Draw.Draw3Line"))
```

输入后保存为 DrawLine4.lsp 文件名。

(2) 在 AutoCAD 2005 中,选择“工具”|“加载应用程序”菜单项,在弹出的“加载”|“卸载应用程序”对话框中,选择刚保存的 DrawLine4.lsp 文件,单击“加载”按钮,然后单击“关闭”按钮,关闭该对话框。

该自定义函数使用格式为:

```
(AutoVBALoad "VBA 文件名" "过程名列表")
```

这样,该函数可自动将过程名列表设置为执行命令。如果过程名格式为“模块.过程名”,则执行命令只取过程名部分,大大方便了 VBA 程序的调用。

5. 通过定义一个空的 LISP 程序再结合 VBA 中的事件来触发程序

(1) 打开记事本,并输入以下内容:

```
(defun C: Draw2Line( )(princ))
(defun C: Draw3Line( )(princ))
```

输入后保存为 DrawLine5.lsp 文件名。

(2) 打开原先保存的 DrawLine.dvb 文件,并在 ThisDrawing 模块中再输入以下程序:

```
Private Sub AcadDocument_BeginLisp(ByVal FirstLine As String)
    Select Case UCase(FirstLine)
```

```
        Case "(C: DRAW2LINE)"
            Draw2Line
        Case "(C: DRAW3LINE)"
            Draw3Line
    End Select
End Sub
```

(3) 在 AutoCAD 2005 中，选择“工具”|“加载应用程序”菜单项，在弹出的“加载”|“卸载应用程序”对话框中选择刚保存的 DrawLine5. lsp 文件，单击“加载”按钮，然后单击“关闭”按钮，关闭该对话框。

6. 通过选定菜单项或工具按钮来执行 VBA 程序

(1) 打开记事本，并输入以下内容：

```
///  菜单组名称
*** MENUGROUP = MccadDrawLine

///  菜单
*** POP1
ID_MCDL     [画线(&L)]
ID_McD2L    [两段线段(&2)]^C^C-VBARUN DrawLine.dvb!Draw2Line
ID_McD3L    [三段线段(&3)]^C^C-VBARUN DrawLine.dvb!Draw.Draw3Line

///  工具栏
*** TOOLBARS
** TB_McDrawLine_TOOLBAR
ID_MCDL [_Toolbar("画线",_Floating,_show,90,110,1)]
ID_McD2L [_Button("两段线段","Draw2Line.BMP","Draw2Line.BMP")]^C^C-VBARUN DrawLine.dvb!
   Draw2Line
ID_McD3L [_Button("三段线段","Draw3Line.BMP","Draw3Line.BMP")]^C^C-VBARUN DrawLine.dvb!
   Draw.Draw3Line

///  帮助信息，选择指定菜单或工具栏时在状态栏显示的信息
*** HELPSTRINGS
ID_McD2L    [绘制两段颜色不同的线段]
ID_McD3L    [绘制三段颜色不同的线段]
```

输入后保存为 DrawLine. mnu 文件名，这个文件即为菜单文件。

(2) 在 AutoCAD 2005 中，选择“工具”|“自定义”|“菜单”菜单项，在弹出的“菜单自定义”对话框中，选择“菜单组”选项卡，并单击右下方的“浏览”按钮，在“选择菜单文件”对话框的“文件类型”下拉列表框中选择“菜单样板(*. mnu)”，找到并选择 DrawLine. mnu 文件，并单击“打开”按钮，返回“菜单自定义”对话框，如图 15.26 所示。

单击“加载”按钮，加载菜单文件，AutoCAD 会提示是否确定该操作，单击“是”按钮后，此时在 AutoCAD 界面上出现有“画线”的工具栏，如图 15.27 所示。如果在 AutoCAD 的支

持文件搜索路径中没有 Draw2Line. BMP 和 Draw3Line. BMP 这两个图标文件，则工具栏按钮显示的是两个笑脸(2002 及以前版本)或问号(2004 及以上版本)。

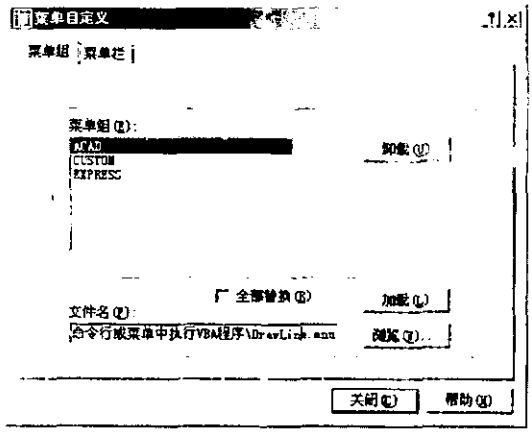


图 15.26 加载菜单文件



图 15.27 自定义的工具栏

菜单的显示还需要进一步的操作，单击“菜单自定义”对话框中的“菜单栏”选项卡，在菜单组列表中选择 MccadDrawLine，在左侧的“菜单”列表框中选择“画线”选项，右侧的“菜单栏”列表框中选择“帮助”选项，然后单击“插入”按钮，将“画线”菜单栏添加到“帮助”菜单栏的前面，如图 15.28 所示。

关闭“菜单自定义”对话框后，在 AutoCAD 的菜单中可以看到自定义的菜单了，如图 15.29所示。

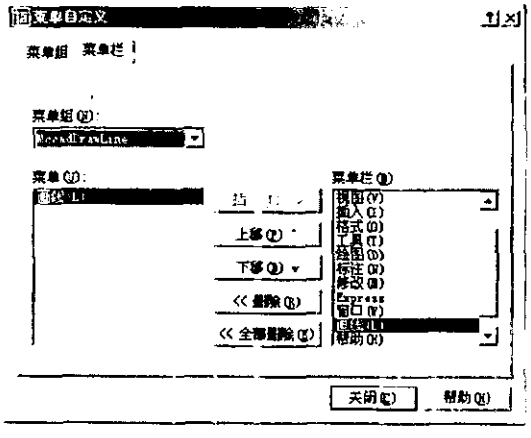


图 15.28 显示自定义菜单的方法

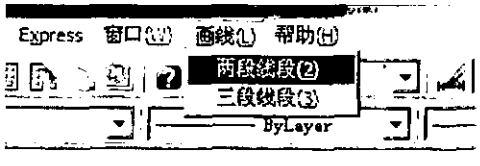


图 15.29 显示自定义的菜单

7. 通过 VBA 生成执行 VBA 程序的菜单或工具栏

这一部分在本书第一版的 10.1 和 10.2 节中已经介绍，这里只指出在菜单或工具栏的宏中需要填入什么样的内容才能让菜单或工具栏执行 VBA 程序。

对于菜单和工具栏，宏部分可以这样写：

```
Chr(27) + Chr(27) + "-vbarun" + Chr(32) + " DrawLine.dvb!Draw2Line" + Chr(32))
```

```
Chr(27) + Chr(27) + "-vbarun" + Chr(32) + " DrawLine.dvb!Draw.Draw3Line" + Chr(32))
```

其中 Chr(27)相当于菜单文件中的^C(即取消),Chr(32)相当于菜单文件中的空格。以上两行编译过后就成为菜单文件中的:

```
^C^C-VBARUN DrawLine.dvb!Draw2Line
^C^C-VBARUN DrawLine.dvb!Draw.Draw3Line
```

15.4.4　实例效果

步骤中的前 5 项,当加载了 LISP 文件后,就可以在命令行中输入 Draw2Line 执行绘制两段线段的程序,输入 Draw3Line 执行绘制三段线段的程序。

在步骤中的 6 及 7 项中,用户可以单击工具栏或菜单的菜单项来执行 VBA 程序。

15.4.5　小结

本节所涉及到的知识点如下所示。

- VbaRun 命令显示的是"宏"对话框,而-VbaRun 命令则仅在命令行执行。
- -VbaRun 命令运行 VBA 程序,可以带文件名,也可以不带文件名。如果 VBA 程序已经加载,则可以不带文件名。
- VisualLISP 的 vl-vbaload 函数用于加载 VBA 程序,而 vl-vbarun 函数用于运行 VBA 程序。如果 VBA 程序已经加载,则可以不带文件名。
- 对于 Thisdrawing 模块中的过程,在使用运行语句时不需要加上 ThisDrawing.,而直接写上过程名则可,如 Draw2Line,而不需要写成 ThisDrawing.Draw2Line。如果在其他模块中的过程,则需要加上模块名,格式为"模块名.过程名",如 Draw.Draw3Line。
- 如果需要带文件名,则格式为"文件名!模块名.过程名"。对于 Thisdrawing 模块中的过程,格式为"文件名!过程名"。
- 对于步骤中的第 3 项中的 AutoVBALoad 函数,如果宏在 ThisDrawing 中,并且 macro 为空,则直接使用 cmdname 为 macro 名称。也就是说,如果过程是在 ThisDrawing 中,并且命令名与过程名一样,则 macro 参数可以省去为空。如本节中的语句

  ```
  (AutoVBALoad "Draw2Line" "DrawLine.dvb" "Draw2Line")
  ```

  可以写成

  ```
  (AutoVBALoad "Draw2Line" "DrawLine.dvb" "")
  ```

- 对于不熟悉 LISP 编程的人员来说,利用事件来触发 VBA 程序有一定的优越性。因为基本上所有的编程都在 VBA 中进行。但由于 AutoCAD 本身存在着一些 bug,这种情况也有一定的限制,目前发现的问题是使用事件来触发对话框显示时,如果隐藏对话框在 AutoCAD 进行交互操作(如选择对象、输入点等操作),则光标无法显示供用户操作。这一点在实际应用中应该注意。
- 使用菜单或工具栏来执行 VBA 程序是一个好的方法,界面友好,非常直观好用。如

果再与命令行方式结合使用则会更加合理。对于编制工具集的用户来说，使用菜单及工具栏来提供可供使用的功能是非常好的选择。

- 对于菜单及工具栏的编制方法，在 AutoCAD 的帮助中有详细的说明，具体位置为“帮助”|“自定义手册”|“菜单文件”。如果 AutoCAD 为中文版本，则这些帮助内容均为中文的。
- 在工具栏中使用的按钮图标为 .bmp位图文件，尺寸要求为 16 像素(宽)×15 像素(高)。可以使用 Windows 附件中的“画图”软件来制作位图文件。位图文件必须位于 AutoCAD 的支持文件搜索路径下，否则必须指定路径。
- 如果 VBA 文件不放在 AutoCAD 的支持目录中，则需要提供路径。一般情况下，为了程序的可移植性，VBA 文件必须放到支持文件搜索路径中，或者将 VBA 文件所在的目录添加到支持文件搜索路径中。
- 菜单文件第一次加载后，在每一次打开 AutoCAD 时均能显示，所以它是最简单的调用方法。而对于 LISP 程序，为了让 AutoCAD 每次均能加载它供用户使用，最直接的方法就是把 LISP 文件放到启动组中(本书第一版 10.4 节已经说明了怎样将文件放到启动组中，对于 VBA 程序和 LISP 程序，操作方法均相同)。
- 如果用户已经用菜单文件将所有的 VBA 程序及 LISP 程序均通过菜单显示出来，则可以通过与菜单文件同名的. mnl 文件来保存 LISP 程序，如本节中的 DrawLine. mnu 文件，则可以使用 DrawLine. mnl 文件来保存在菜单中需要调用或需要自动加载的 LISP 程序。用户可以将本节中的 DrawLine1. LSP 文件改名为 DrawLine. mnl 文件，并放到 AutoCAD 的支持文件搜索路径下，这样当使用了 DrawLine. mnu 菜单时，DrawLine. mnl 文件将会自动加载，这样直接在命令行中输入 Draw2Line 及 Draw3Line 均可运行相应的 VBA 程序。这种自动的加载方法在用户编制工具集时是最有用的方法。.mnl文件同样可以使用记事本打开。

15.5 实现 VBA 程序的先选择后操作和透明命令

15.5.1 实例说明

对于大部分 AutoCAD 用户来说，先选定一部分对象，再输入命令执行对这些选定对象的操作已经是非常习惯的做法。所以，实现先选择后操作是最基本的要求。

虽然在 LISP 程序中，实现这种方式只需要用(ssget)就可以实现这种功能，但在 VBA 中，却一直以来找不到一种简便的方法。

透明命令，也就是在其他命令执行过程中，可以在命令前加“'”(单引号)来暂时中断正在运行的命令，并执行指定的透明命令。在 AutoCAD 中，像 ZOOM(缩放)和 PAN(平移)命令就是透明命令。

本节将采用最简便的方法来实现这两种功能。

可以说，本节所述内容是国内外首次对实现先选择后操作和透明命令的最简便方法的

详细描述。也正是这两个方法的实现，使 VBA 程序能够与 LISP 程序，甚至和 AutoCAD 内部命令一样看待，让 VBA 程序不再低人一等。

15.5.2　编程思路

从程序的原理来说，当从“宏”对话框调用 VBA 程序时，由于在 VBA 程序运行前已经运行了其他的命令，此时当前选定的对象已经失去了焦点而不能被 VBA 程序所识别。同样，使用命令-VbaRun 或 Visual LISP 函数 vl-vbarun 来运行 VBA 程序时，也会使选定的对象失去焦点。

大家都明白，VBA 程序的运行除了事件触发就只能通过 LISP 程序来执行，它没有办法直接生成可在命令行执行的命令。

上一节虽然介绍了很多在命令行执行 VBA 程序的方法，但都使用了 vl-vbarun 函数，所以无法实现先选择后操作。

惟一的出路就是找到另外一个可以运行 VBA 程序的 LISP 函数出来。而且这个函数的使用不会影响目前所选定对象。

可幸的是，Visual LISP 不仅提供了普通的 LISP 函数，同样它也可以使用 ActiveX 方法来调用对象类型库的方法。而 AutoCAD 对象类型库中提供了一个 RunMacro 方法，它也可以用于执行 VBA 程序。因为都是 ActiveX 方法，它不会影响到目前选定对象变化。

先选择后操作的实现，为将 VBA 程序变成透明命令提供了可能性。因为如果一个命令会影响到其他命令的执行，就不能设置成透明命令，就算设置了，也只是表面上。

自从 2000 版以来，Visual LISP 中就提供了设置命令甚至透明命令的函数（vlax-add-cmd），它可以把 LISP 程序设置成透明命令，同样，这里也用这个函数把 VBA 程序设置成透明命令。严格地说，是将 VBA 程序用 LISP 方法来调用，然后再将 LISP 的调用方法设置成透明命令。

15.5.3　步骤

1. 实现先选择后操作

实现先选择后操作，需要两方面的配合，一个是 VBA 程序中必须带有可选择到当前选择集的方法，另外一个就是让调用 VBA 程序的 LISP 程序不影响用户所选定的对象。

(1) 在 AutoCAD 2005 中，打开 VBA 管理器，创建一个新工程，进入 VBA 集成开发环境。

(2) 在“工程资源管理器”窗口中，双击 ThisDrawing 图标，打开其代码窗口，添加宏的定义代码：

```
Sub CC( )
    Dim Ent As AcadEntity
    Dim SS As AcadSelectionSet
    Set SS = PickFirstSSet
    For Each Ent In SS
        Ent.color = acGreen
    Next
```

```
End Sub
'实现先选择后操作的函数
Function PickFirstSSet( ) As AcadSelectionSet
    On Error Resume Next
    ThisDrawing.SelectionSets("PICKFIRST").Delete
    Set PickFirstSSet = ThisDrawing.PickfirstSelectionSet
    If PickFirstSSet.Count = 0 Then PickFirstSSet.SelectOnScreen
End function
```

完成以上程序后，将文件保存在 AutoCAD 的支持目录下，命名为 ChangColor.dvb。这样，文件名为 ChangColor.dvb 的 VBA 文件中有一个名为 CC 的过程，它用来将用户选定的对象的颜色更改为绿色。

在这个程序中，自定义了一个 PickFirstSSet 函数来实现先选择后操作功能。

(3) 打开记事本，并输入以下内容：

```
(defun C: VBARUNX ( )
    (vl-load-com)
    (vla-runmacro
        (vlax-get-acad-object)
        (getstring "\n 宏名称: ")
    )
    (princ)
)
```

输入后保存为 VBARun.lsp 文件名。

(4) 在 AutoCAD 2005 中，选择“工具”|“加载应用程序”菜单项，在弹出的“加载”、“卸载应用程序”对话框中，选择刚保存的 VBARun.lsp 文件，单击“加载”按钮，然后单击“关闭”按钮，关闭该对话框。

2. 实现透明命令

透明命令一般用于图形视图的转换、系统配置的更改以及一些数据的查询。

为了用户能更清楚地看到透明命令运行的效果，这里先用 VBA 定义几个简单的窗口缩放功能的程序。

(1) 在 AutoCAD 2005 中，打开 VBA 管理器，创建一个新工程，进入 VBA 集成开发环境。

(2) 在“工程资源管理器”窗口中，双击 ThisDrawing 图标，打开其代码窗口，添加宏的定义代码：

```
'缩放到前一个视图
Sub ZP( )
    ZoomPrevious
End Sub

'全图缩放
Sub ZA( )
```

```
    ZoomAll
End Sub

'范围缩放
Sub ZE( )
    ZoomExtents
End Sub

'窗口缩放
Sub ZW( )
    ZoomPickWindow
End Sub
```

完成以上程序后，将文件保存在 AutoCAD 的支持目录下，命名为 Zoom.dvb。这样，文件名为 Zoom.dvb 的 VBA 文件中有 4 个过程，它们用来控制窗口的缩放。因为要做为透明命令，所以尽量使用最简单的过程名，以方便命令输入。

(3) 打开记事本，并输入以下内容：

```
(vl-load-com)
'缩放到前一个视图
(defun ZP( )
  (vla-runmacro (vlax-get-acad-object) "Zoom.dvb!ZP")
  (princ)
)
'全图缩放
(defun ZA( )
  (vla-runmacro (vlax-get-acad-object) "Zoom.dvb!ZA")
  (princ)
)
'范围缩放
(defun ZE( )
  (vla-runmacro (vlax-get-acad-object) "Zoom.dvb!ZE")
  (princ)
)
'窗口缩放
(defun ZW( )
  (vla-runmacro (vlax-get-acad-object) "Zoom.dvb!ZW")
  (princ)
)
'生成透明命令
(vlax-add-cmd "zp" 'zp "zp" 1)
(vlax-add-cmd "za" 'za "za" 1)
(vlax-add-cmd "ze" 'ze "ze" 1)
(vlax-add-cmd "zw" 'zw "zw" 1)
(princ)
```

输入后保存为 Zoom. lsp 文件名。

(4) 在 AutoCAD 2005 中,选择"工具"|"加载应用程序"菜单项,在弹出的"加载"|"卸载应用程序"对话框中,选择刚保存的 Zoom. lsp 文件,单击"加载"按钮,然后单击"关闭"按钮,关闭该对话框。

3. 通过定义通用运行函数来实现先选择后操作和透明命令的定义

(1) 打开记事本,并输入以下内容:

```
(vl-load-com)
'自动加载 VBA 程序的函数
(defun AutoVBALoad (app cmdliste flag/qapp strcmd)
  (setq qapp (strcat "\"" app "\""))
  (mapcar
    '(lambda (cmd/nom_cmd dot nodotcmd)
      (progn
        (setq dot (vl-string-search "." cmd))
        (if dot
          (setq nodotcmd (substr cmd ( + dot 2)))
          (setq nodotcmd cmd)
        )
        (setq nom_cmd (strcat "C: " nodotcmd))
        (if (member stat '(0 1 2 3 4 5 6 7))
          (setq strcmd
                (strcat"(vlax-add-cmd \"" nodotcmd "\" \'"
                       nom_cmd "\"" nodotcmd "\"" (itoa flag) " ))"))
        )
        (eval
          (read (strcat
                  "(defun " nom_cmd "(/app)"
                    "(if (setq app(fdvbfile " qapp "))"
                      "(progn(vla-runmacro (vlax-get-acad-object) (strcat app \"!" cmd "\"))"
                      strcmd
                      "(nodvbfile " qapp "))"
                  "(princ ))"
      )))))
    cmdliste
  )
  (princ)
)
(defun fdvbfile (app)
  (if (not (findfile app))
    (if (not (findfile (strcat app ".dvb"))) nil (strcat app ".dvb")) app)
)
(defun nodvbfile (filename)
  (princ (strcat "\n 文件 " filename "(.dvb) 在搜索路径文件夹中未找到。" ))
```

```
    (princ "\n 请检查支持文件的安装,然后重试。")
    (princ)
)
'自动加载 VBA 程序
(AutoVBALoad "ChangColor" '("CC") 3)
(AutoVBALoad "Zoom" '("ZP" "ZA" "ZE" "ZW") 1)
```

输入后保存为 AutoVBA.lsp 文件名。

(2) 在 AutoCAD 2005 中,选择"工具"|"加载应用程序"菜单项,在弹出的"加载"|"卸载应用程序"对话框中,选择刚保存的 AutoVBA.lsp 文件,单击"加载"按钮,然后单击"关闭"按钮,关闭该对话框。

该自定义函数使用格式为:

```
(AutoVBALoad "VBA 文件名" "过程名列表" 透明命令标志)
```

其中"透明命令标志"的含义如下:

- -1:不定义命令。
- 0、2、4、6:只定义命令,不定义为透明命令。
- 1、3、5、7:定义为透明命令。

其中 2-6 的选项与选定方式有关,一般情况下如果定义的透明命令包含选择内容,而为了让透明命令的选择项不干扰正常命令的选择项,可使用标志 3。

15.5.4　实例效果

1. 步骤 1 效果

(1) 当加载了 VBARun.lsp 文件后,用户可在屏幕上选择部分图元,然后在命令行中输入 VBARUNX,并按下 Enter 键,命令行会提示"宏名称:",输入 ChangeColor.dvb!CC,然后按下 Enter 键,此时可以看到刚才选定的对象已经变成了绿色。

(2) 再试一下运行程序前未选定对象的情况。在命令行中输入 VBARUNX,并按下 Enter 键,命令行会提示"宏名称:",输入 ChangeColor.dvb!CC,并按下 Enter 键,命令行会提示"选择对象:",选择一些对象后按下 Enter 键,此时选定的对象变成了绿色。

从运行效果看,经过这样处理的 VBA 程序,已经完成可以和 LISP 程序一样做到先选择后操作和先操作后选择的兼容运行。

2. 步骤 2 效果

当加载了 Zoom.lsp 文件后,用户可以试着使用 AutoCAD 的一些命令来操作图形中的对象,如使用 MOVE 命令来移动对象。

在命令行中输入 MOVE 并按下 Enter 键,在"选择对象"的提示下选定屏幕上的一些对象并按下 Enter 键,在"选择基点"的提示在屏幕上选择一个点作为基点,系统会提示"选择第二点"。这里为了观察定义的透明命令的效果,所以在提示下输入'ze 来显示所有图元,当输入'ze 后,可以看到屏幕按照 VBA 程序中所定义的"范围缩放"功能进行了缩放。这样,用户就可以在屏幕上选择移动的第二点。

用户可以用这种方式来试试其他几个透明命令 za、zp 和 zw。

3. 步骤 3 效果

步骤 3 是步骤 2 的精华浓缩。作者把整个透明命令的定义过程做成了一个函数，让用户可以轻松地定义自己的命令或透明命令，而且该函数也兼容了先选择后操作的功能。

15.5.5 小结

在本节中涉及的一些知识点，需要掌握的如下：

- 先选择后操作功能需要 VBA 程序与 LISP 程序相互配合。在 LISP 程序中，只有使用 ActiveX 的 vla-runmacro 函数，才能实现加载运行 VBA 程序时，图形中已选定对象不被取消。其他如 VbaRun 命令、vl-vbarun 函数等均不能达到这个要求。
- VBA 中可使用 PickfirstSelectionSet 属性来取得图形中已经选定的对象。
- 为了让程序兼容先选择后操作和先操作后选择两种功能，需要使用 PickfirstSelectionSet 属性判断选择集中是否有对象存在，如果不存在则由用户在图形中选定对象。在步骤 1 中已经提供了 PickFirstSSet 自定义函数，该函数可以完成这些功能。
- 透明命令可暂时中断当前命令执行过程而执行命令，它只是暂时中断当前命令，而不是取消当前命令，所以在透明命令完成后，当前的命令还可以继续执行。

15.6 更改 AutoCAD 应用程序的图标和标题栏

15.6.1 实例说明

本节使用了 Windows API(应用程序编程接口)来扩充 AutoCAD 的功能。要使用某个 API，首先必须在程序中声明该 API，这可以用 Declare 语句来完成，该语句包含以下几项信息：

- 动态链接库(DLL，其中包含要使用的过程)的名称。
- 出现在 DLL 中的过程名称。
- 需要在应用程序中使用的过程名称。
- 过程需要接收的参数。
- 返回值的数据类型(如果调用的过程是函数)。

如果将 Declare 语句放在模块中，并使用关键字 Public，则应用程序中的整个工程都可以使用该过程(函数)。

如果将 Declare 语句放在类模块或者用户窗体中，则只能使用关键字 Private，并只有该单元可以使用此过程(函数)。

AutoCAD 应用程序在 Windows 系统中被当作一个窗口来看待，而窗口是通过句柄来惟一识别的，所以利用 API 函数对句柄的操作，可以改变 AutoCAD 应用程序的一些特性。

本实例的运行界面如图 15.30 所示，其中包含了更改 AutoCAD 应用程序的标题、图标和还原 AutoCAD 应用程序的图标。

15.6.2　编程思路

两次使用 GetParent 函数可以获取 AutoCAD 应用程序的窗口句柄，在 AutoCAD 2004 以上版本可以直接使用 Application 对象的 HWND 属性来返回窗口的句柄。

使用 GetWindowText 函数返回窗口的标题，也可以使用 Application 对象的 Caption 属性来返回 AutoCAD 应用程序的窗口标题，使用 SetWindowText 函数设置窗口的标题。

使用 LoadImage 函数载入图标，使用 SendMessage 函数向窗口发送消息设置窗口的图标。

15.6.3　步骤

(1) 在 AutoCAD 中，打开 VBA 管理器，创建一个新工程，保存在适当的位置，进入 VBA 集成开发环境。

(2) 选择“插入”|“用户窗体”菜单项，向程序中添加一个用户窗体，添加一个 CommonDialog 控件(附加控件 Microsoft Common Dialog Control，version 6.0)，并在其中放置如图 15.31 所示的控件。

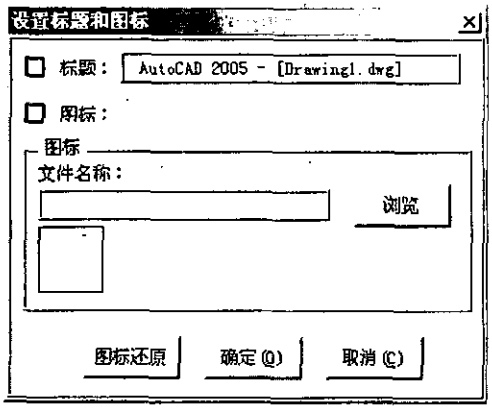


图 15.30　程序运行界面

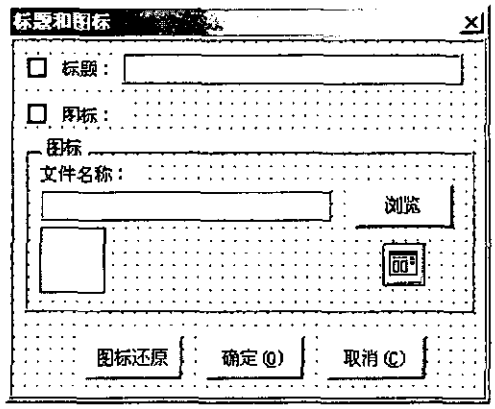


图 15.31　添加窗体控件

(3) 选择“插入”|“模块”菜单项，向程序中添加一个标准模块，添加所要使用的 API 函数和常量的声明：

```
Private Declare Function GetParent Lib "user32" (ByVal hwnd As Long) As Long
Private Declare Function SetWindowText Lib "user32" Alias "SetWindowTextA" (ByVal hwnd As Long,
    ByVal lpString As String) As Long
Private Declare Function LoadImage Lib "user32" Alias "LoadImageA" (ByVal hInst As Long,ByVal
    lpsz As String,ByVal un1 As Long,ByVal n1 As Long,ByVal n2 As Long,ByVal un2 As Long)
    As Long
Private Declare Function SendMessage Lib "user32" Alias "SendMessageA" (ByVal hwnd As Long,
    ByVal wMsg As Long,ByVal wParam As Long,lParam As Any) As Long
Private Const WM_SETICON = &H80
Private Const IMAGE_ICON = 1
Private Const LR_LOADFROMFILE = &H10
```

提示：如果已经安装了 VB 6.0，就可以在 Windows 中选择“开始”|“所有程序”|Microsoft Visual Studio 6.0|Microsoft Visual Studio 6.0 Tools|API Text Viewer，启动 API Viewer 程序，通过该程序获得 API 函数和常量的声明，如图 15.32 所示。

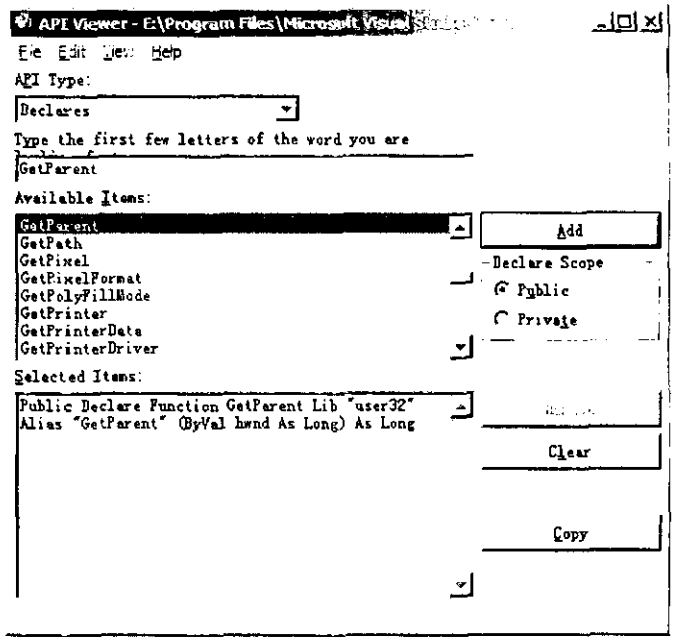


图 15.32 通过 API Viewer 获得 API 函数和常量的声明

(4) 在标准模块中添加宏的启动代码：

```
Public Sub SetTitleAndIcon( )
    UserForm1.Show
End Sub
```

(5) 在窗体的初始化事件中，返回 AutoCAD 应用程序的窗口标题，其实现代码为：

```
Private Sub UserForm_Initialize( )
    TextBox1.Text = Application.Caption
End Sub
```

(6) 窗体中“浏览”按钮的单击事件，用于显示标准对话框，获取用户选择的文件名称，并将选择的结果添加到文本框中，其实现代码为：

```
Private Sub CommandButton1_Click( )
    Dim s As String

    '显示打开文件对话框
    CommonDialog1.Filter = "图标 (*.ico)|*.ico"
    CommonDialog1.ShowOpen
    TextBox2.Text = CommonDialog1.FileName
    If TextBox2.Text = "" Then Exit Sub
```

```
    '载入图标
    Image1.Picture = LoadPicture(TextBox2.Text)
End Sub
```

(7) 窗体中“确定”按钮的单击事件，用于设置标题和图标，其实现代码为：

```
Private Sub CommandButton2_Click( )
    If CheckBox1.Value = True Then
        SetTitle TextBox1.Text
    End If
    If CheckBox2.Value = True Then
        SetIcon TextBox2.Text
    End If
End Sub
```

上面代码中使用了自定义函数 SetTitle，用来设置窗口的标题，自定义函数 SetIcon，用来设置窗口的图标，自定义函数 ReSetIcon，还原 AutoCAD 应用程序的窗口图标，其实现代码为：

```
Public Sub SetTitle(ByVal Title As String)
    Dim AcadHwnd As Long
    Dim hIcon As Long

    '获取 AutoCAD 应用程序的窗口句柄
    AcadHwnd = GetParent(GetParent(ThisDrawing.hwnd))
    '设置标题
    SetWindowText AcadHwnd,Title
End Sub

Public Sub SetIcon(ByVal FileName As String)
    Dim AcadHwnd As Long
    Dim hIcon As Long

    '获取 AutoCAD 应用程序的窗口句柄
    AcadHwnd = GetParent(GetParent(ThisDrawing.hwnd))
    '从文件载入图标，16 * 16 大小
    hIcon = LoadImage(0&,FileName,IMAGE_ICON,16,16,LR_LOADFROMFILE)
    If hIcon <> 0 Then
        '发送消息，设置图标
        Call SendMessage(AcadHwnd,WM_SETICON,0,ByVal hIcon)
    End If
End Sub

Sub ReSetIcon( )
    Dim AcadHwnd As Long

    '获取 AutoCAD 应用程序的窗口句柄
    AcadHwnd = GetParent(GetParent(ThisDrawing.hwnd))
    '发送消息，设置图标
```

```
    Call SendMessage(AcadHwnd,WM_SETICON,0,ByVal 0)
End Sub
```

(8) 最后，在窗体中“取消”按钮的单击事件中，添加用于退出程序的代码：

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

15.6.4 实例效果

(1) 在 VBA 集成开发环境中，按下 F5 键运行程序，系统会弹出如图 15.33 所示的“设置标题和图标”对话框。

(2) 在“设置标题和图标”对话框中，选中“标题”复选框，输入标题的名称，如“AutoCAD 2005-明经通道”。

(3) 在“设置标题和图标”对话框中，选中“图标”复选框，单击“浏览”按钮，系统会弹出如图 15.34 所示的“选择图标文件”对话框，用户可以从中选择一个图标文件，单击“打开”按钮就可以将文件名称添加到“设置标题和图标”对话框的文本框中，如图 15.35 所示。

(4) 在“设置标题和图标”对话框中，单击“确定”按钮，就可以更改 AutoCAD 应用程序的标题栏和图标，如图 15.36 所示。

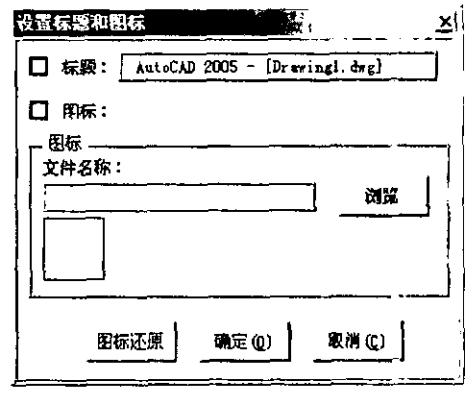


图 15.33 程序运行界面

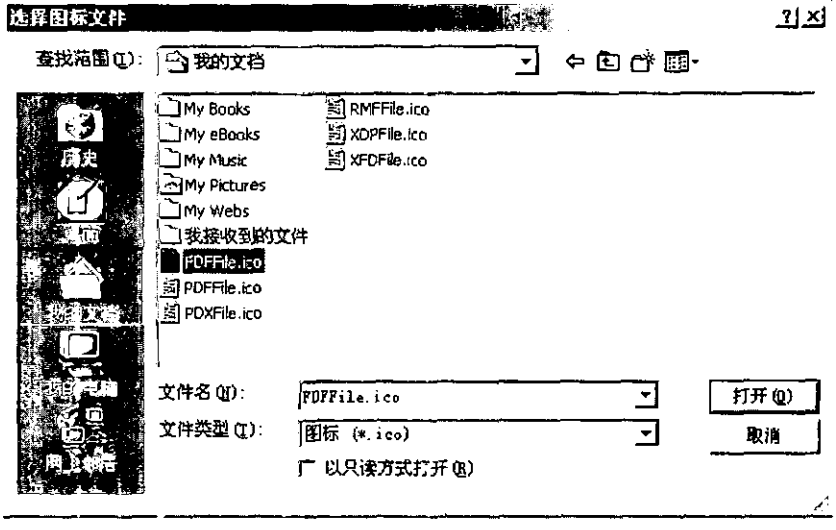


图 15.34 选择图标文件

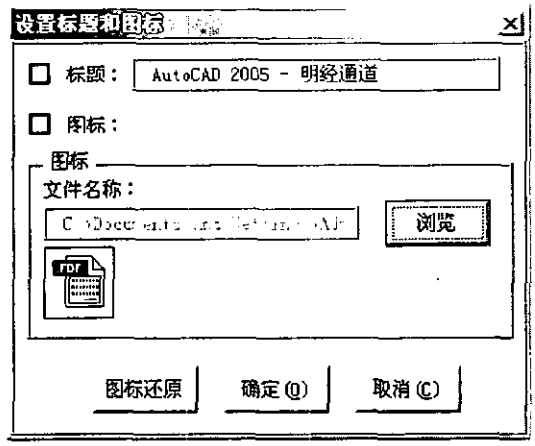


图 15.35　窗体输入结果

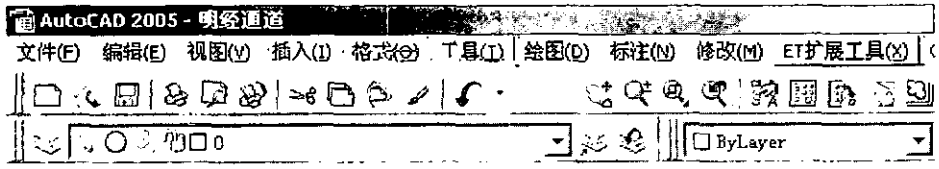


图 15.36　更改 AutoCAD 图标和标题栏的结果

15.6.5　小结

本节则介绍了使用标准打开对话框实现文件的选择，演示了 API 在 AutoCAD VBA 中的应用。同时通过窗口的句柄，利用 API 函数设置了应用程序新的标题和图标。

15.7　发布 VBA 应用程序

15.7.1　实例说明

作为开发者，开发程序大部分的目的都是将程序发布给别人使用，既然说是发布，就不能简单地把菜单文件和 DVB 工程文件直接给用户，让用户加载菜单、设置自动加载、添加支持文件路径，进行一系列的操作来完成程序的配置。

对于一个商业化的程序来说，安装包的好坏从一个侧面反映了该软件的技术水平，而且安装程序将成为应用程序给用户的第一印象，因此，在完成程序之后，创建一个界面友好、功能强大、使用方便的安装包来发布程序非常有必要。

本节的程序使用 InstallShield 6.22 专业版来完成所有的工作，InstallShield 是世界上应用非常广泛的安装制作工具，在 Visual Studio 6.0 中，就包含了 InstallShield 的一个简化版，但是这里使用的是包含远东语言包的 InstallShield 6.22 专业版（默认情况下只能制作英文的安装程序）。

15.7.2 编程思路

作为一个 AutoCAD 二次开发应用程序的安装包，需要完成下面的工作：

- 将程序文件、资源文件和其他支持文件复制到客户的计算机中。
- 在客户的 AutoCAD 软件中创建一个配置，用于保存应用程序的各种信息。
- 将程序文件和其他支持文件所在的路径添加到客户计算机中 AutoCAD 的支持文件搜索路径中。
- 在 AutoCAD 启动时，加载指定的程序文件和指定的菜单。
- 其他的相关工作，例如设置模板文件的位置，修改 AutoCAD 启动时的图标、标题栏文字等。

之所以选择 InstallShield 来作为安装制作的工具，一方面由于 InstallShield 使用方便，最主要的还是 InstallShield 可以完全使用脚本语言来进行安装程序的定制，具有极大的灵活性，这对于创建 AutoCAD 二次开发应用程序的安装包无疑是最合适的。

15.7.3 步骤

(1) 在创建安装包之前，应该在多种操作系统和 AutoCAD 平台上对程序文件进行测试，确保程序没有错误。创建一个特定的文件夹，用于保存安装包所要使用的文件，在其中放置了两个文件夹：VBA Macros 文件夹中放置了所有的 DVB 程序文件，Templates 文件夹包含了应用程序所要使用的样板文件。

为简单起见，从本书所介绍的实例中选出 12 个，将其 DVB 工程文件直接放在 VBA Macros 文件夹中，如图 15.37 所示。

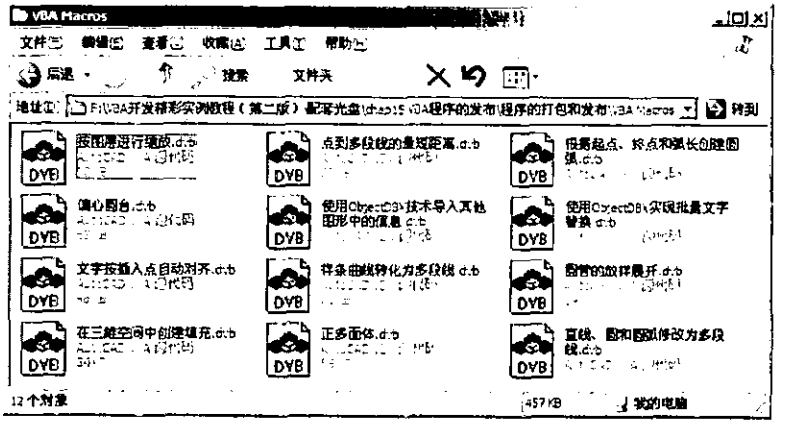


图 15.37 安装包中的 DVB 工程文件

(2) 打开 AutoCAD 2005 的菜单文件 acad.mnu(其位置可以在 AutoCAD 的“选项”对话框中查看，如图 15.38 所示)，按下组合键 Ctrl+A，然后按下组合键 Ctrl+C，将其所有内容复制到剪贴板中，关闭 acad.mnu 文件。

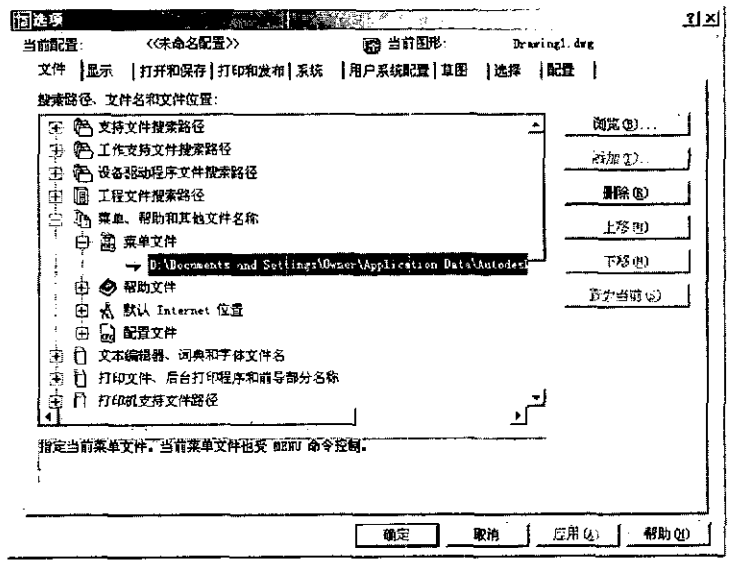


图 15.38　AutoCAD 2005 中菜单文件的位置

(3) 使用记事本应用程序创建一个新的文本文件，按下 Ctrl＋V 组合键，找到下拉菜单项保存的位置(可以查找 POP11 字符串)，在其后面添加下面的内容(粗体是新增加的内容)：

```
*** POP11
** HELP
ID_MnHelp    [帮助(&H)]
ID_Help      [帮助(&H)\tF1]'_help
ID_INFOPAL   [$(if,$(and,$(getvar,assiststate),1),!.)信息选项板(&I)\tCtrl+5]$M=
    $(if,$(and,$(getvar,assiststate),1),'_assistclose,^C^C_assist)
ID_DevHelp   [开发人员帮助(&V)]^C^C^P(help "acad_dev" "") ^P
ID_WhatsNew  [$(if,$(eq,$(getvar,nfwstate),0),～)新功能专题研习(&W)]^C^C^P(progn (help
    "acad_nfw")(princ)) ^P

             [--]
ID_OnlineRes  [->联机资源(&R) ]
ID_ProdSuppSafe [产品支持(&P)]^C^C^P_ai_product_support_safe ^P
ID_TrainingSafe [培训(&T)]^C^C^P_ai_training_safe ^P
ID_CustomSafe    [自定义(&C)]^C^C^P_ai_custom_safe ^P
ID_AUGI      [<-Autodesk 国际用户组(&A)]^C^C^P(command "_browser" (findfile "augi.htm")) ^P
             [--]
ID_About     [关于(&O)]'_about

//自定义的菜单
*** POP12
** MJ

ID_MJ [明经通道(&M)]
ID_MJ [按图层缩放(&Z)]^C^C-VBARUN 按图层进行缩放.dvb!ZoomToLayer
             [--]
```

```
ID_MJ [起点、终点和弧长创建圆弧(&A)]^C^C-VBARUN 根据起点、终点和弧长创建圆弧.dvb!CreateArc
ID_MJ [修改为多段线(&M)]^C^C-VBARUN 直线、圆和圆弧修改为多段线.dvb!mdlMain.TransformToPolyline
ID_MJ [样条曲线转化为多段线(&T)]^C^C-VBARUN 样条曲线转化为多段线.dvb!Spline2Pline
ID_MJ [导入其他图形中的信息(&I)]^C^C-VBARUN 使用 ObjectDBX 技术导入其他图形中的信息.
      dvb!Main
ID_MJ [圆管的放样展开(&E)]^C^C-VBARUN 圆管的放样展开.dvb!UnwrapPipe
ID_MJ [点到多段线的最短距离(&D)]^C^C-VBARUN 点到多段线的最短距离.dvb!GetDistancePt2Poly
            [--]
ID_MJ [三维空间创建填充(&H)]^C^C-VBARUN 在三维空间中创建填充.dvb!CreateHatch
ID_MJ [偏心圆台(&C)]^C^C-VBARUN 偏心圆台.dvb!Main
ID_MJ [正多面体(&P)]^C^C-VBARUN 正多面体.dvb!Main
            [--]
ID_MJ [对齐文字(&A)]^C^C-VBARUN 文字按插入点自动对齐.dvb!TextAlign
ID_MJ [批量文字替换(&R)]^C^C-VBARUN 使用 ObjectDBX 实现批量文字替换.dvb!ReplaceText
```

将该文件保存为 mjtd.mnu，放置在安装包的根目录下。

(4) 大型商业程序在安装的过程中都会有许可协议、注意事项等内容，我们的安装包同样会有这些东西。在安装包的根目录下添加一个名为 license.txt 的文件，用于提供许可协议的内容，如图 15.39 所示。

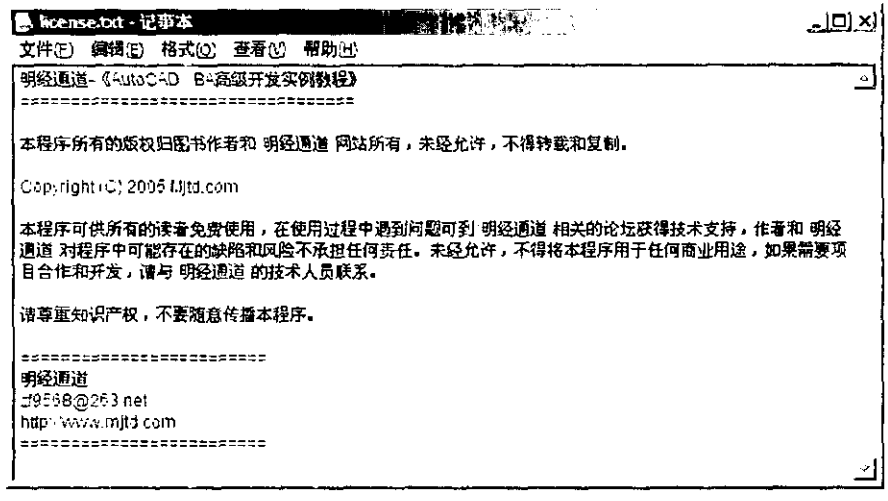


图 15.39 许可协议

在安装包的根目录下添加一个名为 infolist.txt 的文件，用于提示用户在安装期间关闭 AutoCAD 和相关的应用程序，如图 15.40 所示。

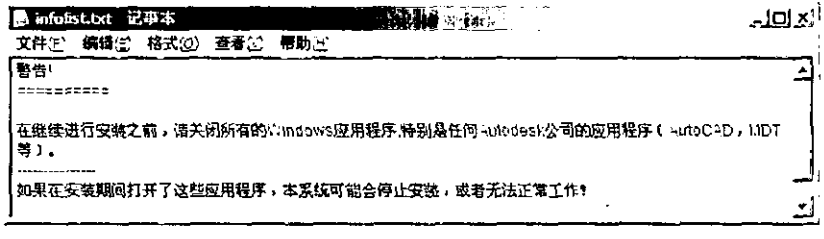


图 15.40 警告

(5) 打开记事本应用程序，在其中添加下面的内容：

```
'自定义的启动函数
(defun-q MJTDSTARTUP ( )
    (prompt "\n 明经通道《AutoCAD VBA 高级开发实例教程》安装包示例已经加载成功……\n 祝您读
        书愉快……")
    (command "_-vbarun" "InitializeEvents")
);defun-q

(setq S:: STARTUP (append S:: STARTUP MJTDSTARTUP))
```

将其保存为 acaddoc.lsp 文件，放置在安装包的根目录下。该文件中的 STARTUP 函数会在新建图形文件或者打开一个图形文件时被执行，该函数在 AutoCAD 启动时自动执行 acad.dvb 文件(会在后面的步骤中创建)中的 InitializeEvents 宏。

(6) 准备两个文件 setup.bmp 和 mjtd.ico，前者用于提供安装开始时的程序背景，后者将会作为应用程序的图标。此时，安装包的根目录下包含了 6 个文件和 2 个文件夹，如图 15.41所示。

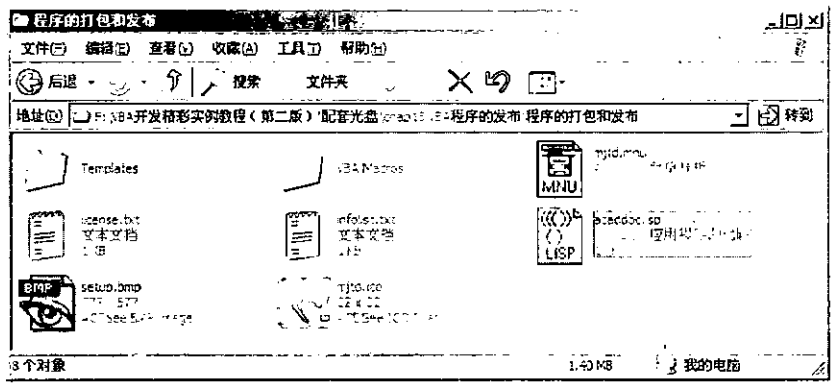


图 15.41　安装包中的文件

(7) 由于 AutoCAD 2002 以上的版本中，使用/P 选项启动会将启动的配置文件修改为当前配置，这样在启动自己的应用程序之后再执行 AutoCAD 自身的快捷方式，仍然会进入自己的应用程序，这对成熟的安装程序来说实在不应该出现，为了解决这个问题，必须专门创建一个 DVB 工程文件在自己的应用程序退出时将当前配置修改为 AutoCAD 默认的配置。

在 DVB 工程中对 AutoCAD 的应用程序层事件 BeginQuit 进行处理，在该事件中恢复 AutoCAD 默认的配置。对于 AutoCAD 2005 中文版，AutoCAD 当前的配置保存在注册表 HKEY_CURRENT_USER\Software\Autodesk\AutoCAD\R16.1\ACAD-301：804\Profiles 的默认键值中(加粗的部分随 AutoCAD 版本的不同而有所变化)，如图 15.42 所示。只要在退出 AutoCAD 之前将该键值清空，即可保证 AutoCAD 快捷方式能够用默认的“未命名配置”启动。

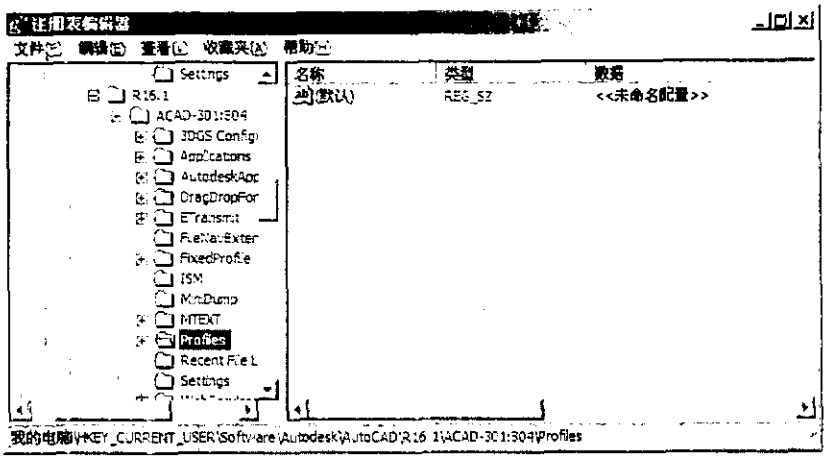


图 15.42 AutoCAD 当前配置保存的位置

(8) 运行 AutoCAD 2005,打开 VBA 管理器,创建一个新工程,保存在安装包的根目录中,命名为 acad.dvb(该文件能被 AutoCAD 在启动时自动加载)。进入 VBA 集成开发环境,向工程中添加一个类模块,在属性窗口中将其名称修改为 EventClassModule,在其中添加一些 API 函数和常量的声明:

```
'打开注册表中指定的键
Private Declare Function RegOpenKey Lib "advapi32.dll" Alias "RegOpenKeyA" _
    (ByVal hKey As Long,ByVal lpSubKey As String,phkResult As Long) As Long
'修改注册表中某个键的值
Private Declare Function RegSetValue Lib "advapi32.dll" Alias "RegSetValueA" _
    (ByVal hKey As Long,ByVal lpSubKey As String,ByVal dwType As Long,_
    ByVal lpData As String,ByVal cbData As Long) As Long
'关闭打开的键
Private Declare Function RegCloseKey Lib "advapi32.dll" (ByVal hKey As Long) As Long

Private Const HKEY_CURRENT_USER = &H80000001
Private Const REG_SZ = 1
```

(9) 定义一个常量 App 选项,用于启动 AutoCAD 应用程序层事件:

```
Public WithEvents App As AcadApplication
```

(10) 在代码窗口的左上角的列表中选择 App 选项,右上角的列表中选择 BeginQuit 选项,添加下面的代码:

```
Private Sub App_BeginQuit(Cancel As Boolean)
    '获得 AutoCAD 的版本
    Dim strAcadVersion As String
    strAcadVersion = Application.Version

    Dim hKey As Long,ret As Long
    If StrComp(Left(strAcadVersion,4),"15.0",vbTextCompare) = 0 Then
```

```
        'AutoCAD 2002 版本
        ret = RegOpenKey(HKEY_CURRENT_USER,"Software\Autodesk\AutoCAD\R15.0\ACAD-1:804\
              Profiles",hKey)
        If ret = 0 Then
            RegSetValue HKEY_CURRENT_USER,"Software\Autodesk\AutoCAD\R15.0\ACAD-1:409\
                Profiles",_
                        REG_SZ,"",0
            RegCloseKey hKey
        End If
    ElseIf StrComp(Left(strAcadVersion,4),"16.0",vbTextCompare) = 0 Then
        'AutoCAD 2004 版本
    ElseIf StrComp(Left(strAcadVersion,4),"16.1",vbTextCompare) = 0 Then
        'AutoCAD 2005 版本
        ret = RegOpenKey(HKEY_CURRENT_USER,"Software\Autodesk\AutoCAD\R16.1\ACAD-301:804\
              Profiles",hKey)
        If ret = 0 Then
            RegSetValue HKEY_CURRENT_USER,"Software\Autodesk\AutoCAD\R16.1\ACAD-301:409\
                Profiles",_
                        REG_SZ,"",0
            RegCloseKey hKey
        End If
    ElseIf StrComp(Left(strAcadVersion,4),"16.2",vbTextCompare) = 0 Then
        'AutoCAD 2006 版本
    End If
End Sub
```

上面的代码首先判断当前 AutoCAD 的版本，根据版本确定注册表中保存当前 AutoCAD 默认配置的位置，将对应的键值设置为空字符串。

(11) 在“工程资源管理器”窗口中双击 ThisDrawing 图标，打开其代码窗口，在其中添加启用应用程序层事件的代码：

```
Dim X As New EventClassModule

Public Sub InitializeEvents( )
    Set X.App = Application
End Sub
```

注意，这里不能使用 ThisDrawing.Application 来获得应用程序对象，因为在 BeginQuit 事件发生时所有的文档已经被关闭。

前面已经介绍，InitializeEvents 宏会在 AutoCAD 启动时被自动执行，启用 AutoCAD 应用程序层事件，确保应用程序退出时恢复 AutoCAD 默认的配置。

(12) 启动 InstallShield，系统会显示如图 15.43 所示的界面，双击 Project Wizard 图标，使用向导来创建新的安装工程。

(13) 系统弹出如图 15.44 所示的对话框，提示用户选择工程类型，在此选中 Standard Setup Project 单选按钮，单击 OK 按钮。

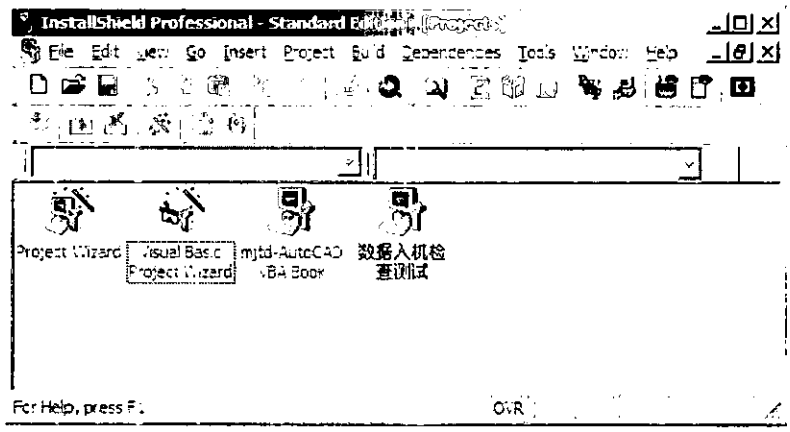


图 15.43 InstallShield 启动界面

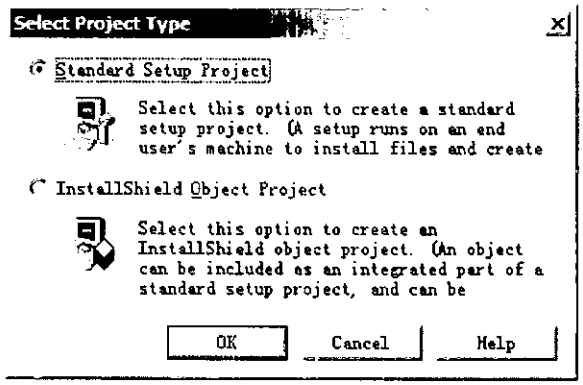


图 15.44 选择工程类型

(14) 系统弹出如图 15.45 所示的对话框,在其中分别输入项目名称、应用程序名称、公司名称、应用程序版本,可执行文件名称这一栏暂时不填,在后面需要专门设置。单击"下一步"按钮。

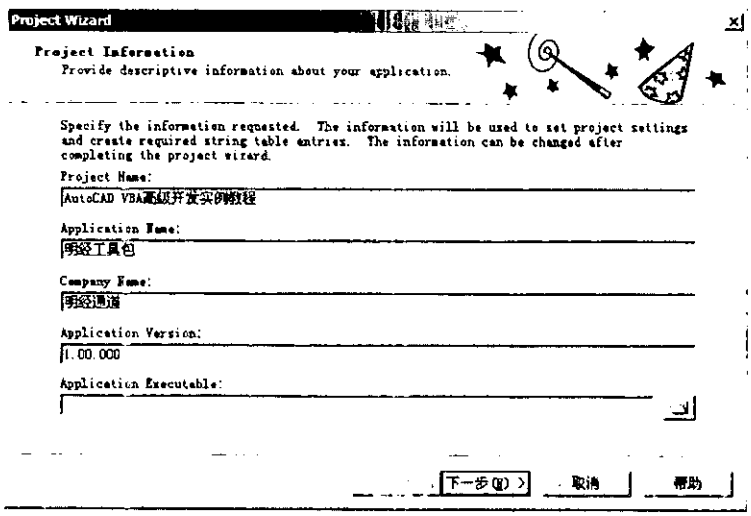


图 15.45 输入工程信息

(15) 系统会弹出如图 15.46 所示的对话框，仅选中 Cinese (PRC)复选框，指定安装工程的语言为简体中文，单击“下一步”按钮。

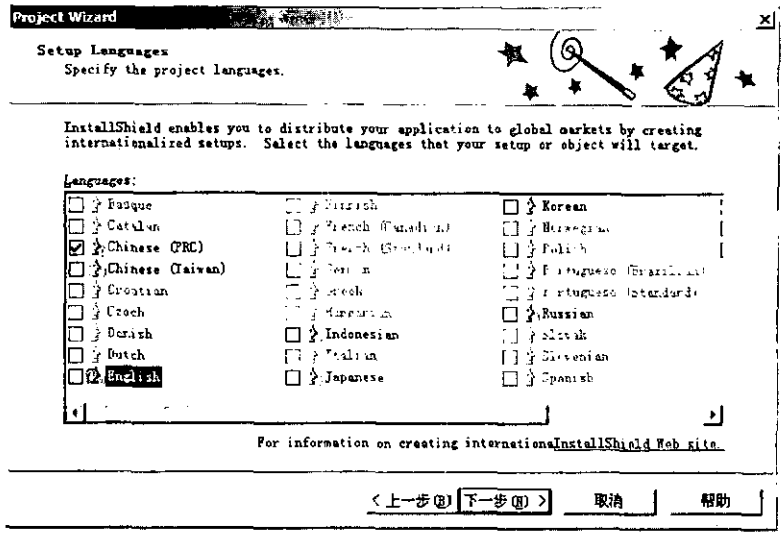


图 15.46　指定安装工程的语言

(16) 系统会弹出如图 15.47 所示的对话框，在左边的列表中分别选择 Tutorial 和 Examples 选项，然后单击 Delete 按钮将其删除，因为我们的程序仅提供主程序，未提供示例和教程文件。单击“下一步”按钮。

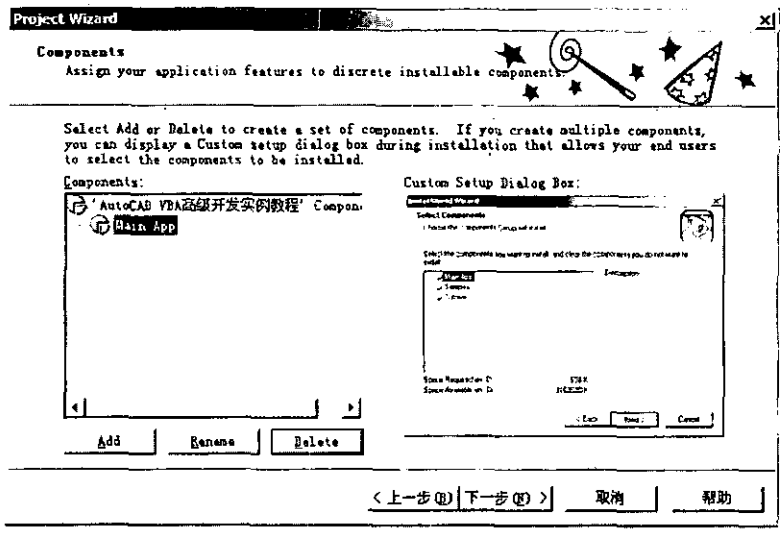


图 15.47　选择工程包含的组件

(17) 系统弹出如图 15.48 所示的对话框，分别选择 App Self Reg DLLs、Example Files 和 Tutorial Files 选项，单击 Delete 按钮将其删除，单击“下一步”按钮。

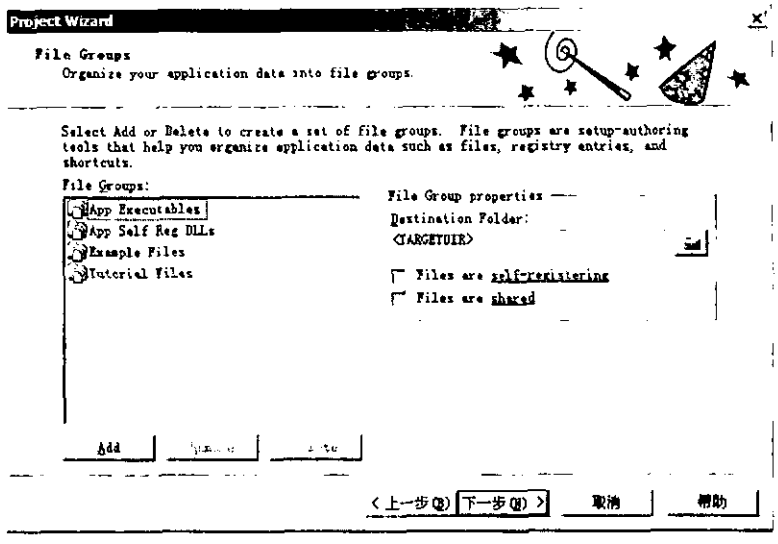


图 15.48 修改文件组

(18) 系统弹出如图 15.49 所示的对话框，由于这里无法添加文件夹，故不在此进行任何操作，直接单击“下一步”按钮。

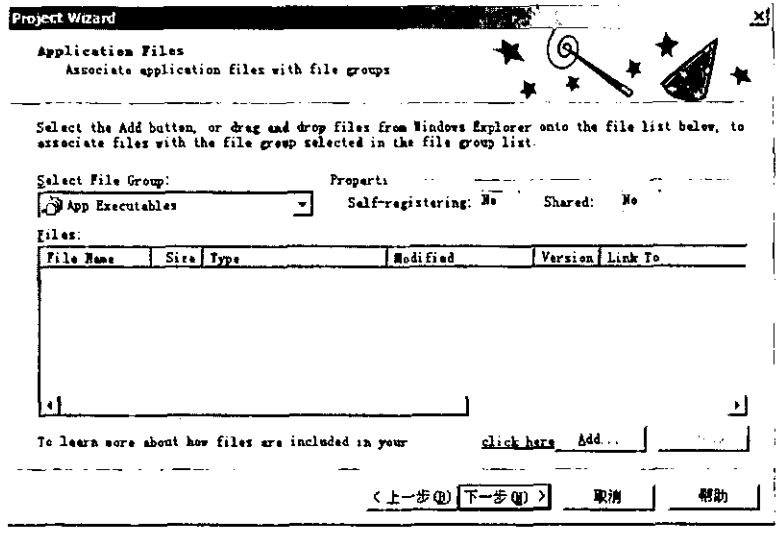


图 15.49 跳过添加文件的步骤

(19) 系统弹出如图 15.50 所示的对话框，跳过文件组和组件关联的步骤(因为还没有添加文件组中的文件)，单击“下一步”按钮。

(20) 系统会弹出如图 15.51 所示的对话框，这里没有我们需要的对象，直接单击“下一步”按钮。

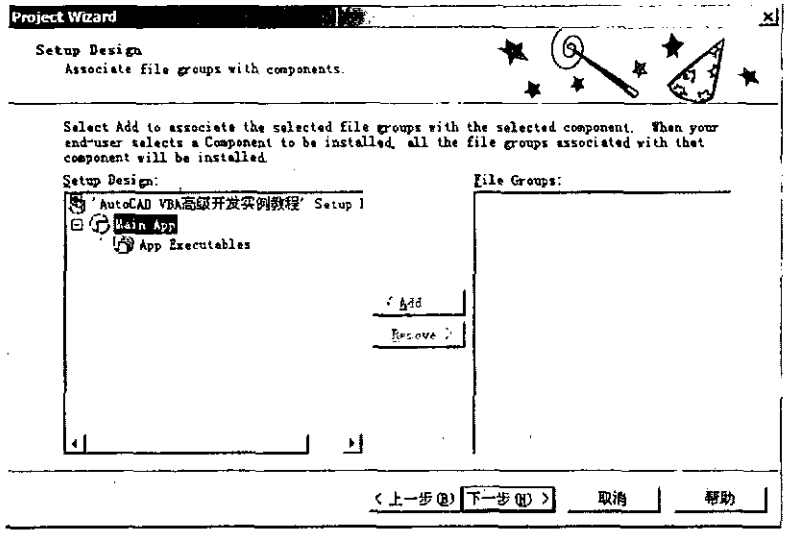


图 15.50　跳过文件组和组件的关联操作

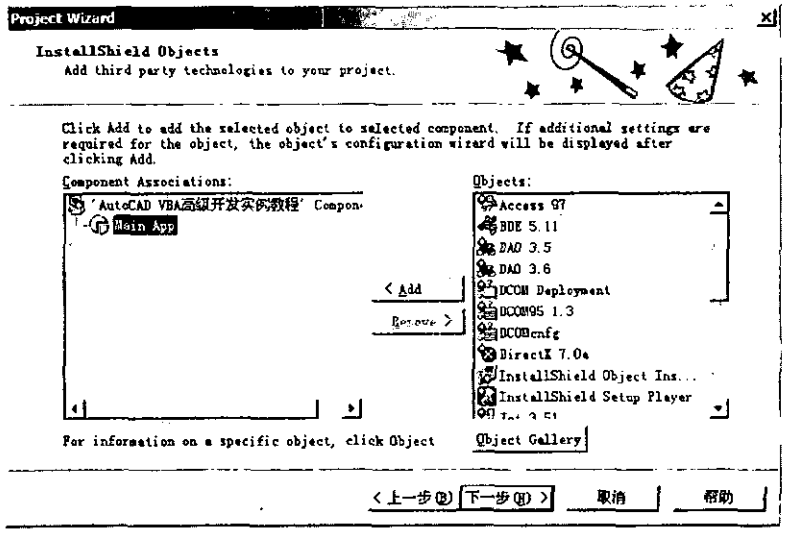


图 15.51　未添加对象

(21) 系统弹出如图 15.52 所示的对话框，由于 AutoCAD 二次开发应用程序的快捷方式比较特殊，故不在此添加，直接单击“下一步”按钮。

(22) 系统会弹出如图 15.53 所示的对话框，选中除 Setup Type 和 Custom Setup 之外的所有选项，单击“下一步”按钮。

(23) 系统弹出如图 15.54 所示的对话框，给出当前安装工程的特性的总结，单击“完成”按钮，系统便会创建该安装工程的框架。

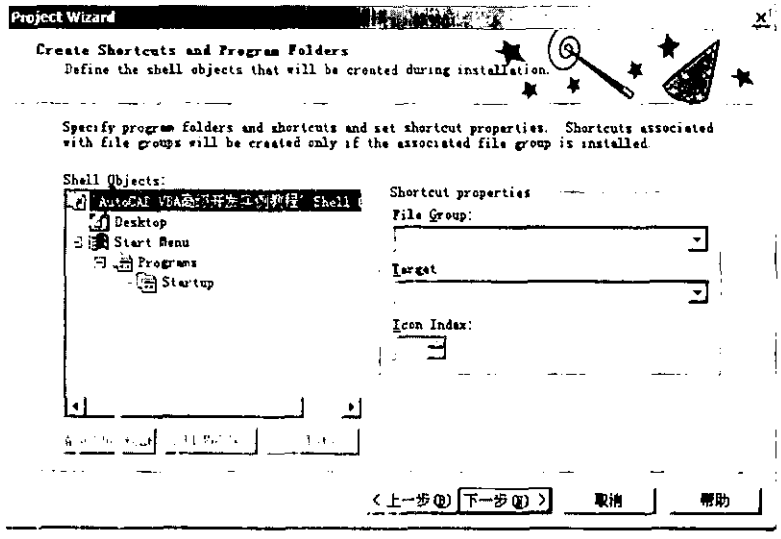


图 15.52 不添加快捷方式

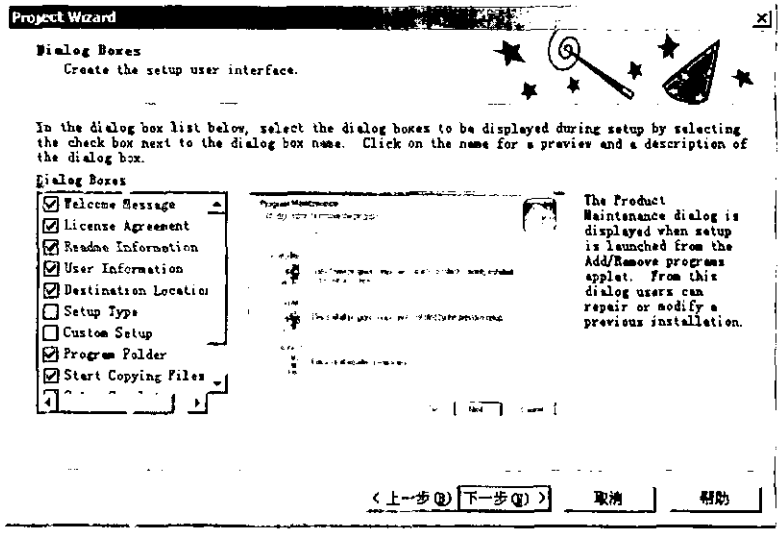


图 15.53 创建安装工程的用户界面

(24) 在 InsallShield 的主界面左侧的面板中(见图 15.55),切换到 Setup Files 选项卡,双击 Language Independent 选项,选择右侧界面出现的 Setup. bmp 文件,按下 Delete 键将其删除,然后在空白区域右击,从弹出的快捷菜单中选择 Insert Files 菜单项,在弹出的对话框中选择安装包根目录下的 setup. bmp 文件,将其插入到工程中。

(25) 选择左侧面板 Chinese (PRC)中的 Operating System Independent 节点,在右侧的空白区域右击,从弹出的快捷菜单中选择 Insert Files 菜单项,在弹出的对话框中选择安装包根目录下的 infolist. txt 和 license. txt 文件,将其插入到工程中。

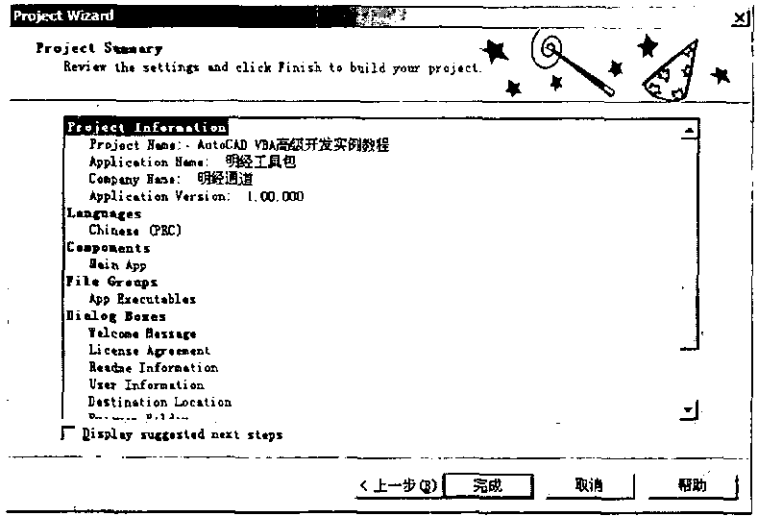


图 15.54　显示工程概况

(26) 在左侧的面板中切换到 File Groups 选项卡，选择 App Executables 中的 Static File Links 节点，在右侧的空白区域右击，从弹出的快捷菜单中选择 Launch Explore 菜单项，在系统打开的浏览器窗口中定位到安装包的根目录，将其中的 Templates 和 VBA Macros 文件夹，以及 mjtd. mnu 和 acaddoc. lsp 文件拖动到 InstallShield 中，就将这些文件添加到安装工程的链接文件列表中。在安装进行时，这些文件会被复制到用户指定的文件夹中。

(27) 在左侧的面板中切换到 Resources 选项卡，展开 Shell Objects 节点，右击 Desktop 选项，从弹出的快捷菜单中选择 New|Shortcut 菜单项，输入“明经通道—工具包”作为快捷方式的名称。

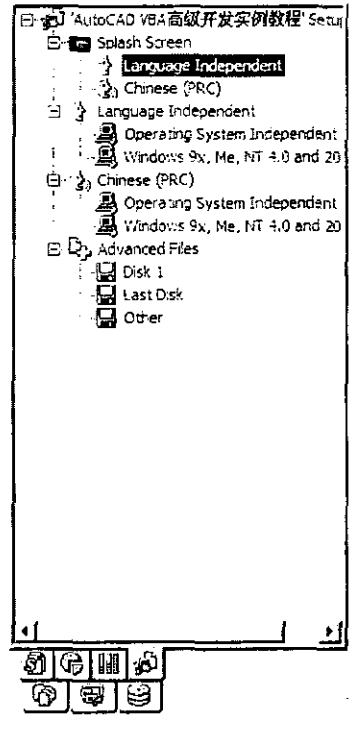


图 15.55　InstallShield 的工作面板

在右侧的任意字段上双击，系统会弹出 Shortcut Propertied 对话框，在 Target 选项卡中进行如图 15.56 所示的设置。其中，Target 文本框指定了快捷方式属性窗口中“目标”文本框中的内容，这里指定的内容是 AutoCAD 2005 的快捷方式，但是由于此时不知道客户计算机中 AutoCAD 安装的位置，只能用一个变量来代替，此变量将在脚本执行过程中获得具体的路径；Parameters 文本框指定应用程序的启动参数，这里使用了/p 来表示启动后进入指定的配置；Start 文本框的内容指定了应用程序的起始位置，该起始位置将被设置为应用程序的安装路径，用 InstallShield 的系统变量 TARGETDIR 来指定。

在对话框的 Settings 选项卡中，指定快捷方式图标的位置，在 Icon 文本框中指定图标在用户的计算机中的绝对位置，然后在下面的 Icon 列表框中输入 0，如图 15.57 所示。

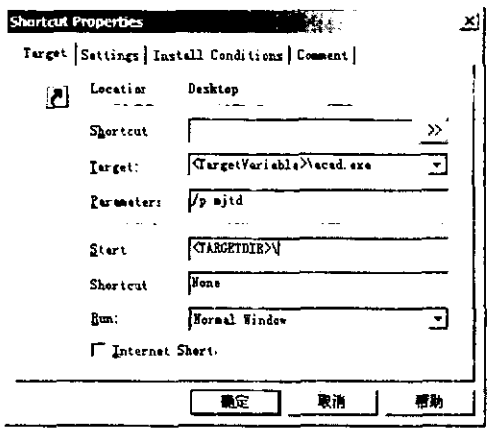


图 15.56 设置应用程序目标属性

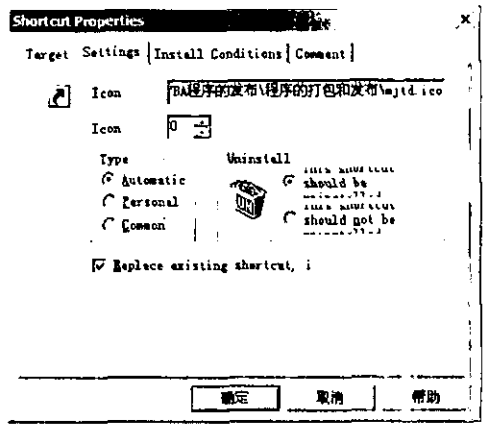


图 15.57 设置快捷方式图标

在 Install Conditions 选项卡中，选中 App Executables 复选框，如图 15.58 所示。

在 Comment 选项卡中，可以输入关于快捷方式的注释，如图 15.59 所示。单击“确定”按钮完成快捷方式的设置。

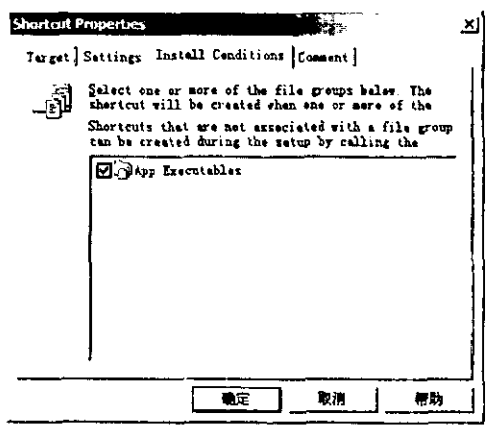


图 15.58 创建快捷方式和文件组的关联

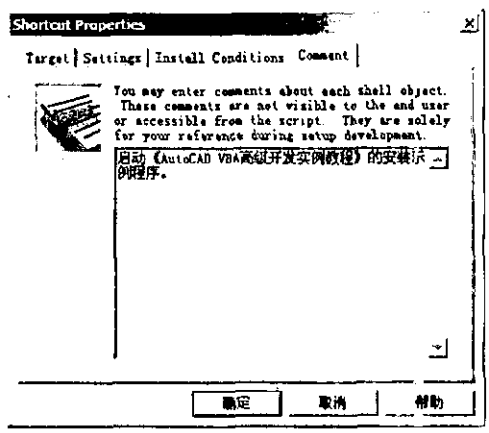


图 15.59 添加快捷方式的注释

(28) 使用同样的方法，在 Start Menu 节点中创建一个快捷方式，这样安装包运行的时候会同时在“开始”菜单中创建一个应用程序，该快捷方式的设置与桌面快捷方式完全相同。此时，Resources 选项卡包含了如图 15.60 所示的项目。

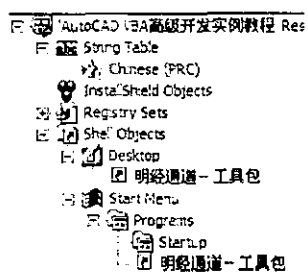


图 15.60 创建两个快捷方式的位置

(29) 在左侧的面板中切换到 Scripts 选项卡，打开脚本代码窗口。由于 InstallShield 中的代码比较零乱(相对于传统的编程习惯)，为了便于大家学习，这里列出带有详细注释的主脚本文件的所有代码(相对于原来的脚本

、文件，手工添加的代码用粗体标识）：

```
////////////////////////////////////////////////////////////
//
//  File Name: Setup.rul
//
//  Description: InstallShield script
//
//  Comments: This script was generated based on the selections you made in
//                the Project Wizard. Refer to the help topic entitled "Modify
//                the script that the Project Wizard generates" for information
//                          on possible next steps.
//
////////////////////////////////////////////////////////////


//Include header files

#include "ifx.h"

//////////////////string defines//////////////////////
#define MAX_KEY_LEN                512     //键的最大长度

////////////////installation declarations///////////////

//---- DLL function prototypes ----


    //your DLL function prototypes


//---- script function prototypes ----
prototype IsAcadInstalled(BYREF string);
prototype AddAcadProfile(string,string);
prototype DeleteAcadProfile( );

    //your script function prototypes

    //your global variables
string szAcadPath;              //AutoCAD 2005 的安装路径
string szAcadProfilePath;         //注册表中保存配置的路径


////////////////////////////////////////////////////////////
//
//  FUNCTION:   OnFirstUIBefore
```

```
//
//  EVENT:      FirstUIBefore event is sent when installation is run for the first
//                time on given machine. In the handler installation usually displays
//                UI allowing end user to specify installation parameters. After this
//                function returns,ComponentTransferData is called to perform file
//                transfer.
//
////////////////////////////////////////////////////////////
function OnFirstUIBefore( )
    number nResult,nSetupType;
    string szTitle,szMsg;
    string szLicenseFile,szQuestion;
    string szName,szCompany,szSerial;
    string szFile;
    string szDir;
    string szfolder;
    LIST    listStartCopy;
    LIST    list;
begin
    //TO DO: if you want to enable background,window title,and caption bar title

    //SetTitle( @TITLE_MAIN,24,WHITE );
    //SetTitle( @TITLE_CAPTIONBAR,0,BACKGROUNDCAPTION );
    //Enable( FULLWINDOWMODE );
    //Enable( BACKGROUND );
    //SetColor(BACKGROUND,RGB (0,128,128));

    TARGETDIR = PROGRAMFILES ^@COMPANY_NAME ^@PRODUCT_NAME;
    szDir = TARGETDIR;
    SHELL_OBJECT_FOLDER = @FOLDER_NAME;
    szName    = "";
    szCompany = "";
    szSerial = "";

Dlg_Start:
    //判断 AutoCAD 2002 是否已经安装
    if ( !IsAcadInstalled(szAcadPath)) then
        return -1;
    endif;

Dlg_SdWelcome:
    szTitle = "安装向导 - 开始";
    szMsg   = "欢迎你学习安装包示例程序!";
    nResult = SdWelcome( szTitle,szMsg );
    if (nResult = BACK) goto Dlg_Start;
```

```
Dlg_SdLicense:
    szLicenseFile = SUPPORTDIR ^ "license.txt";
    szTitle     = "安装向导 - 许可协议";
    szMsg       = "请阅读许可协议";
    szQuestion = "是否同意许可协议?";
    nResult     = SdLicense( szTitle,szMsg,szQuestion,szLicenseFile );
    if (nResult = BACK) goto Dlg_SdWelcome;

Dlg_SdShowInfoList:
    szFile = SUPPORTDIR ^ "infolist.txt";
    list = ListCreate( STRINGLIST );
    ListReadFromFile( list,szFile );
    szTitle = "安装向导 - 注意事项";
    szMsg     = "在安装前请阅读下面的注意事项";
    nResult = SdShowInfoList( szTitle,szMsg,list );
    ListDestroy( list );
    if (nResult = BACK) goto Dlg_SdLicense;

Dlg_SdRegisterUserEx:
    szMsg   = "安装向导 - 身份验证";
    szTitle = "输入序列号";
    nResult = SdRegisterUserEx( szTitle,szMsg,szName,szCompany,szSerial );
    if (nResult = BACK) goto Dlg_SdShowInfoList;
    //检查序列号是否正确
        while (StrCompare(szSerial,"8808-123456") ! = 0)
            MessageBox( "无效的序列号!",SEVERE);
            nResult = SdRegisterUserEx( szTitle,szMsg,szName,szCompany,szSerial );
        endwhile;

Dlg_SdAskDestPath:
    szTitle = "安装向导 - 安装路径";
    szMsg   = "选择安装示例程序的路径";
    nResult = SdAskDestPath( szTitle,szMsg,szDir,0 );
    TARGETDIR = szDir;
    if (nResult = BACK) goto Dlg_SdRegisterUserEx;

Dlg_ObjDialogs:
    nResult = ShowObjWizardPages(nResult);
    if (nResult = BACK) goto Dlg_SdAskDestPath;

Dlg_SdSelectFolder:
    szfolder = SHELL_OBJECT_FOLDER;
    szTitle  = "安装向导 - 程序组";
    szMsg    = "输入程序组的名称";
    nResult  = SdSelectFolder( szTitle,szMsg,szfolder );
```

```
    SHELL_OBJECT_FOLDER = szfolder;
    if (nResult = BACK) goto Dlg_ObjDialogs;

Dlg_SdStartCopy:
    szTitle = "安装向导 - 复制文件";
    szMsg   = "准备开始复制文件";
    listStartCopy = ListCreate( STRINGLIST );
    //The following is an example of how to add a string(szName) to a list(listStartCopy).
    //eg. ListAddString(listStartCopy,szName,AFTER);
    nResult = SdStartCopy( szTitle,szMsg,listStartCopy );
    ListDestroy(listStartCopy);
    if (nResult = BACK) goto Dlg_SdSelectFolder;

    //setup default status
    SetStatusWindow(0,"正在复制文件……");
    Enable(STATUSEX);
    StatusUpdate(ON,100);

CreateShortCut:          //创建快捷方式,设置 TargetVariable 的值(AutoCAD 安装路径)
    ComponentSetTarget( MKDIA,"<TargetVariable>",szAcadPath );
    CreateShellObjects( "" );

    return 0;
end;

////////////////////////////////////////////////////////////
//
//  FUNCTION:   OnMoving
//
//  EVENT:      Moving event is sent when file transfer is started as a result of
//                ComponentTransferData call,before any file transfer operations
//                are performed.
//
////////////////////////////////////////////////////////////
function OnMoving( )
    string szAppPath;
begin
    //Set LOGO Compliance Application Path
    //TO DO : if your application .exe is in a subfolder of TARGETDIR then add subfolder
    szAppPath = TARGETDIR;
    RegDBSetItem(REGDB_APPPATH,szAppPath);
    RegDBSetItem(REGDB_APPPATH_DEFAULT,szAppPath ^ @PRODUCT_KEY);
end;

//--- include script file section ---
```

```
//////////////////////////////////////////////////////////
//
//  FUNCTION:   IsAcadInstalled 判断 AutoCAD 2005 是否安装
//
//
//////////////////////////////////////////////////////////
function IsAcadInstalled(szAcadPath)
    BOOL bExist;        //键值是否存在
    STRING szKey1,szKey2,szPathName,szPathValue;
    NUMBER nRootKey,nvType,nvSize;
begin
    nRootKey = HKEY_LOCAL_MACHINE;
    if (RegDBSetDefaultRoot(nRootKey) < 0) then
        //退出该函数
    endif;

    szKey1 = "SOFTWARE\\Autodesk\\AutoCAD\\R16.1\\ACAD-301:804";  //中文版
    szKey2 = "SOFTWARE\\Autodesk\\AutoCAD\\R16.1\\ACAD-301:409";  //英文版
    bExist = FALSE;
    szPathName = "AcadLocation";      //保存 AutoCAD 安装路径的键的名称
    nvType = REGDB_STRING;            //字符串类型
    szAcadPath = "";                  //保存 AutoCAD 安装路径的全局变量

    if (RegDBKeyExist(szKey1) < 0 && RegDBKeyExist(szKey2) < 0) then
        //未安装 AutoCAD 2005
        return bExist;
    else
        //已安装
        if (RegDBKeyExist(szKey1) < 0) then      //未安装中文版
            if (RegDBGetKeyValueEx(szKey2,szPathName,nvType,szPathValue,
                                   nvSize) < 0) then
                MessageBox("获得 AutoCAD 2005 安装路径失败!",SEVERE);
                return bExist;
            else
                szAcadPath = szPathValue;
                bExist = TRUE;
                //获得英文版配置文件在注册表中存放的路径
                szAcadProfilePath = szKey2 ^ "\\Profiles";

                return bExist;
            endif;
        else     //安装中文版
            if (RegDBGetKeyValueEx(szKey1,szPathName,nvType,szPathValue,
                                   nvSize) < 0) then
                MessageBox("获得 AutoCAD 2005 安装路径失败!",SEVERE);
```

```
                return bExist;
            else
                szAcadPath = szPathValue;
                //获得中文版配置文件在注册表中存放的路径
                szAcadProfilePath = szKey1 ^ "\\Profiles";
                bExist = TRUE;
                return bExist;
            endif;
        endif;
    endif;
end;

////////////////////////////////////////////////////////////
//
//  FUNCTION:   OnEnd
//
//  EVENT:      End event is the last event. It is not sent if the installation
//               has been aborted. In this case Abort event is sent
//
////////////////////////////////////////////////////////////
function OnEnd( )

begin
    //安装结束时向 AutoCAD 添加配置文件
    AddAcadProfile(szAcadProfilePath,TARGETDIR);
end;

////////////////////////////////////////////////////////////
//
//  FUNCTION:   AddAcadProfile 向 AutoCAD 添加配置文件
//
//
////////////////////////////////////////////////////////////
function AddAcadProfile ( szProfilesKey,szSupportPath )
    STRING szSubKey,szPrimaryKey;
    STRING szAcad,szMenuFile,szTemplatePath;
    STRING szDefault;      //默认的配置名称
    NUMBER nvSize,nRet,nvType;
    STRING szKey,szARXName;
begin
    RegDBSetDefaultRoot ( HKEY_CURRENT_USER );

    //首先获得默认的配置名称 szDefault
    nvType = REGDB_STRING;      //键值的类型
    nvSize = MAX_KEY_LEN;       //键名的最大长度
    szPrimaryKey = szAcadProfilePath;
    nRet = RegDBGetKeyValueEx (szPrimaryKey,"",nvType,szDefault,nvSize);
```

```
    //获得原有的 ACAD 配置中的支持文件路径
    nvSize = MAX_KEY_LEN;
    szPrimaryKey = szProfilesKey^"\\"^szDefault^"\\General";
    nRet = RegDBGetKeyValueEx (szPrimaryKey,"ACAD",nvType,szAcad,nvSize);
    if(StrLength(szAcad) = 0) then
        szAcad = szAcadPath + "\\support;" + szAcadPath + "\\fonts";
    endif;

    //自定义应用程序的支持文件路径和模板文件路径
    szSupportPath = TARGETDIR + "\\VBA Macros";
    szAcad = szSupportPath + ";" + szAcad;
    szTemplatePath = TARGETDIR + "\\Templates";
    szMenuFile = TARGETDIR + "\\mjtd";

    //创建自定义应用程序的配置
    szSubKey = szProfilesKey + "\\mjtd";
    szPrimaryKey = szSubKey^"\\General";
    //创建配置文件的主键 mjtd
    nRet = RegDBCreateKeyEx(szPrimaryKey,"" );
    //创建其中的子键 ACAD
    nRet = RegDBSetKeyValueEx(szPrimaryKey,"ACAD",REGDB_STRING,szAcad,-1);
    //指定模板文件的路径
    nRet = RegDBSetKeyValueEx(szPrimaryKey,"TemplatePath",REGDB_STRING,szTemplatePath,-1);
    szPrimaryKey = szSubKey ^ "\\General Configuration";
    //创建其中的子键 General Configuration
    nRet = RegDBCreateKeyEx(szPrimaryKey,"" );
    //指定菜单文件的位置
    nRet = RegDBSetKeyValueEx(szPrimaryKey,"MenuFile",REGDB_STRING,szMenuFile,-1);
end;

////////////////////////////////////////////////////////////
//
//  FUNCTION: DeleteAcadProfile  卸载程序时删除 mjtd 配置
//
////////////////////////////////////////////////////////////
function DeleteAcadProfile( )
    STRING szSubKey,szMjtdProfilePath;
    NUMBER nRet;
begin
    nRet = RegDBSetDefaultRoot ( HKEY_CURRENT_USER );

    //删除 mjtd 配置对应的键
    szSubKey = szAcadProfilePath + "\\mjtd";
    nRet = RegDBDeleteKey(szSubKey);

    //将当前配置的键值清空
```

```
    nRet = RegDBSetKeyValueEx(szAcadProfilePath,"",REGDB_STRING,"",-1);
end;
#include "ComponentEvents.rul"
```

(30) InstallShield 代码窗口上方有一个用于处理事件的工具栏，如图 15.61 所示。从左侧的列表中选择 After Move Data 选项，从右侧的列表中选择 End 选项，系统会添加处理 End 事件的函数 OnEnd，在其中添加一句代码“AddAcadProfile(szAcadProfilePath, TARGETDIR);”，这在上面的代码中已经手工添加。



图 15.61 处理事件的工具栏

(31) 从左侧的列表中选择 Main App 选项，从右侧的列表中选择 UnInstalled 选项，系统会添加一个文件 ComponentEvents.rul，打开该文件，修改其中的代码：

```
export prototype MainApp_UnInstalled( );
function MainApp_UnInstalled( )
begin
    DeleteAcadProfile( );
end;
```

(32) 关闭安装包中的所有文件，在 InstallShield 中选择 Build|Build (default)Media 菜单项，或者直接按下 F7 快捷键，对脚本文件进行编译，完成编译之后，系统会在操作系统所在的磁盘分区的“\My Installations\AutoCAD VBA 高级开发实例教程\Media\Default\Disk Images\Disk1”文件夹中输出安装包。至此为止，安装包的制作已经完成。

15.7.4 实例效果

(1) 双击输出的安装包中的 Setup.exe 文件，完成初始化工作之后，系统会弹出如图 15.62 所示的对话框，准备开始安装应用程序，单击“下一步”按钮。

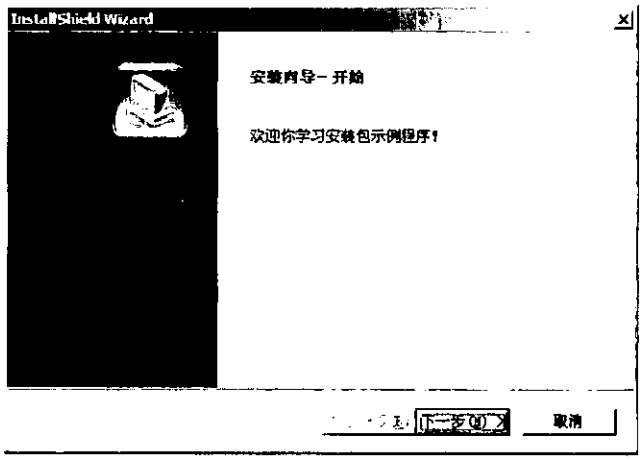


图 15.62 准备开始安装应用程序

(2) 系统弹出如图 15.63 所示的对话框，询问用户是否同意许可协议，单击“是”按钮才能继续安装。

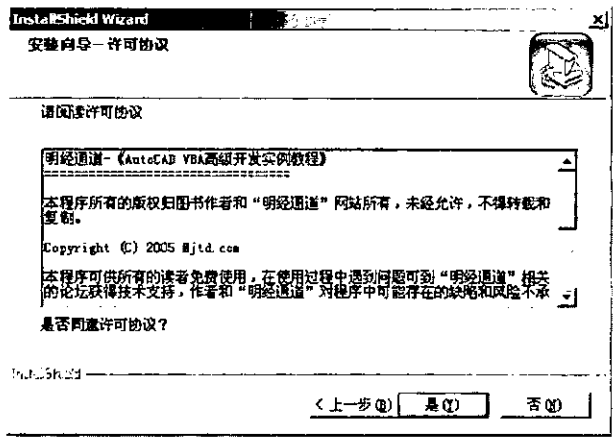


图 15.63　阅读许可协议

(3) 系统弹出如图 15.64 所示的对话框，输入正确的序列号才能够继续安装，这里在程序中限制了序列号必须为 8808-123456，其他的序列号均无法继续安装。输入正确的序列号，单击“下一步”按钮。

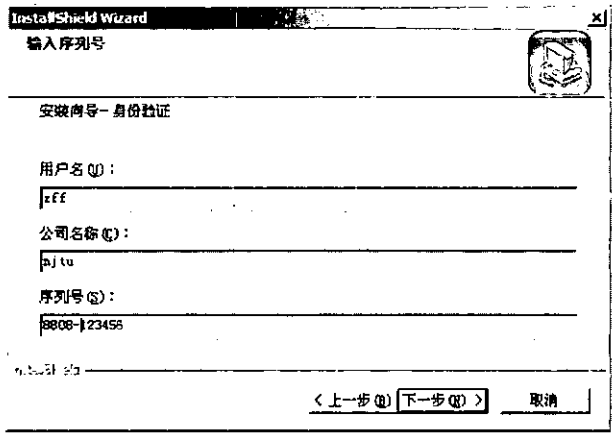


图 15.64　输入序列号

(4) 系统弹出如图 15.65 所示的对话框，单击“浏览”按钮，选择合适的安装位置，单击“下一步”按钮继续安装。

(5) 系统弹出如图 15.66 所示的对话框，提示用户输入程序组的名称，单击“下一步”按钮。

(6) 系统弹出如图 15.67 所示的对话框，准备开始复制文件，单击“下一步”按钮继续安装。

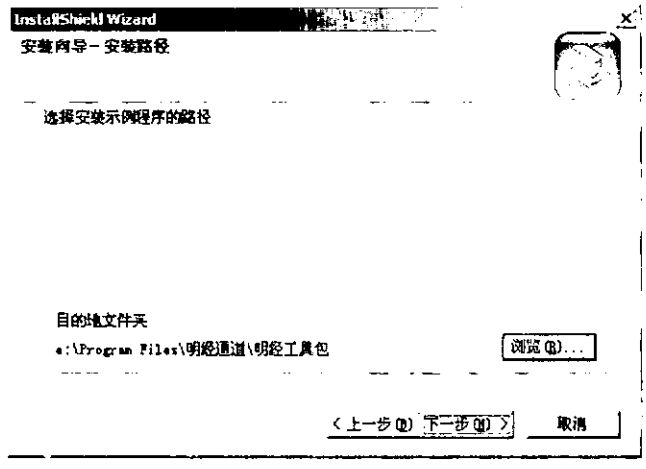


图 15.65 选择目标文件夹

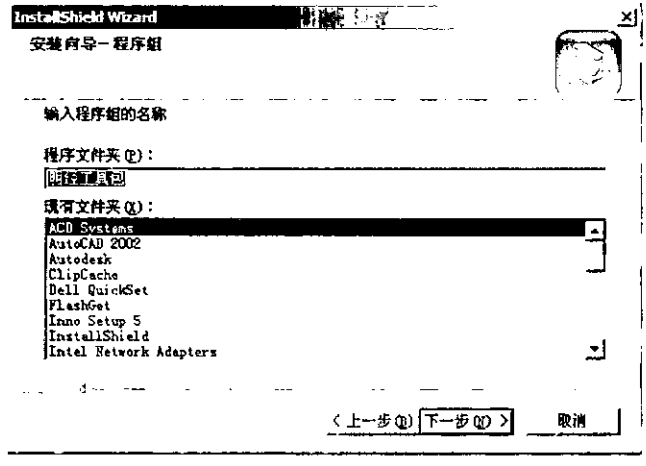


图 15.66 输出程序组的名称

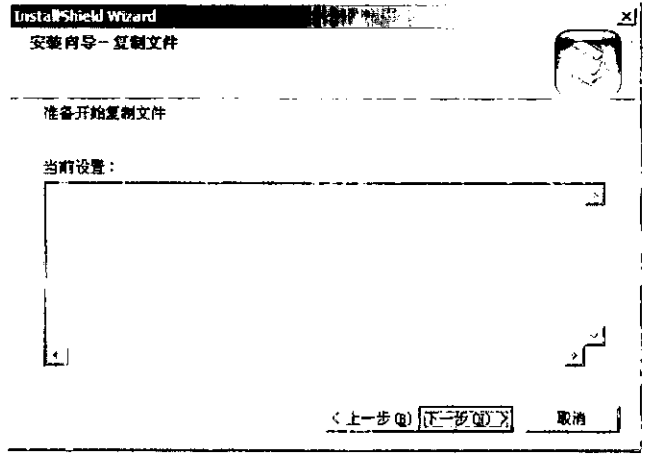


图 15.67 准备开始复制文件

(7) 经过短暂的文件复制过程，系统弹出如图 15.68 所示的对话框，提示用户安装已经完成，单击“完成”按钮，完成程序的安装。

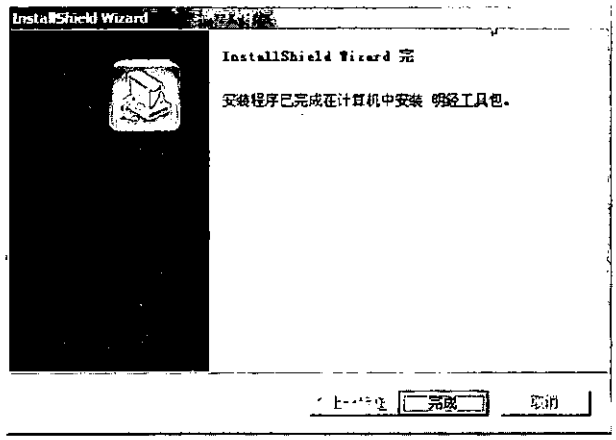


图 15.68　完成程序的安装

(8) 完成安装后，在桌面上得到一个快捷方式图标，如图 15.69 所示。在该快捷方式图标上右击，从弹出的快捷菜单中选择“属性”菜单项，系统会弹出如图 15.70 所示的对话框，在其中可以看到“目标”文本框中添加了/p 的启动选项，“起始位置”被设置成应用程序安装的位置。



图 15.69　得到的桌面快捷方式

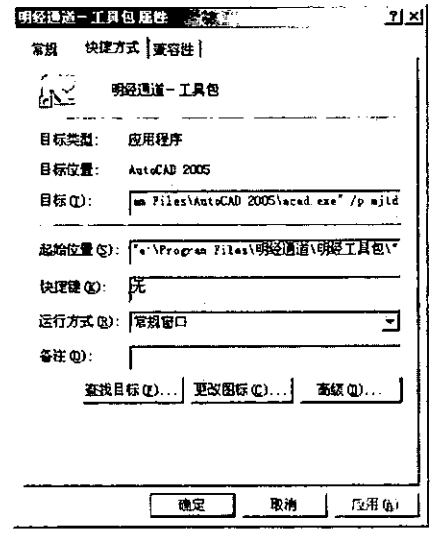


图 15.70　桌面快捷方式的属性

(9) 此外，在开始菜单中同样得到了一个名为“明经通道－工具包”的快捷方式菜单。此时，使用应用程序的图标可以进入应用程序，并且加载指定的菜单，而使用 AutoCAD 的快捷方式则会直接进入 AutoCAD，两者不会互相影响。使用“明经通道－工具包”快捷方式进入应用程序，在 AutoCAD 中增加了一个“明经通道”菜单栏，如图 15.71 所示。

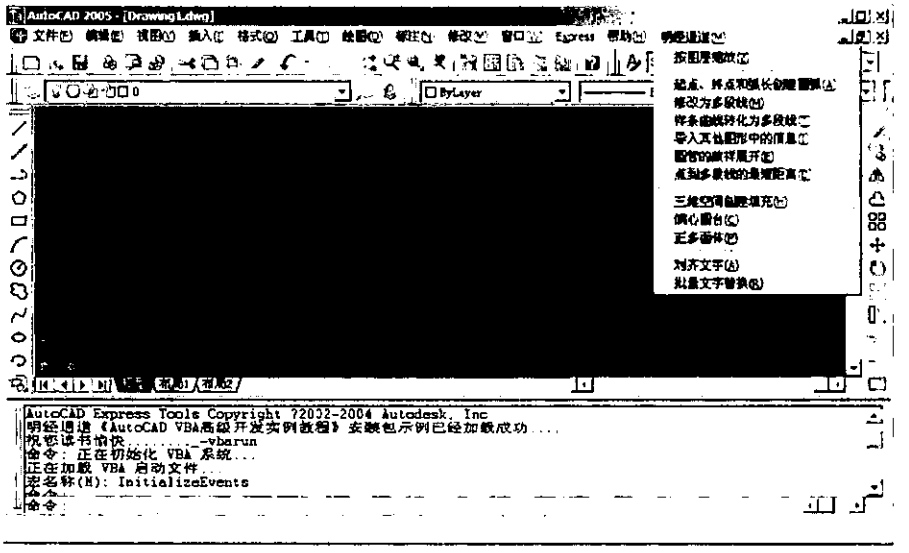


图 15.71 进入应用程序的界面

(10) 要卸载应用程序,可在控制面板中双击“添加或删除程序”图标,打开“添加或删除程序”对话框,在列表中选择“明经工具包”选项,单击“更改/删除”按钮,系统会弹出如图 15.72 所示的对话框,选中“删除”单选按钮,单击“下一步”按钮。

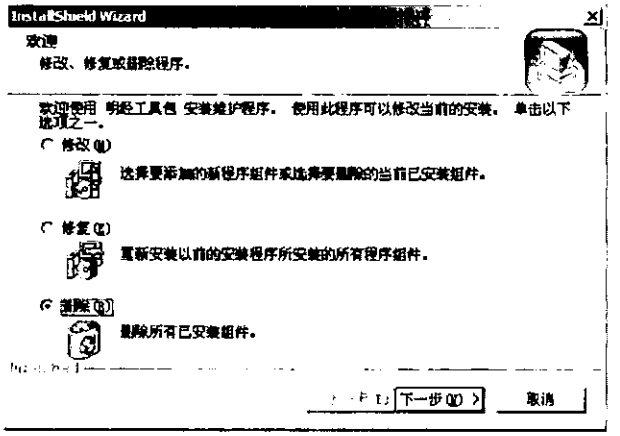


图 15.72 卸载应用程序的界面

(11) 系统弹出一个对话框,需要用户确认卸载操作,单击“确定”按钮,自动进行一些处理之后,系统弹出如图 15.73 所示的对话框,单击“完成”按钮,完成卸载的操作。

(12) 卸载应用程序之后,两个快捷方式被删除,复制到客户计算机中的所有文档也会被删除,此外,客户计算机的 AutoCAD 中的 mjtd 配置被删除,原来 AutoCAD 的功能不会受到丝毫的影响。

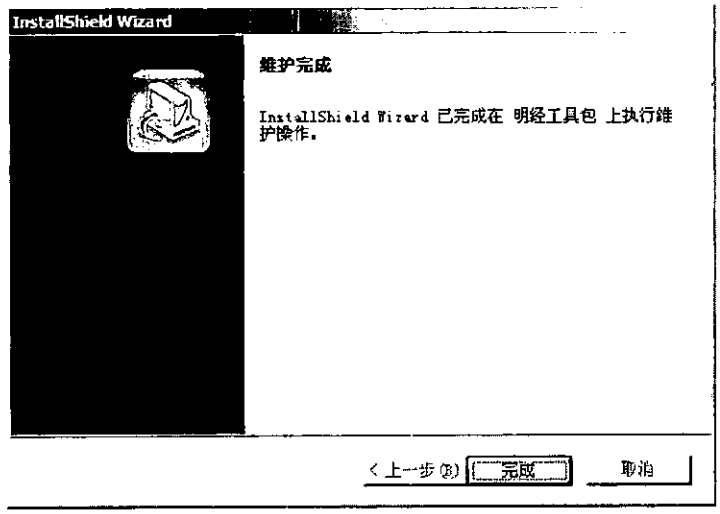


图 15.73　卸载程序完成

15.7.5　小结

本节涉及到非常多的内容，如果读者需要在自己的程序中使用图像控件菜单，就可以参考“基本图库的实现”一节；如果要实现先选择后操作的功能，可以参考“实现 VBA 程序的先选择后操作”一节；如果要在自己的应用程序中更改 AutoCAD 的图标和标题栏，可以参考“更改 AutoCAD 应用程序的图标和标题栏”一节。

另外，本节的安装程序仅能适用于 AutoCAD 2005 中英文版本，将其修改成适用于其他 AutoCAD 版本的安装包非常容易。Visal LISP 和 ObjectARX 应用程序的安装打包同样可以使用相同的方法，惟一的不同在于程序文件自动加载的方式：LISP 文件可以在 acaddoc.lsp 中加载，ObjectARX 应用程序可以用 acad.rx 文件实现自动加载。

第 16 章　VBA 开发相关技术

由于 VBA 和 VB 有一些小的差别，例如 VB 中所能使用的控件多于 VBA，因此某些情况下必须在 VB 中创建应用程序，本章介绍了将 VBA 程序转换到 VB 中的方法，并且通过将带有窗体的 VB 程序转化成 DLL 工程在 VBA 中调用，加快程序执行的速度。

搜索文件在 VBA 程序中可能会用到，本章提供了搜索某个文件夹中所有的文件的方法，并且配合 VBA IDE 的编程，可以查找包含某个函数应用的所有的 DVB 文件，这对学习 VBA 开发的读者来说是个有用的工具。

通过对 VBA IDE 进行编程，能够获得当前 VBA 工程的路径信息、代码中的过程、模块和其他的信息。

16.1　在 VB 中创建应用程序

16.1.1　实例说明

本节将把《AutoCAD VBA 开发精彩实例教程》中“批量替换文字”的程序转换成 VB 中的程序，两个程序所实现的功能完全相同，只是该程序在 VB 中通过 OLE 技术访问 AutoCAD，并且在对话框中实现了一些 VBA 中无法实现的功能。

16.1.2　编程思路

要将 VBA 中的代码转换到 VB 中，必须先在 VB 中引用 AutoCAD 的类型库，然后使用 CreateObject 和 GetObject 函数与 AutoCAD 建立连接，并通过应用程序对象来访问需要的功能，下面的代码能够在 VB 中与 AutoCAD 建立连接：

```
Public Sub LinkAutoCAD( )
    Dim acadApp As AcadApplication

    On Error Resume Next
    '获得正在运行的 AutoCAD 应用程序对象
    Set acadApp = GetObject(,"AutoCAD.Application.16")

    If Err Then
        Err.Clear
        '创建一个新的 AutoCAD 应用程序对象
        Set acadApp = CreateObject("AutoCAD.Application.16")

        If Err Then
            MsgBox Err.Description
```

```
            Exit Sub
        End If
    End If
End Sub
```

连接 AutoCAD 之后，就可以使用 acadApp 来访问 AutoCAD 中的 Application 对象。使用下面的代码可以获得当前活动的文档，acadDoc 对象就可以用于代替 VBA 中的 ThisDrawing 对象：

```
Dim acadDoc As AcadDocument
Set acadDoc = acadApp.ActiveDocument
```

如果要显示 AutoCAD 应用程序，可以将 AutoCAD 应用程序对象的 Visible 属性设置为 True。

16.1.3　步骤

(1) 运行 VB 6.0，创建一个标准应用程序，在窗体中放置如图 16.1 所示的控件。

其中，窗体的名称修改为 frmMain，文本框的名称修改为 txtFind 和 txtReplace，设置文档框的 Text 属性为空，列表框的名称修改为 lstFile，4 个命令按钮的名称修改为 cmdOpen、cmdDelete、cmdCancel 和 cmdOk，标准对话框控件的名称修改为 comDlg。

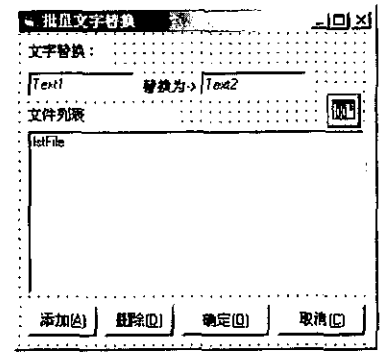


图 16.1　设计窗体

在 VB 中设置按钮的关键字与 VBA 中有所不同，VBA 中设置按钮的关键字，一方面要设置 Caption 属性为“确定(O)”，然后设置 Accelerator 属性为 O；在 VB 中则将按钮的 Caption 属性设置为“确定(&O)”即可。

(2) 在属性窗口中将工程的名称设置为 TextReplace，保存工程在合适的位置。选择 Project|References 菜单项，系统会弹出如图 16.2 所示的对话框，选中 AutoCAD 2005 Type Library 选项，单击 OK 按钮完成对象类型库的引用。

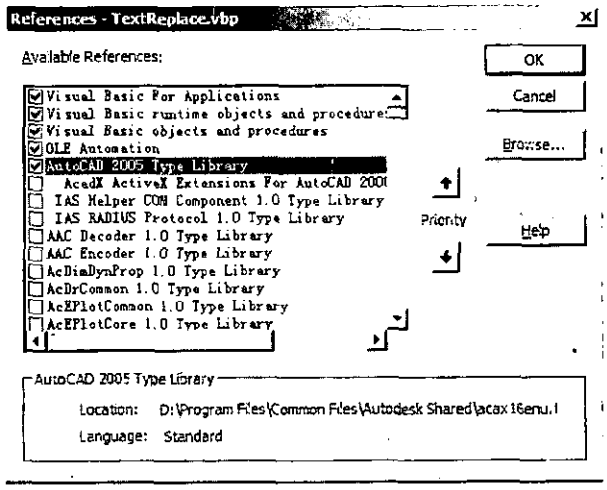


图 16.2　引用 AutoCAD 2005 对象类型库

(3) 打开对话框的代码窗口,在其中定义两个模块级变量:

```
Dim acadApp As AcadApplication        'AutoCAD 应用程序对象
Dim acadDoc As AcadDocument           '当前活动文档对象
```

(4) 在窗体的加载过程中,完成 VB 和 AutoCAD 的连接,并且显示 AutoCAD 应用程序窗口,其相关代码为:

```
Private Sub Form_Load( )
    On Error Resume Next
    '获得正在运行的 AutoCAD 应用程序对象
    Set acadApp = GetObject( , "AutoCAD.Application.16")

    If Err Then
        Err.Clear
        '创建一个新的 AutoCAD 应用程序对象
        Set acadApp = CreateObject("AutoCAD.Application.16")

        If Err Then
            MsgBox Err.Description
            Exit Sub
        End If
    End If

    '显示 AutoCAD 应用程序
    acadApp.Visible = True

    lstFile.Clear
End Sub
```

(5)在"添加"按钮的单击事件中,显示标准对话框,获取用户的选择,并将选择的结果添加到列表框中,其实现代码为:

```
Private Sub cmdOpen_Click( )
    On Error GoTo errHandle

    Dim i As Integer
    Dim Y As Integer
    Dim Z As Integer
    Dim fileNames( ) As String

    With comDlg
        .CancelError = True
        .MaxFileSize = 32767
        .Flags = cdlOFNHideReadOnly Or cdlOFNAllowMultiselect Or cdlOFNExplorer
            Or cdlOFNNoDereferenceLinks
        .DialogTitle = "选择图形文件"
        .Filter = "图形文件(*.dwg)|*.dwg|所有文件(*.*)|*.*"
```

```
        .FileName = ""
        .ShowOpen
    End With

    comDlg.FileName = comDlg.FileName & Chr(0)      '这些文件名是用空字符 Chr(0)分隔符,而不
                                                    '是空格分隔符隔开

    Z = 1
    For i = 1 To Len(comDlg.FileName)
        'InStr 函数,返回 Variant (Long),指定一字符串在另一字符串中最先出现的位置
        '语法 InStr(起点位置, string1, string2)
        i = InStr(Z, comDlg.FileName, Chr(0))
        If i = 0 Then Exit For
        ReDim Preserve fileNames(Y)
        'Mid 函数,返回 Variant (String),其中包含字符串中指定数量的字符
        '语法 Mid(string, start[, length])
        fileNames(Y) = Mid(comDlg.FileName, Z, i - Z)
        Z = i + 1
        Y = Y + 1
    Next i

    '向列表框中添加对象
    Dim count As Integer
    count = lstFile.ListCount
    If Y = 1 Then
        If Not HasItem(fileNames(Y - 1)) Then
            lstFile.AddItem fileNames(Y - 1), count
        End If
    Else
        For i = 1 To Y - 1
            If StrComp(Right$(fileNames(0), 1), "\") = 0 Then
                fileNames(i) = fileNames(0) & fileNames(i)
            Else
                fileNames(i) = fileNames(0) & "\" & fileNames(i)
            End If

            If Not HasItem(fileNames(i)) Then
                lstFile.AddItem fileNames(i), i - 1 + count
            End If
        Next i
    End If

errHandle:

End Sub
```

其中，HasItem是自定义函数，用于检查列表框中是否存在指定名称的项目，其实现代码为：

```
Private Function HasItem(ByVal strDwgName As String) As Boolean
    HasItem = False

    Dim i As Integer
    For i = 0 To lstFile.ListCount - 1
        If StrComp(lstFile.List(i), strDwgName, vbTextCompare) = 0 Then
            HasItem = True
            Exit Function
        End If
    Next i
End Function
```

(6) 窗体中“删除”按钮的单击事件，用于从列表框中删除指定的元素，其实现代码为：

```
Private Sub cmdDelete_Click( )
    '确认列表框包含列表项
    If lstFile.ListCount > = 1 Then
        '如果没有选中的内容，用上一次的列表项
        If lstFile.ListIndex = - 1 Then
            MsgBox "请选择列表中的图形名称!"
            Exit Sub
        End If
        lstFile.RemoveItem (lstFile.ListIndex)
    End If

End Sub
```

(7) 窗体中“确定”按钮的单击事件，用于打开每一个列表框中显示的图形文件，并且对文件进行文字替换的操作，其实现代码为：

```
Private Sub cmdOk_Click( )
    Dim adText As AcadText
    Dim adMText As AcadMText
    Dim adSS As AcadSelectionSet
    Dim fType(0 To 1) As Integer, fData(0 To 1)
    Dim i As Integer

    If txtFind.Text = "" Or txtReplace.Text = "" Then
        MsgBox "输入所要替换的字符串内容!"
        Exit Sub
    End If
    If lstFile.ListCount = 0 Then
        MsgBox "请添加所要操作的图形!"
        Exit Sub
    End If
```

```
    '获得替换数据
    Dim strFind As String
    Dim strReplace As String
    strFind = txtFind.Text
    strReplace = txtReplace.Text

    '打开图形进行操作
    For i = 0 To lstFile.ListCount - 1
        Call ReplaceTextInDwg(lstFile.List(i), strFind, strReplace)
    Next i

    '在退出应用程序之前关闭 AutoCAD
    acadApp.Quit
    End
End Sub
```

其中，ReplaceTextInDwg 是一个自定义函数，用于打开指定的图形，并且使用 ObjectDBX 技术完成文字的查找和替换，其实现代码为：

```
Private Sub ReplaceTextInDwg(ByVal strDwgName As String, ByVal strFind As String, ByVal
    strReplace As String)
    '打开指定的图形
    acadApp.Documents.Open strDwgName
    Set acadDoc = acadApp.ActiveDocument

    Dim ent As AcadEntity
    For Each ent In acadDoc.ModelSpace
        If TypeOf ent Is AcadText Or TypeOf ent Is AcadMText Then
            With ent
                If InStr(.TextString, strFind) Then .TextString = ReplaceStr(.TextString,
                    strFind, strReplace, False)
            End With
        End If
    Next ent

    '保存并关闭图形
    acadDoc.Close True
End Sub
```

上面代码中使用了自定义函数 ReplaceStr，用来完成对图形中指定字符串的替换，其实现代码为：

```
Public Function ReplaceStr(ByVal searchStr As String, ByVal oldStr As String, _
        ByVal newStr As String, ByVal firstOnly As Boolean) As String
    '对错误操作的处理
    If searchStr = "" Then Exit Function
    If oldStr = "" Then Exit Function
```

```
    ReplaceStr = ""
    Dim i As Integer, oldStrLen As Integer, holdStr As String, StrLoc As Integer

    '计算原来字符串的长度
    oldStrLen = Len(oldStr)
    StrLoc = InStr(searchStr, oldStr)

    While StrLoc > 0
        '获得图形中文字对象位于查找字符串之前的字符串
        holdStr = holdStr & Left(searchStr, StrLoc - 1) & newStr
        '获得文字对象位于查找字符串之后的字符串
        searchStr = Mid(searchStr, StrLoc + oldStrLen)
        StrLoc = InStr(searchStr, oldStr)
        If firstOnly Then ReplaceStr = holdStr & searchStr: Exit Function
    Wend

    ReplaceStr = holdStr & searchStr
End Function
```

至此为止，对应的 VBA 实例的所有代码已经转换成 VB 代码，并且已经可以完成与 VBA 实例相同的功能。下面的部分，将在列表框中元素不能完全显示的情况下弹出一个提示框，可以供用户查看完整的图形文件名称。

(8) 在对话框“取消”按钮的单击事件中，添加退出应用程序和关闭 AutoCAD 的代码：

```
Private Sub cmdCancel_Click( )
    acadApp.Quit
    End
End Sub
```

(9) 在对话框的通用代码区，添加 API 函数和常量的声明：

```
Const LB_ITEMFROMPOINT = &H1A9
Private Declare Function SendMessage Lib "user32" Alias "SendMessageA" _
        (ByVal hwnd As Long, ByVal wMsg As Long, _
        ByVal wParam As Long, lParam As Any) As Long
```

(10) 在列表框的 MouseMove 事件中，判断当前光标所在的元素行，并将列表框的 Tooltip 属性设置为该元素行的内容，其实现代码为：

```
Private Sub lstFile_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
    Dim lXPoint As Long
    Dim lYPoint As Long
    Dim lIndex As Long

    '确定在移动光标的同时没有按下功能键或者鼠标键
    If Button = 0 Then
        '获得光标的位置，以像素为单位
```

```
        lXPoint = CLng(X/Screen.TwipsPerPixelX)
        lYPoint = CLng(Y/Screen.TwipsPerPixelY)

        '显示列表框中元素的内容
        With lstFile
            '获得光标所在行的索引
            lIndex = SendMessage(.hwnd, LB_ITEMFROMPOINT, 0, _
                                 ByVal ((lYPoint * 65536) + lXPoint))

            '将 ListBox 的 Tooltip 属性设置为该行的文本
            If (lIndex >= 0) And (lIndex <= .ListCount) Then
                .ToolTipText = .List(lIndex)
            Else
                .ToolTipText = ""
            End If
        End With
    End If
End Sub
```

16.1.4　实例效果

(1) 在 VB 中，按下 F5 键执行应用程序，在显示 frmMain 窗体的同时加载 AutoCAD 2005，如图 16.3 所示。

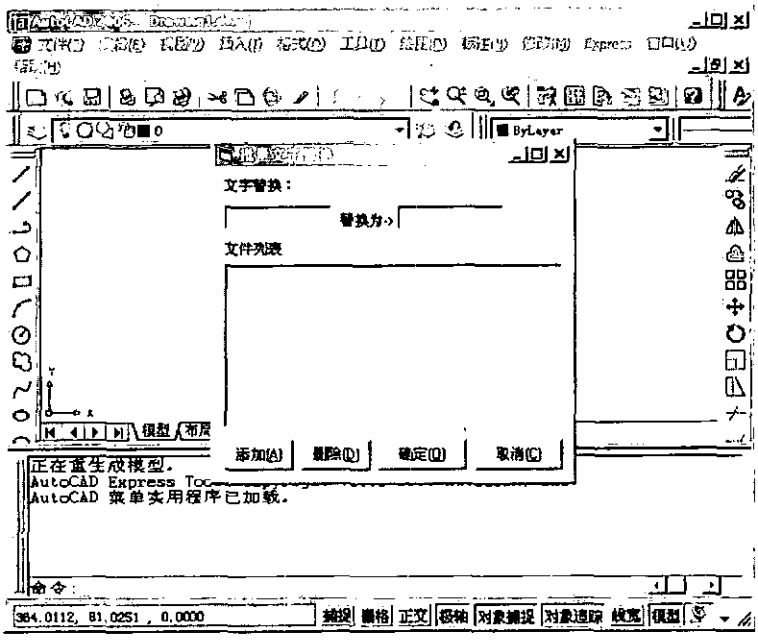


图 16.3　启动应用程序

(2) 单击“添加”按钮，在弹出的“选择图形文件”对话框中选择所要进行文字替换的图形，将其添加到列表框中。如果列表框中某个列表项的文字长度超出了列表框的宽度，当光

标移动到该列表项上方，系统会显示该列表项的完整名称，如图 16.4 所示。

(3)在文本框中输入所要查找和替换的文字，单击“确定”按钮，就能完成文字的查找和替换。

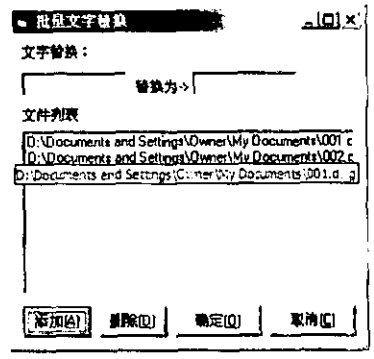


图 16.4 提示列表项的完整名称

16.1.5 小结

1. 在 AutoCAD 和 VB 窗口之间切换焦点

在 VB 开发 AutoCAD 应用程序时，还有一个问题需要考虑，就是从 VB 窗口中将焦点转移到 AutoCAD 中，有时候通过隐藏 VB 窗体的方法不能实现，所以必须找到一个更好的方法。

在 VB 工程中插入一个标准模块，其中添加下面的代码：

```
Private Declare Function GetWindowThreadProcessId Lib "user32" _
    (ByVal hWnd As Long, lpdwProcessId As Long) As Long
Private Declare Function AttachThreadInput Lib "user32" _
    (ByVal idAttach As Long, ByVal idAttachTo As Long, ByVal fAttach As Long) As Long
Private Declare Function GetForegroundWindow Lib "user32" ( ) As Long
Private Declare Function SetForegroundWindow Lib "user32" (ByVal hWnd As Long) As Long
Private Declare Function IsIconic Lib "user32" (ByVal hWnd As Long) As Long
Private Declare Function ShowWindow Lib "user32" _
    (ByVal hWnd As Long, ByVal nCmdShow As Long) As Long

Private Const SW_SHOW = 5
Private Const SW_RESTORE = 9

Public Function ForceForegroundWindow(ByVal hWnd As Long) As Boolean
  Dim ThreadID1 As Long     '线程 ID
  Dim ThreadID2 As Long     '线程 ID
  Dim nRet As Long

  '如果指定的窗体已经在前台，不做任何操作
  If hWnd = GetForegroundWindow( ) Then
    ForceForegroundWindow = True
  Else
    '首先获得指定窗体相关的线程和当前前台窗口所在的线程
    ThreadID1 = GetWindowThreadProcessId(GetForegroundWindow, ByVal 0&)
    ThreadID2 = GetWindowThreadProcessId(hWnd, ByVal 0&)

    '通过共享输入状态，两个线程分享当前窗口
    If ThreadID1 <> ThreadID2 Then
        Call AttachThreadInput(ThreadID1, ThreadID2, True)
        nRet = SetForegroundWindow(hWnd)
```

```
            Call AttachThreadInput(ThreadID1, ThreadID2, False)
        Else
            nRet = SetForegroundWindow(hWnd)
        End If

        '恢复和重画
        If IsIconic(hWnd) Then
            Call ShowWindow(hWnd, SW_RESTORE)
        Else
            Call ShowWindow(hWnd, SW_SHOW)
        End If

        '精确地返回函数执行结果
        ForceForegroundWindow = CBool(nRet)
    End If
End Function
```

在 frmMain 窗体中列表框的 DblClick 事件中，添加下面的代码：

```
Private Sub lstFile_DblClick( )
    Dim pt As Variant
    '将焦点切换到 AutoCAD
    ForceForegroundWindow acadApp.hWnd
    pt = acadApp.ActiveDocument.Utility.GetPoint(, "拾取一点：")

    '焦点切换回当前的窗体
    ForceForegroundWindow frmMain.hWnd

    '显示点的坐标
    MsgBox "拾取点的坐标为：(" & pt(0) & "," & pt(1) & "," & pt(2) & ")"
End Sub
```

执行改进后的程序，单击“添加”按钮，向列表框中添加一个文件，然后双击列表框中的列表项，AutoCAD 会获得焦点，并提示用户输入点，在图形窗口中拾取一个点，系统会弹出如图 16.5 所示的对话框。

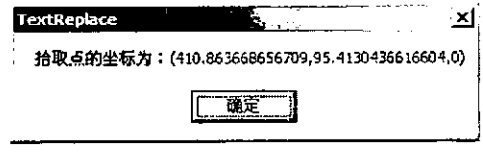


图 16.5　显示拾取点的坐标

单击“确定”按钮之后，“批量文字替换”对话框会获得焦点。

2. 将程序封装到 DLL 中

在 VB 中通过 OLE 技术控制 AutoCAD 完成的一些操作，由于需要通过 COM 在两个进程之间交换数据，速度会比 VBA 慢很多。如果是因为一些技术上的原因必须使用 VB 来

开发应用程序，为提高执行的速度，可以将 VB 的程序封装到一个 DLL 中，然后在 VBA 中调用 DLL，这样就避免了在两个进程中交换数据，速度能得到很大的提高。

下面用具体步骤来描述这个应用。

(1) 在 VB 中创建一个 ActiveX DLL 工程，在其中将类模块的名称修改为 CTest，在其中添加下面的代码：

```
Private m_oApp As Object

Public Property Set Application(ByVal vNewValue As Object)
    Set m_oApp = vNewValue
End Property

Public Sub ShowForm( )
    Set frmTest.Application = m_oApp
    frmTest.Show
End Sub
```

(2) 向 VB 工程中添加一个窗体，将其名称修改为 frmTest，该窗体中仅包含一个 Label 控件，如图 16.6 所示。

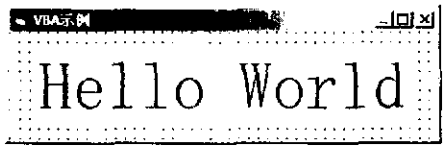


图 16.6 设计窗体

(3) 在 frmTest 窗体的代码窗口中，添加下面的代码：

```
Private Declare Function SetParent Lib "user32" (ByVal hWndChild As Long, ByVal hWndNewParent
    As Long) As Long
Private Declare Function GetParent Lib "user32" (ByVal hwnd As Long) As Long
Private m_oApp As Object

Public Property Set Application(ByVal vNewValue As Object)
    Set m_oApp = vNewValue
End Property

Private Sub Form_Load( )
    SetParent Me.hwnd, GetParent(GetParent(m_oApp.ActiveDocument.hwnd))
End Sub
```

(4)创建一个新的 VBA 工程，在其中引用 prjTest 组件(如果列表中不包含该组件，可以单击“浏览”按钮在计算机中定位该组件)，如图 16.7 所示。

(5) 在 VBA 工程中添加一个宏，用于显示 DLL 中的窗体，其相关代码：

```
Public Sub UseDll( )
    Dim objForm As New prjTest.CTest
```

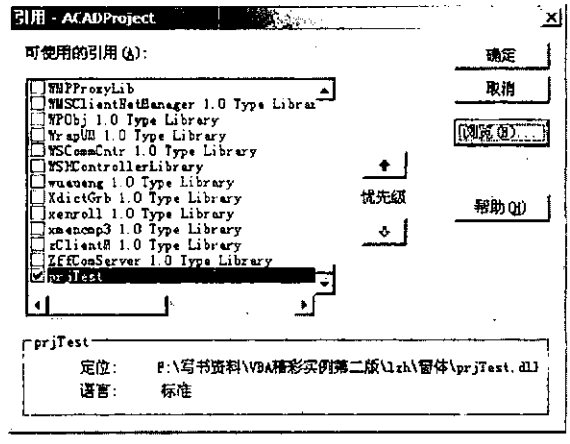


图 16.7　引用 prjTest 组件

```
    Set objForm.Application = Application
    objForm.ShowForm
End Sub
```

(6) 执行 UseDll 宏，能在 AutoCAD 中以非模态的形式显示 DLL 中的对话框，而且不会出现 VBA 中直接使用非模态对话框的焦点问题，如图 16.8 所示。

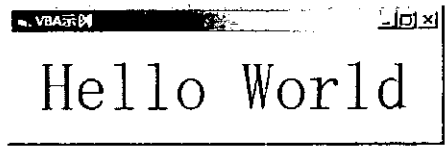


图 16.8　显示非模态窗体

(7) 如果要在 VBA 中以模态的形式显示 DLL 中的对话框，则需要将 DLL 工程类模块中显示对话框的函数修改成下面的形式：

```
Public Sub ShowForm( )
    Set frmTest.Application = m_oApp
    frmTest.Show vbModal
End Sub
```

16.2　搜索某个文件夹中所有的文件

16.2.1　实例说明

在 VBA 开发的过程中，不可避免地经常要和计算机中的文件打交道，实际上所有的桌面应用程序都会经常用到文件系统的知识。在 VBA 开发的过程中，使用频率比较高的一个操作，就是搜索某个文件夹(可能包含子文件夹)中所有的文件，本节的实例专门讨论这个问题。

本节创建了一个非常有用的函数 FindAllFiles，能够根据用户指定的条件，搜索文件夹中满足条件的所有文件。例如，可以搜索某个文件夹中（包含子文件夹）所有的 DVB 文件，这在下一节就会用到。

16.2.2 编程思路

1. 查找文件

VBA 中有一个内置函数 Dir 用于查找文件，执行该函数会返回一个字符串，用以表示一个文件名、目录名或文件夹名称，它必须与指定的模式或文件属性、磁盘卷标相匹配。利用这个函数，可以轻松编写一个判断文件是否存在的函数：

```
Public Function FileExists(ByVal strFile As String) As Boolean
    FileExists = (Len(Dir(strFile)) > 0)
End Function
```

Dir 函数使用起来非常方便，但是它对查找文件时文件属性的限定不够严密，例如要查找一个文件名包含 sys 的隐藏文件，Dir 函数会查找所有满足这两个条件（文件名包含 sys，是隐藏文件）的所有文件，甚至会把符合要求的文件夹查找出来。

但是如果不想让系统文件被检索出来，也就是搜索仅包含这两个条件的所有文件，此时 Dir 函数就无能为力了。为了解决问题，本节创建一个代替 Dir 的函数 dhDir，除了能够进行排他性（满足指定属性的同时，不满足其他的属性）查找之外，其他功能和 Dir 函数完全一致。

为了能够在查找文件的时候严格检查文件的属性，必须通过一些 Windows API 函数来达到目的。实现方法在操作步骤中给出了详细的说明。

2. 搜索指定文件夹中的所有文件

搜索文件夹的所有文件，必须通过一种叫做“递归”的算法，什么是递归？

举个例子，编写一个函数，计算一个整数的阶乘 n!。利用常规的方法解决这个问题比较困难，但是可以注意到，n 的阶乘实际上就是$(n-1)!$的 n 倍，$(n-1)$的阶乘就是$(n-2)!$的倍$(n-1)$，依此类推。按照这个原理就可以写出下面的函数来：

```
Public Function Factorial(ByVal n As Long) As Long
    If n = 0 Then           '退出条件
        Factorial = 1
    Else
        Factorial = n * Factorial(n - 1)
    End If
End Function
```

所谓递归函数，就是自己调用自己的函数。递归函数有两个特点：

- 在递归前后的函数值之间有固定的规律可循。
- 函数中有退出条件，否则就是死循环。

搜索文件夹中的所有文件，也必须要用到递归的算法，其算法原理为：

(1) 搜索指定文件夹中的所有文件，将其添加到集合中。

(2) 判断文件夹中是否包括子文件夹，如果不包含子文件夹，函数退出；如果包含子文件夹，对子文件夹执行步骤(1)～(2)的操作。

16.2.3 步骤

(1) 在 AutoCAD 2005 中，打开 VBA 管理器，创建一个新工程，保存在适当的位置。进入 VBA 集成开发环境，选择“插入”|“模块”菜单项，向工程中添加一个标准模块，将其重新命名为 mdlSearchFile。

(2) 在 mdlSearchFile 模块中，添加常量、类型和 API 函数的声明：

```
Private Const MAX_PATH = 260      'Windows 的最大路径长度为 260

'64 位的参数，以非常精确的单位来计算从 1601 - 01 - 01 开始所经历的时间
Private Type FILETIME
    dwLowDateTime As Long
    dwHighDateTime As Long
End Type

'用于 FindFirstFile 和 FindNextFile 函数中，来保存匹配文件的信息
Type WIN32_FIND_DATA
    lngFileAttributes As Long          '文件属性
    ftCreationTime As FILETIME         '创建时间
    ftLastAccessTime As FILETIME       '最后访问时间
    ftLastWriteTime As FILETIME        '最后修改时间
    lngFileSizeHigh As Long            '大小 (high word)
    lngFileSizeLow As Long             '大小 (low word)
    lngReserved0 As Long               '保留
    lngReserved1 As Long               '保留
    strFilename As String * MAX_PATH   '文件名称
    strAlternate As String * 14        '文件的可替代名称
End Type

'在给定的路径中搜索文件的 API 函数
Private Declare Function FindFirstFile Lib "kernel32" _
    Alias "FindFirstFileA" (ByVal lpFileName As String, _
    lpFindFileData As WIN32_FIND_DATA) As Long

Private Declare Function FindNextFile Lib "kernel32" _
    Alias "FindNextFileA" (ByVal hFindFile As Long, _
    lpFindFileData As WIN32_FIND_DATA) As Long

Private Declare Function FindClose Lib "kernel32" _
    (ByVal hFindFile As Long) As Long
```

(3) 添加一个函数 TrimNull，可对指定的字符串在第一个空字符处截尾，其实现代码：

```
'strValue: 输入可能包含一个空字符的字符串
```

```
Public Function TrimNull(ByVal strValue As String) As String
    Dim intPos As Integer

    '判断第一个空字符在字符串中出现的位置
    intPos = InStr(strValue, vbNullChar)
    Select Case intPos
        Case 0
            '如果未找到空字符,返回输入的字符串
            TrimNull = strValue
        Case 1
            '第一个字符是空字符,则返回空字符串
            TrimNull = ""
        Case Is > 1
            '返回第一个空字符之前的所有字符
            TrimNull = Left$(strValue, intPos - 1)
    End Select
End Function
```

(4) 添加一个函数 FindByAttr,用于获得下一个与指定的属性相匹配的文件,其实现代码:

```
'hFind: Windows API 查找句柄; fd: 指向 WIN32_FIND_DATA 结构的指针
'lngAttr: 所要测试的属性
'fExclusive: 如果选择 True, 只返回与指定的文件属性严格匹配的文件
Public Function FindByAttr(hFind As Long, _
    fd As WIN32_FIND_DATA, lngAttr As Long, _
    fExclusive As Boolean) As String

    Dim fOk As Boolean

    '查找文件,直到一个文件与给定的属性精确匹配(如果 fExclusive 设置为 True)
    '或者仅仅包含给定的属性(如果 fExclusive 设置为 False)
    Do
        If fExclusive Then
            '判断文件属性是否与给定的属性严格匹配
            If fd.lngFileAttributes = lngAttr Then
                fOk = True
            Else
                fOk = False
            End If
        Else
            '判断文件是否包含指定的属性
            If (fd.lngFileAttributes And lngAttr) = lngAttr Then
                fOk = True
            Else
                fOk = False
```

```
            End If
        End If

        '返回属性匹配的文件的名称
        If fOk Then
            dhFindByAttr = TrimNull(fd.strFilename)
            Exit Do
        End If
    Loop While FindNextFile(hFind, fd)
End Function
```

lngAttr 参数指定要匹配的文件属性，一般可取 vbNormal、vbReadOnly、vbHidden、VbSystem、vbVolume 和 vbDirectory 等常量，其具体含义可以参考 MSDN 中的说明。

fExclusive 参数是实现排他性查找的关键，如果其值为 True，用 fd. lngFileAttributes＝lngAttr 来保证文件的属性和指定的属性完全匹配；如果其值为 False，用(fd. lngFileAttributes And lngAttr)＝lngAttr 来检索文件是否包含指定的属性。

And 运算符对两个数值表达式中位置相同的位进行逐位比较，如果文件属性包含 lngAttr 所指定的所有属性，两者进行运算的结果正好是 lngAttr(关于位运算符的具体含义，可以参考 MSDN 中对 And 运算符的说明)，因此(fd. lngFileAttributes And lngAttr)＝lngAttr 为真；如果文件属性不完全包含 lngAttr 所指定的属性，两者使用 And 运算符进行位运算的结果不等于 lngAttr，则(fd. lngFileAttributes And lngAttr)＝lngAttr 为假。

(5) 添加一个函数 dhDir，用于实现与 Dir 函数类似的功能，其实现代码：

```
'strPath (可选，默认为空字符串)：指定所要搜索的文件或路径
'lngAttributes (可选，默认值为 vbNormal)：文件属性
'fExclusive (可选，默认为 True)：如果使用 True，仅返回严格匹配文件属性的文件
Public Function dhDir(Optional ByVal strPath As String = "", _
    Optional lngAttributes As Long = vbNormal, _
    Optional fExclusive As Boolean = True) As String

    Dim fd As WIN32_FIND_DATA '文件信息
    Static hFind As Long          '查找文件的句柄
    Static lngAttr As Long      '文件属性
    Static fEx As Boolean       '是否排除

    '如果没有传递路径，就查找下一个文件
    If strPath = "" Then
        If hFind > 0 Then
            If CBool(FindNextFile(hFind, fd)) Then
                dhDir = FindByAttr(hFind, fd, lngAttr, fEx)
            End If
        End If

    '否则，开始一次新的搜索
    Else
```

```
        '如果还有查找句柄存在,关闭它
        If hFind > 0 Then
            Call FindClose(hFind)
        End If

        '设置属性和排他性
        lngAttr = lngAttributes
        fEx = fExclusive

        '如果路径以"\"结束,确保搜索所有文件,因此加上"*.*"
        If Right(strPath, 1) = "\" Then
            strPath = strPath & "*.*"
        End If

        '查找第一个文件
        hFind = FindFirstFile(strPath, fd)
        If hFind > 0 Then
            dhDir = FindByAttr(hFind, fd, lngAttr, fEx)
        End If
    End If
End Function
```

如果在指定文件限制名称(可包含通配符)的情况下被调用,返回第一个匹配的文件名称;如果不带文件限制名称调用,则会返回下一个匹配的文件名称;如果没有找到匹配的文件,会返回一个空字符串。

(6) 添加一个函数 FindAllFiles,用于查找指定文件夹中所有的文件,其实现代码:

```
'strSpec: 所要搜索的文件限定符; strPath: 搜索路径
'colFound: 指向 VBA 集合对象的指针
'lngAttr: 所要搜索的文件具有的属性(-1 表示所有属性的文件)
'fRecursive: 是否递归调用(如果递归调用,会包含子文件夹)
Public Function FindAllFiles(strSpec As String, _
    ByVal strPath As String, colFound As Collection, _
    Optional lngAttr As Long = -1, _
    Optional fRecursive As Boolean = True) As Long

    Dim strFile As String              '文件名
    Dim colSubDir As New Collection    '子文件夹集合
    Dim varDir As Variant              '子文件夹的路径

    '确保字符串的结尾是"\"
    If Right(strPath, 1) <> "\" Then
        strPath = strPath & "\"
    End If

    '在路径中搜索所有文件,如果未指定属性,查找包含指定属性的文件
```

```
    If lngAttr = -1 Then
        strFile = dhDir(strPath & strSpec, , False)
    Else
        strFile = dhDir(strPath & strSpec, lngAttr)
    End If

    '搜索当前文件夹中所有符合要求的文件
    Do Until strFile = ""

        '将属性匹配的文件添加到集合中,注意排除特殊形式的路径(".."和"...")
        If ((GetAttr(strPath & strFile) And lngAttr) > 0) _
            And (strFile <> ".") And (strFile <> "..") Then
            colFound.Add strPath & strFile
        End If

        '查找下一个文件
        strFile = dhDir
    Loop

    '如果 fRecursive 为 True,进行递归调用
    If fRecursive Then
        '查找当前文件夹中所有的子文件夹
        strFile = dhDir(strPath, vbDirectory)
        Do Until strFile = ""
            '忽略"."和".."文件夹
            If strFile <> "." And strFile <> ".." Then
                '添加到子文件夹集合中
                colSubDir.Add strPath & strFile
            End If
            strFile = dhDir
        Loop

        '对每个子文件夹进行递归
        For Each varDir In colSubDir
            FindAllFiles strSpec, varDir, colFound, _
                         lngAttr, fRecursive
        Next
    End If

    '返回找到文件的数量
    FindAllFiles = colFound.count
End Function
```

16.2.4　实例效果

(1) 在 VBA 集成开发环境中,双击"工程资源管理器"窗口中的 ThisDrawing 图标,打开其代码窗口,创建下面的宏对 FindAllFiles 函数进行测试:

```
Public Sub FindFilesInFolder( )
    Dim fileCollection As New Collection
    Dim count As Long

    count = mdlSearchFile.FindAllFiles(" * .dvb", _
        "F: \VBA 开发精彩实例教程(第二版)\配套光盘\chap13 与其他应用程序和 Windows API 交互", _
        fileCollection)

    '显示所获得的文件名称
    Dim i As Integer
    For i = 1 To fileCollection.count
        Debug.Print fileCollection.Item(i)
    Next i
    Debug.Print "找到" & count & "个符合要求的文件。"
End Sub
```

(2) 将光标定位于 FindFilesInFolder 宏内部,按下 F5 键执行该宏。执行完毕,选择"视图"|"立即窗口"菜单项,能够在立即窗口中得到如图 16.9 所示的结果。

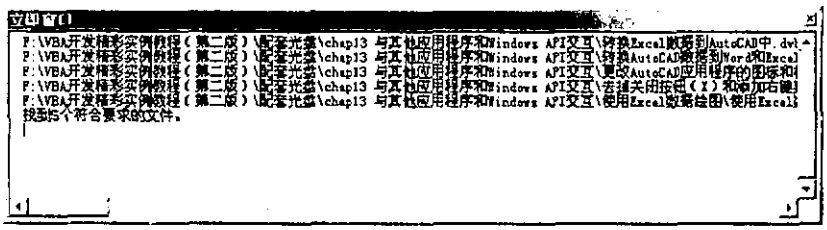


图 16.9 在立即窗口中显示查找结果

16.2.5 小结

学习本节内容之后,需要掌握下面的几个知识点:

- 递归算法的原理与应用。
- 在程序中判断一个文件是否存在。

16.3 VBA IDE 的编程

16.3.1 实例说明

VBA 的 IDE(集成开发环境)本身也输出了 ActiveX 模型,同样可以通过 VBA 操作其完成一些任务。VBA IDE 的模型并不算太复杂,但是开放的接口还是非常丰富的。本节不对 VBA IDE 的模型进行全面探讨,仅以与 AutoCAD VBA 编程比较密切的几个功能为例,说明 VBA IDE 的使用,同时提供了几个问题的解决方案:

- 在 VBA 中使用相对路径来访问资源、数据库和外部文件。
- 便于查找某个方法、属性的具体使用方法。

- 提取模块中的函数和过程，便于编写说明文档。

16.3.2　编程思路

本书前面的若干实例中，曾经使用下面的语句来获得当前工程的名称：

ThisDrawing. Application. VBE. ActiveVBProject. FileName

在本节中要获得当前加载的所有 DVB 文件的名称，可以访问 VBProjects 集合中的所有对象(即加载的各个工程)，获得其名称即可。

使用 VBComponents 集合的 CodeModule 属性可以访问模块的代码区，在 CodeModule 对象中提供了多种访问内部对象的方法和属性，例如，CountOfLines 属性返回模块中代码的行数，ProcOfLine 属性返回指定代码行所在过程的名称等。具体的对象属性和方法，可以在对象浏览器中查看，如图 16.10 所示。

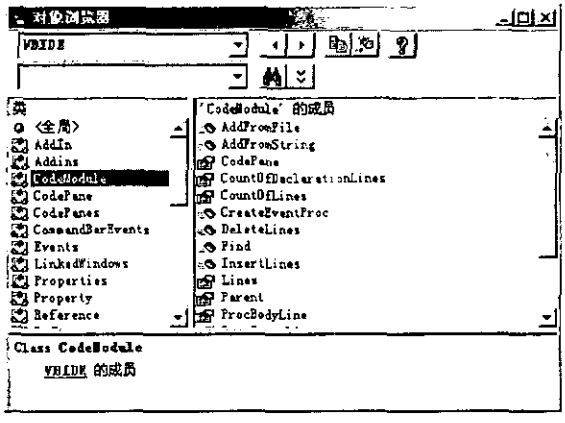


图 16.10　查看 IDE 对象模型

在指定的工程中搜索某个字符串，并且返回该字符串出现的位置(行号)和所在过程的名称，同样是利用 CodeModule 对象的属性和方法来实现。Find 方法用于按照一定的规则搜索指定的字符串，Lines 方法则返回指定行的代码内容。

16.3.3　步骤

(1) 在 AutoCAD 2005 中，打开 VBA 管理器，创建一个新工程，保存在适当的位置。在 VBA 管理器中，加载上一节创建的工程文件，进入 VBA 集成开发环境，在“工程资源管理器”窗口中右击 mdlSearchFile 模块，在弹出的快捷菜单中选择“导出文件”菜单项，将该模块保存在适当的位置。

(2) 切换到本节新建的工程文件中，选择“文件”|“导入文件”菜单项，在系统弹出的“导入文件”对话框中，选择步骤(1)导出的模块，将其添加到当前的工程中。选择“工具”|“引用”菜单项，在弹出的对话框中引用 Microsoft Visual Basic for Applications Extensibility 5.3 类型库，如图 16.11 所示。

(3) 在“工程资源管理器”窗口中双击 ThisDrawing 图标，打开其代码窗口，在其中添加一个宏 GetPathOfAllProjects，用于获得当前加载的所有工程名称，其实现代码：

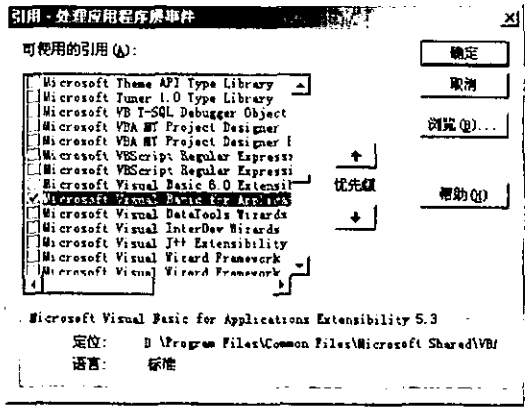


图 16.11 引用 VBA IDE 的类型库

```
Public Sub GetPathOfAllProjects( )
    On Error Resume Next
    Dim i As Integer
    Dim objProject As VBProject

    'VBA IDE 中的集合索引均是从 1 开始
    For i = 1 To ThisDrawing.Application.VBE.VBProjects.count
        Set objProject = ThisDrawing.Application.VBE.VBProjects(i)

        Debug.Print objProject.FileName

        If Err And Err.Number = 76 Then      '错误号 76 表示 FileName 的值无效
            Debug.Print "未保存的项目…"
        End If
    Next i
End Sub
```

如果有一些新建的工程还没来得及保存,那么 FileName 属性就会出错,错误处理的语句正是针对这一问题。

(4) 添加一个过程 ListProcess,用于将指定模块中所有的过程和函数打印在立即窗口中,其实现代码:

```
Public Sub ListProcess(module As CodeModule)
    Dim countLines As Long
    Dim lngType As Long
    Dim strProcess As String

    '打印标题栏
    Debug.Print "行号"; Tab(30); "过程名称"
    Debug.Print " ================================================== "

    With module
```

```
        '遍历所有的代码行,查找一个过程
        For countLines = 1 To .CountOfLines
            '获得当前行所在的过程名称
            strProcess = .ProcOfLine(countLines, lngType)

            '如果不是空行,就找到了一个过程
            If strProcess <> "" Then
                '打印行号、过程名称和类型
                Debug.Print "Line" & "(" & countLines & ")"; Tab(30); strProcess

                '跳过剩下的代码行
                countLines = countLines + .ProcCountLines(strProcess, lngType) - 1
            End If
        Next countLines
    End With

    Debug.Print " ================================================== "
End Sub
```

上面的代码使用了 CodeModule 对象的多个方法和属性,CountOfLines 属性返回模块中代码的行数,ProcOfLine 方法返回指定行号所在过程的名称,ProcCountLines 方法返回指定过程的代码行数。需要注意的是,VBA IDE 所有的集合对象索引都是从 1 开始,最大索引值是集合对象 count 属性的值,VB 中 Collection 对象也是如此。

为了获得在立即窗口中对齐各列文字的效果,这里在 Print 方法中使用了 Tab 函数。在使用的过程中费了好一番功夫才找到使用方法,因为 MSDN 以及 Visual Basic 参考中关于该函数用法的说明是有错误的,如图 16.12 所示。MSDN 中关于 Tab 函数的说明是在该函数前后加冒号(:),但是将示例的代码复制到 VB 的代码窗口中会显示语法不正确,正确的用法是在该函数前后加分号(;)。

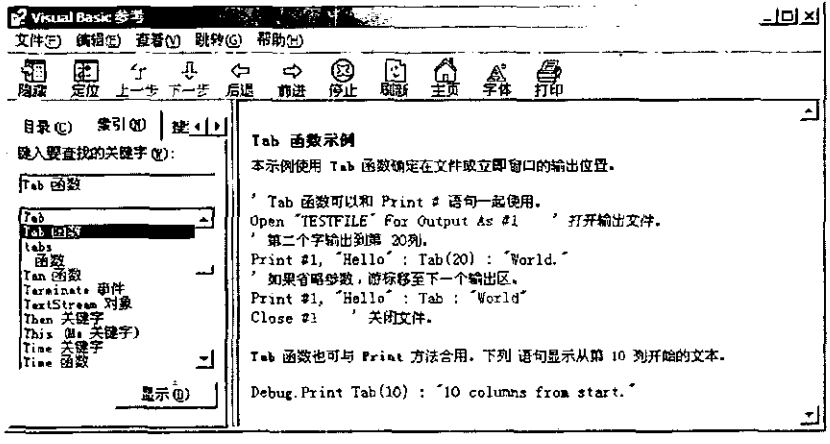


图 16.12　存在疏漏的 Tab 函数示例

(5) 添加一个过程 SearchString,用于在指定的工程中搜索指定的字符串,并将搜索的结果在立即窗口打印输出,其实现代码:

```
Public Sub SearchString(ByVal strFind As String, ByVal project As VBProject, ByRef count As Long)
    Dim lngStartLine As Long
    Dim lngStartCol As Long
    Dim lngEndLine As Long
    Dim lngEndCol As Long

    '将计数器清零
    count = 0

    Dim i As Integer
    Dim module As VBComponent

    Debug.Print "工程：" & project.FileName

    For i = 1 To project.VBComponents.count
        Set module = project.VBComponents.Item(i)
        Debug.Print "模块：" & module.Name

        With module.CodeModule
            lngStartLine = 1
            lngStartCol = 1
            lngEndLine = .CountOfLines
            lngEndCol = Len(.Lines(.CountOfLines, 1))

            Do While .Find(strFind, lngStartLine, lngStartCol, lngEndLine, lngEndCol, True, False)
                Debug.Print "Line" & lngEndLine; Tab(30); .ProcOfLine(lngEndLine, 0)

                '重新设置查找参数
                lngStartLine = lngEndLine + 1
                lngStartCol = 1
                lngEndLine = .CountOfLines
                lngEndCol = Len(.Lines(.CountOfLines, 1))

                count = count + 1
            Loop
        End With
    Next i
End Sub
Find方法的定义为(从对象浏览器中获得)：
Function Find(Target As String, StartLine As Long, StartColumn As Long, EndLine As Long,
    EndColumn As Long, [WholeWord As Boolean = False], [MatchCase As Boolean = False],
    [PatternSearch As Boolean = False]) As Boolean
```

Target 为要查找的字符串，StartLine 是开始查找的行号，StartColumn 是开始查找的列号，EndLine 指定结束查找的行号，EndColumn 指定结束查找的列号，WholeWord 指定是否要求全字匹配，MatchCase 指定是否要求大小写匹配，PatternSearch 的含义不清楚。

(6) 添加一个过程 SearchStringInFolder，用于在指定的文件夹中访问所有的 DVB 文件，搜索包含指定字符串的位置，其实现代码为：

```
Public Sub SearchStringInFolder(ByVal strFind As String, ByVal strPath As String)
    Dim fileCollection As New Collection
    Dim count As Long, totalCount As Long
    totalCount = 0

    '查找该文件夹中所有的 DVB 文件
    mdlSearchFile.FindAllFiles " *.dvb", strPath, fileCollection

    '在所有的 DVB 文件中查找 strFind 字符串
    Dim i As Long
    Dim objProject As VBProject
    For i = 1 To fileCollection.count
        '加载 DVB 文件
        ThisDrawing.Application.LoadDVB fileCollection.Item(i)
        Set objProject = ThisDrawing.Application.VBE.VBProjects.Item(2)

        '在指定的工程中搜索 strFind 字符串
        Call SearchString(strFind, objProject, count)
        Debug.Print " ============================================== "
        totalCount = totalCount + count

        '卸载 DVB 文件
        ThisDrawing.Application.UnloadDVB fileCollection.Item(i)
    Next i
    Debug.Print "找到(" & totalCount; ")个结果。"
End Sub
```

注意下面的一句代码：

```
Set objProject = ThisDrawing.Application.VBE.VBProjects.Item(2)
```

Item 方法允许通过索引值或者 Name 来指定工程，但是很多用户编写 VBA 工程时并不习惯修改 VBA 工程的名称(在“工程资源管理器”窗口中单击工程的根目录，即可在属性窗口中修改)，因此可能会存在重名的工程，只能用索引值来指定工程。

由于这里的索引值被固定为 2，也就意味着在执行该过程时，要求 VBA 集成开发环境中仅加载了当前工程，而不包含其他的工程，否则就可能导致执行的失败。

16.3.4　实例效果

(1) 在 VBA 集成开发环境中，执行 GetPathOfAllProjects 宏，获得当前加载的所有工程列表，如图 16.13 所示。

(2) 在 ThisDrawing 代码窗口中添加一个宏 ShowAllProcesses，用于显示当前工程 ThisDrawing 窗口所有的过程，其相关代码：

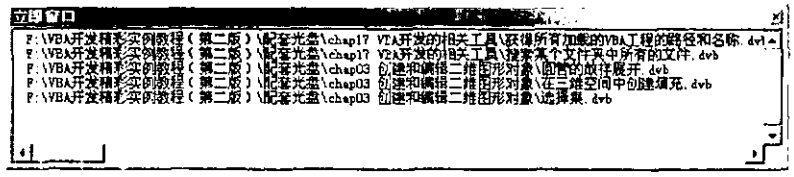


图 16.13 获得加载的 VBA 工程列表

```
Public Sub ShowAllProcesses( )
        Call
ListProcess(ThisDrawing.Application.VBE.ActiveVBProject.VBComponents("ThisDrawing").CodeModule)
    End Sub
```

(3)执行 ShowAllProcesses 宏,能够得到如图 16.14 所示的结果。注意,这里显示的就是用 Tab 函数对齐列的效果。

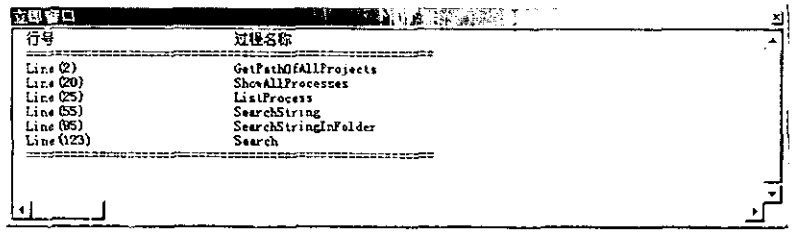


图 16.14 显示当前工程 ThisDrawing 模块的过程和函数

(4) 添加一个宏 Search,用于搜索本书配套光盘中第 3 章的所有程序,查找包含 GetPoint 字符串的位置,其实现代码:

```
Public Sub Search( )
    Call SearchStringInFolder("GetPoint", _
        "F:\VBA 开发精彩实例教程(第二版)\配套光盘\chap03 创建和编辑二维图形对象")
    End Sub
```

(5) 执行 Search 宏,能够得到如图 16.15 所示的结果。

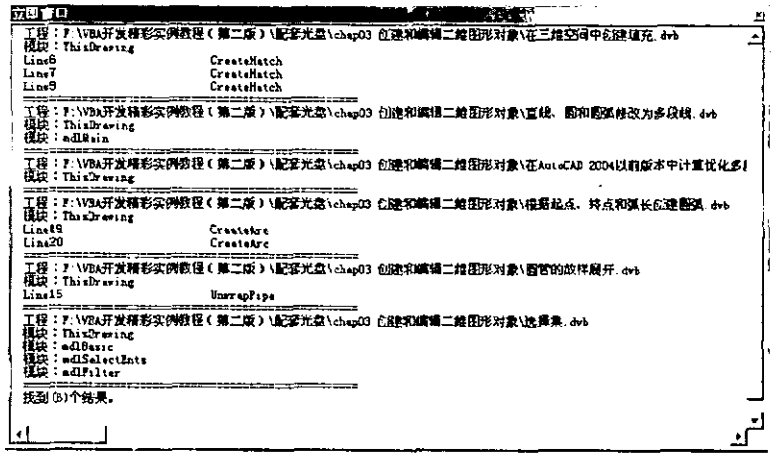


图 16.15 搜索字符串的结果

16.3.5　小结

1. 关于程序搜索的工具

很久以来，在遇到问题的时候，就羡慕 VC 6.0 中提供的“在文件中查找”的功能，搜索之后能在结果栏中给出详细的位置，如图 16.16 所示。在某个位置上双击就可以直接在代码窗口中打开对应的文件，定位在该行代码处，这对于初学者来说，非常方便！

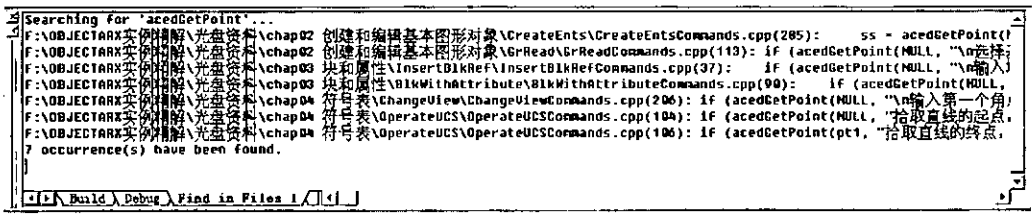


图 16.16　VC 中“在文件中查找”的结果

不可否认，本节中实现的“在文件中查找”功能还是比较原始的，但是基本原理已经完全剖析出来，读者还可以自己试着用一个非模态对话框来模拟 VC 中的这种效果。

2. 检测一个 DVB 文件是否加载

```
'检测一个 DVB 文件是否加载
'strFile：要检测的 DVB 文件的名称；bLoad：如果没有加载，是否加载该文件
Public Function IsLoaded(ByVal strFile As String, ByVal bLoadIt As Boolean) As Boolean
    Dim i As Integer
    For i = 1 To ThisDrawing.Application.VBE.VBProjects.Count
        If StrComp(ThisDrawing.Application.VBE.VBProjects(i).Name, strFile, vbTextCompare)
            = 0 Then
            IsLoaded = True
            Exit Function
        End If
    Next i

    '如果没有加载，是否需要加载
    If bLoadIt Then
        ThisDrawing.Application.LoadDVB strFile
    End If

    IsLoaded = False
End Function
```

3. 在宏中传递参数

从 AutoCAD 直接传递参数到一个 VBA 宏中是不可能的，但是可以用 VBA 的 GetString 方法和 LISP 的 command 函数来传递信息。

可以定义一个 VBA 宏：

```
Sub TestParams( )
    Dim str1, str2 As String
    str1 = ThisDrawing.Utility.GetString(False)
    str2 = ThisDrawing.Utility.GetString(False)

    MsgBox str1
    MsgBox str2
End Sub
```

然后用以下 AutoLISP 语句调用此宏：

```
(command "-VBARUN" "TestParams" "param1" "param2")
```

附录 A　AutoCAD ActiveX 和 VBA 参考

A.1　对象

对象列表如表 A.1 所示。

表 A.1　对象

对象	描述
3DFace	一个三维三角形或四边形平面
3dPolyline	由直线线段组成的三维多段线
3DSolid	支持自由外形表面的实体对象
_DAcadApplicationEvents 接口	AutoCAD 应用程序的事件接口
_DAcadDocumentEvents 接口	AutoCAD 文档的事件接口
AcadState	一种特殊的对象，用于其他进程应用程序监视 AutoCAD 的状态
AcCmColor	AutoCAD 真彩色对象
Application	一个 AutoCAD 应用程序实例
Arc	圆弧
Attribute	属性，文字字符串形式的对象，描述属性参照的特性
AttributeReference	属性参照，含有连接到块的文字的对象
Block	包含一个名称和一组对象的块定义
BlockRef	块定义插入到图形中的一个实例
Blocks 集合	图形中所有块的集合
Circle	完整的圆
Database	外部参照块的内容
DatabasePreferences	该对象指定当前 AutoCAD 图形特有的设置值
Dictionaries 集合	图形中所有字典的集合
Dictionary	字典，用于保存和检索对象的容器对象
Dim3PointAngular	测量三点间角度的标注
DimAligned	线性标注，测量两点间距离，平行于被测量点显示
DimAngular	测量两线夹角或圆弧包含角度的标注
DimArcLength	测量弧长的标注（AutoCAD 2006 新增）
DimDiametric	测量圆或圆弧直径的标注
DimOrdinate	测量某点到原点绝对 X 或 Y 坐标值的标注
DimRadial	测量圆或圆弧半径的标注
DimRadialLarge	测量大圆弧的标注，也称为折角半径标注（AutoCAD 2006 新增）
DimRotated	测量两点间距离，并按指定角度显示的标注
DimStyle	标注样式，可决定标注外观的一组标注设置
DimStyles 集合	图形中所有标注样式的集合

续表

对象	描述
Document	一个 AutoCAD 图形
Documents 集合	在当前进程中所有打开的 AutoCAD 图形的集合
DynamicBlock-ReferenceProperty	存储动态块属性的对象(AutoCAD 2006 新增)
Ellipse	椭圆弧或完整椭圆
ExternalReference	插入到图形的外部参照的实例
FileDependency	提供关于一个图形文件的文件相关表中条目的信息
Group	编组,命名的 SelectionSet 对象
Groups 集合	图形中所有编组的集合
Hatch	图案填充,由直线图案组成的区域填充
Hyperlink	超级链接,URL 和 URL 描述
IAcadDatabase 接口	提供一个到 AutoCAD 文档中的数据库常驻对象的接口
IAcadDimension 接口	AutoCAD 标注的通用接口
IAcadEntity 接口	基本 AutoCAD 图元的标准接口
IAcadObject 接口	基本 AutoCAD 对象的标准接口
IAcadObjectEvents 接口	基本 AutoCAD 对象的事件接口
IDPair	供 CopyObjects 方法使用的特殊对象,它包含源对象和目标对象的对象识别码(ID)
Layer	图层,数据的逻辑分组,类似于覆叠于图形上的透明胶片
Layers 集合	图形中所有图层的集合
LayerStateManager	图层状态管理器,保存和恢复图层的设置
Layout	布局,模型空间或图纸空间块的打印设置和显示特性
Layouts 集合	图形中所有布局的集合
Leader	引线,由附着到样条曲线或直线段的箭头所组成的对象
LightweightPolyline	优化多段线,由线和弧段组成的可调节宽度的二维线
Line	直线,单一线段
Linetype	线型,结合虚线、点和空格所组成的线的特征
Linetypes 集合	图形中的线型的集合
MenuBar 集合	PopupMenu 对象的集合,代表当前 AutoCAD 菜单列
MenuGroup	AutoCAD 菜单组
MenuGroups 集合	MenuGroup 对象的集合,代表已加载到当前 AutoCAD 进程的所有菜单组
MInsertBlock	块参照的矩形阵列
MLine	多线,多重平行线
ModelSpace 图元集合	包含所有模型空间图元的特殊的 Block 对象
MText	多行文字,放到非打印文字边界内的文字、数字字符串段落
PaperSpace 图元集合	包含当前图纸空间布局中所有图元的特殊 Block 对象
Plot	打印,用于打印布局的方法和属性集
PlotConfiguration	打印设置的命名集合对象
PlotConfigurations 集合	命名打印设置的集合
Point	点,作为点、正方形、圆、X 形、短线或加号(+),或者这些符号组合起来形成的点标记
PolyfaceMesh	三维多段面网格

续表

对象	描述
PolygonMesh	自由形式的三维网格
Polyline	多段线，可调整宽度的二维线，或不可调整宽度的由线和弧段组成的三维线或网络
PopupMenu	AutoCAD 下拉菜单
PopupMenuItem	AutoCAD 下拉菜单中的一个菜单项
PopupMenus 集合	*PopupMenu* 对象的集合，代表加载到 *MenuGroup* 中的所有弹出式菜单
Preferences	该对象指定当前的 AutoCAD 设置
PreferencesDisplay	该对象包含“选项”对话框中“显示”选项卡中的选项
PreferencesDrafting	该对象包含“选项”对话框中“草图”选项卡中的选项
PreferencesFiles	该对象包含“选项”对话框中“文件”选项卡中的选项
PreferencesOpenSave	该对象包含“选项”对话框中“打开和保存”选项卡中的选项
PreferencesOutput	该对象包含“选项”对话框中“打印”选项卡中的选项
PreferencesProfiles	该对象包含“选项”对话框中“配置”选项卡中的选项
PreferencesSelection	该对象包含“选项”对话框中“选择”选项卡中的选项
PreferencesSystem	该对象包含“选项”对话框中“系统”选项卡中的选项
PreferencesUser	该对象包含“选项”对话框中“用户系统配置”选项卡中的选项
PViewport	浮动视口，在图纸空间中创建的用于显示视图的矩形对象
Raster	光栅，由小方块形成的矩形栅格（或光栅），或称为像素的点组成的光栅图像
Ray	射线，半边无限延长的直线
Region	面域，由线、圆、弧、椭圆弧及样条曲线组成的带边界平面
RegisteredApplication	已经添加到图形中的外部应用程序
RegisteredApplication 集合	图形中所有已注册应用程序的集合
SecurityParams	保密参数，提供图形文件上执行保密操作所必须的信息
SelectionSet	选择集，作为单一单元处理的一个或多个 AutoCAD 对象组成的组合
SelectionSets 集合	图形中所有选择集的集合
Shape	形，由 SHX 文件中定义的线、弧和圆组成的对象
Solid	实体，一个实体填充多边形
SortentsTable	包含并操作绘图顺序信息
Spline	样条曲线，二次或三次 NURBS（非均匀有理 B-样条）曲线
SummaryInfo	访问图形属性，如标题、主题、作者和关键字属性
Table	AutoCAD 表格对象
TableStyle	表格样式，用于添加、修改和删除表格样式
Text	单行文本对象
TextStyle	文字样式（或称字型），已命名并保存的用于确定文字字符串外观的设置集
TextStyles 集合	图形中所有文字样式的集合
Tolerance	包含在特征控制框中的几何公差
Toolbar	AutoCAD 工具栏
ToolbarItem	在 AutoCAD 工具栏上的一个按钮项
Toolbars	Toolbar 对象的集合，代表加载到当前 AutoCAD 进程的所有工具栏

续表

对象	描述
Trace	宽线，指定宽度的二维实体线
UCS	由用户指定三维空间中 X、Y 和 Z 轴方向自定义的坐标系
UCSs 集合	图形中所有用户坐标系(UCS)的集合
Utility	提供一系列实用工具方法
View	从空间中特定位置(观察点)来表现二维图形或三维模型的图形
Viewport	平铺视口，显示一个图形的模型空间的某一部分的边界区域
Viewports 集合	图形中所有视口的集合
Views	图形中所有视图的集合
XLine	两边无限延伸的构造线
XRecord	XRecord 对象用于存储和管理任意数据

A.2 方法

方法列表如表 A.2 所示。

表 A.2 方法

方法	描述
Activate	激活指定图形，将其作为当前图形
Add	创建一个成员对象并将它添加到适当的集合中，所有的集合对象均可使用该方法添加元素
Add3DFace	给定 4 个顶点创建 3DFace 对象
Add3DMesh	给定 M 和 N 方向点的数量和 M 和 N 方向点的坐标，创建自由形式的三维网格
Add3DPoly	给定坐标值数组创建三维多段线
AddArc	给定弧的圆心、半径、起始角度和终止角度来创建圆弧
AddAttribute	用指定特性在给定的位置创建属性定义
AddBox	创建边与 WCS 的轴相平行的三维长方体实体
AddCircle	给定中心点和半径创建圆
AddCone	创建基底在 WCS 的 XY 平面的三维圆锥体实体
AddCustomInfo	添加自定义属性字段到现有字段列表的尾部
AddCustomObject	创建自定义对象
AddCylinder	创建基底在 WCS 的 XY 平面的三维圆柱体实体
AddDim3PointAngular	使用三点创建角度标注
AddDimAligned	创建对齐标注对象
AddDimAngular	创建圆弧、两条线或圆的角度标注
AddDimArc	创建圆弧的弧长标注(AutoCAD 2006 新增)
AddDimDiametric	给定直径上的两点和引线长度创建圆或弧的直径标注
AddDimOrdinate	给定定义点和引线终点创建坐标标注

续表

方法	描述
AddDimRadial	在给定位置创建选定对象的半径标注
AddDimRadialLarge	创建圆弧、圆或多段线弧段的折角半径标注(AutoCAD 2006 新增)
AddDimRotated	创建旋转的线性标注
AddEllipse	给定中心点、长轴上的一点和半径比在 WCS 的 XY 平面上创建椭圆
AddEllipticalCone	给定圆心、长轴半径、短轴半径和高度在 WCS 的 XY 平面上创建三维实体椭圆锥体
AddEllipticalCylinder	给定圆心、长轴半径、短轴半径和高度创建基底在 WCS 的 XY 平面上的三维实体椭圆柱体
AddExtrudedSolid	给定截面、高度和倾斜角度创建拉伸实体
AddExtrudedSolidAlongPath	给定截面和拉伸路径创建拉伸实体
AddFitPoint	按给定索引位置添加拟合点到样条曲线上
AddHatch	创建图案填充对象
AddItems	添加一个或多个对象到指定选择集
AddLeader	给定坐标点创建引线
AddLightweightPolyline	由顶点列表创建优化多段线
AddLine	通过两个点创建直线
AddMenuItem	添加弹出菜单项到弹出菜单
AddMInsertBlock	插入块的阵列
AddMLine	通过点数组创建多线
AddMText	在由插入点和边框宽度所定义的矩形中创建多行文字
AddObject	添加对象到命名的词典中
AddPoint	在给定位置创建点对象
AddPolyfaceMesh	从顶点列表创建多面网格
AddPolyline	从顶点列表创建多段线
AddPViewport	以给定中心点、高度和宽度添加图纸空间的视口
AddRaster	基于现有图像文件创建新的光栅图像
AddRay	创建通过两个点的惟一的射线
AddRegion	从一组图元创建面域,给定的图元必须构成闭合的共面区域
AddRevolvedSolid	以给定的面域绕轴创建旋转实体
AddSeparator	向现有菜单或工具栏添加分隔符
AddShape	基于给定名称的样板、给定插入点、比例因子和旋转角度创建形对象
AddSolid	创建二维实心多边形
AddSphere	以给定的球心和半径创建球体
AddSpline	创建二次或三次 NURBS(非一致有理 B 样条)曲线
AddSubMenu	向现有菜单添加子菜单
AddTable	添加表格到图形中
AddText	创建单行文字
AddTolerance	创建公差图元
AddToolbarButton	添加工具栏项目到工具栏的指定位置
AddTorus	在给定位置创建圆环
AddTrace	根据一组点创建宽线对象
AddVertex	向优化多段线中添加顶点

续表

方法	描述
AddWedge	给定长度、宽度和高度创建边与轴平行的楔体
AddXLine	创建通过两个指定点构造线(无限长线)
AddXRecord	在任何字典中创建扩展记录(XRecord)对象
AngleFromXAxis	获取直线与 X 坐标轴正半轴的夹角
AngleToReal	将角度值从字符串转换为双精度实数值
AngleToString	将角度从双精度实数值转换为字符串
AppendInnerLoop	向图案填充中附加内边界
AppendItems	向指定的组中附加一个或多个图元
AppendOuterLoop	向图案填充中附加外边界
AppendVertex	向三维多段线、多段线或多段面网格对象的最后附加顶点
ArrayPolar	以给定的对象数量、填充角度和中心点创建对象的环形阵列
ArrayRectangular	创建对象的二维或三维矩形阵列
AttachExternalReference	将外部参照附着到图形
AttachToolbarToFlyout	将工具栏附着到定义为弹出式的工具栏按钮
AuditInfo	评估图形的完整性
Bind	将外部引用(xref)并入到图形中
Block	返回 SortentsTable 对象的块
Boolean	执行对象与其他三维实体或面域对象之间的布尔运算(并集、交集或差集)
CheckInterference	检查两实体间的干涉,如果指定创建干涉实体,则由干涉部分创建实体
Clear	清除指定选择集中的所有项目
ClearSubSelection	移去表格中单元格的子选择集
ClearTableStyleOverrides	清除表格样式替换
ClipBoundary	为光栅图像指定剪裁边界
Close	关闭指定的图形或所有打开的图形
ConvertToAnonymousBlock	转换动态块为常规的匿名块(AutoCAD 2006 新增)
ConvertToStaticBlock	转换动态块为常规的命名块(AutoCAD 2006 新增)
Copy	在同一位置复制给定对象
CopyFrom	从给定的标注样式、布局或打印配置中复制设置
CopyObjects	复制多个对象(完全克隆)
CopyProfile	复制指定的配置
CreateEntry	在文件相关表中创建新的条目
CreateTypedArray	创建包含一组各种参数的变体
Delete	删除指定对象或一组保存的图层设置
DeleteCellContent	删除指定行和列的单元格内容
DeleteColumns	删除表格中的列
DeleteConfiguration	删除视口配置
DeleteFitPoint	删除样条曲线上给定索引位置的拟合点
DeleteProfile	删除指定的配置
DeleteRows	删除表格中的行
Detach	从图形中分离外部参照(xref)
Display	切换 PViewport 对象的显示控制为打开或关闭

续表

方法	描述
AddWedge	给定长度、宽度和高度创建边与轴平行的楔体
AddXLine	创建通过两个指定点构造线(无限长线)
AddXRecord	在任何字典中创建扩展记录(XRecord)对象
AngleFromXAxis	获取直线与 X 坐标轴正半轴的夹角
AngleToReal	将角度值从字符串转换为双精度实数值
AngleToString	将角度从双精度实数值转换为字符串
AppendInnerLoop	向图案填充中附加内边界
AppendItems	向指定的组中附加一个或多个图元
AppendOuterLoop	向图案填充中附加外边界
AppendVertex	向三维多段线、多段线或多段面网格对象的最后附加顶点
ArrayPolar	以给定的对象数量、填充角度和中心点创建对象的环形阵列
ArrayRectangular	创建对象的二维或三维矩形阵列
AttachExternalReference	将外部参照附着到图形
AttachToolbarToFlyout	将工具栏附着到定义为弹出式的工具栏按钮
AuditInfo	评估图形的完整性
Bind	将外部引用(xref)并入到图形中
Block	返回 SortentsTable 对象的块
Boolean	执行对象与其他三维实体或面域对象之间的布尔运算(并集、交集或差集)
CheckInterference	检查两实体间的干涉,如果指定创建干涉实体,则由干涉部分创建实体
Clear	清除指定选择集中的所有项目
ClearSubSelection	移去表格中单元格的子选择集
ClearTableStyleOverrides	清除表格样式替换
ClipBoundary	为光栅图像指定剪裁边界
Close	关闭指定的图形或所有打开的图形
ConvertToAnonymousBlock	转换动态块为常规的匿名块(AutoCAD 2006 新增)
ConvertToStaticBlock	转换动态块为常规的命名块(AutoCAD 2006 新增)
Copy	在同一位置复制给定对象
CopyFrom	从给定的标注样式、布局或打印配置中复制设置
CopyObjects	复制多个对象(完全克隆)
CopyProfile	复制指定的配置
CreateEntry	在文件相关表中创建新的条目
CreateTypedArray	创建包含一组各种参数的变体
Delete	删除指定对象或一组保存的图层设置
DeleteCellContent	删除指定行和列的单元格内容
DeleteColumns	删除表格中的列
DeleteConfiguration	删除视口配置
DeleteFitPoint	删除样条曲线上给定索引位置的拟合点
DeleteProfile	删除指定的配置
DeleteRows	删除表格中的行
Detach	从图形中分离外部参照(xref)
Display	切换 PViewport 对象的显示控制为打开或关闭

续表

方法	描述
GetCellType	返回指定行和列的单元格类型
GetColor	返回指定行类型的真彩色值
GetColumnWidth	返回表格中指定列索引所在列的列宽
GetConstantAttributes	获取块或外部参照中的固定属性
GetContentColor	返回指定行类型的真彩色值
GetControlPoint	返回给定索引的样条曲线控制点
GetCorner	获取矩形的角点
GetCustomByIndex	返回与索引值对应的自定义属性的名称(键)和值
GetCustomByKey	返回与名称(键)值对应的自定义属性值
GetCustomScale	获取布局或打印配置的自定义比例
GetDistance	获取由提示行或由屏幕上选定的一组点的距离
GetDynamicBlockProperties	获取动态块的属性
GetEntity	以交互方式获取对象
GetExtensionDictionary	获取与对象关联的扩展词典
GetFieldId	返回指定单元格的字段对象 ID
GetFitPoint	获取样条曲线上给定索引位置的拟合点
GetFont	获取 TextStyle 的字体定义数据
GetFullDrawOrder	返回块中的所有对象,按最底下的对象为第一个的绘图顺序排序
GetGridColor	返回指定格线类型和行类型的格线颜色值
GetGridLineWeight	返回指定格线类型和行类型的格线线宽值
GetGridSpacing	获取视口的栅格间距
GetGridVisibility	返回指定格线类型和行类型的格线可见性值
GetInput	将用户输入的字符串转换为关键词索引
GetInteger	从用户处获取整数值
GetInterfaceObject	接收程序 ID,并试图将其作为进程内服务器加载到 AutoCAD 中
GetInvisibleEdge	获取 3DFace 对象上给定索引位置的边的可见性设置
GetKeyword	从用户处获取关键词字符串
GetLocaleMediaName	获取标准介质名称的本地版本
GetLoopAt	获取给定索引位置的图案填充边界
GetMinimumColumnWidth	获取表格中指定列索引所在的列的最小列宽
GetMinimumRowHeight	返回指定行的最小行高
GetName	获取词典中对象的名称(关键字)
GetObject	获取词典中给定名称(关键字)的对象
GetOrientation	获取指定角度,忽略 ANGBASE 系统变量的设置
GetPaperMargins	获取布局或打印配置的页边距
GetPaperSize	获取配置的图纸的宽度和高度
GetPlotDeviceNames	获取所有可用的打印设备名称
GetPlotStyleTableNames	获取所有可用的打印样式表名称
GetPoint	获取 AutoCAD 中选定的点
GetProjectFilePath	获取 AutoCAD 查找外部参照文件的目录
GetReal	从用户获取双精度实数值
GetRelativeDrawOrder	返回指定的对象,它按最底下的对象为第一个的绘图顺序排序

续表

方法	描述
GetRemoteFile	下载由 URL 指定的文件
GetRowHeight	返回表格中指定行索引所在行的行高
GetRowType	返回指定行的行类型
GetSnapSpacing	获取视口的捕捉间距
GetString	从用户获取字符串
GetSubEntity	以交互方式获取对象或子图元
GetSubSelection	返回在子选择集中单元格的行和列索引
GetText	返回指定行和列的文字值
GetTextHeight	返回指定行类型的文字高度
GetTextRotation	指定行和列的文字旋转角度
GetTextStyle	返回指定行类型的文字样式名称
GetUCSMatrix	获取由 UCS 坐标系数据组成的转换矩阵
GetVariable	获取 AutoCAD 系统变量的当前设置
GetWeight	返回给定控制点索引的样条曲线权值
GetWidth	返回多段线的起始和终止宽度
GetWindowToPlot	获取定义要打印的布局范围的坐标
GetXData	获取与对象关联的扩展数据(Xdata)
GetXRecordData	获取与词典相关联的扩展记录数据(XrecordData)
HandleToObject	获取与给定句柄对应的对象
Highlight	设置给定对象或给定选择集中所有对象的亮显状态
HitTest	返回指定位置的单元格
Import	从文件输入图形或一组保存的图层设置
ImportProfile	输入由其他用户创建的配置
IndexOf	提供文件相关表中条目的索引
InitializeUserInput	初始化 GetKeyword 方法
InsertBlock	插入图形文件或当前图形中已定义的命名块
InsertColumns	在表格中插入列
InsertInMenuBar	将弹出菜单插入 AutoCAD 菜单栏的指定位置
InsertLoopAt	按给定索引在图案填充中插入边界
InsertMenuInMenuBar	将菜单插入 AutoCAD 菜单栏中
InsertRows	在表格中插入行
IntersectWith	获取对象与图形中其他对象的相交点
IsMergedCell	返回单元格的合并状态
IsRemoteFile	返回下载远程文件的源 URL
IsURL	验证给定的 URL
Item	获取集合、组或选择集中给定索引的成员对象
LaunchBrowserDialog	启动 Web 浏览器窗口,允许用户导航到任意 URL,并可指定 URL
ListArx	获取当前加载的 AutoCAD ARX 应用程序
Load	从菜单或自定义(CUI)文件中加载菜单组,或从库(LIN)文件中加载线型定义
LoadARX	加载指定的 AutoCAD ARX 应用程序
LoadDVB	加载指定的 AutoCAD VBA 工程文件

续表

方法	描述
LoadShapeFile	加载形文件(SHX)
MergeCells	合并表格中的单元格
Mirror	围绕轴创建平面对象的镜像图像副本
Mirror3D	绕由三点定义的平面镜像选定的对象
Move	将图元对象从源向目标移动
MoveAbove	将对象移动到绘图顺序中目标的上面
MoveBelow	将对象移动到绘图顺序中目标的下面
MoveToBottom	将对象移动到绘图顺序的最下面
MoveToTop	将对象移动到绘图顺序的最上面
New	在 SDI(单文档)模式中创建新文档
NumCustomInfo	返回已设置的自定义属性字段的数量
ObjectIDToObject	获取与给定对象 ID 相对应的对象
Offset	创建由现有对象偏移指定距离的新对象
Open	打开现有图形文件(DWG),并将其设为活动的文档
PlotToDevice	打印布局到设备
PlotToFile	打印布局到指定文件
PolarPoint	获取与给定点指定角度和距离的点
Prompt	向命令行发送提示
PurgeAll	从文档中删除未使用的命名参照,例如未使用的块或图层
PurgeFitData	清理样条曲线的拟合数据
PutRemoteFile	将文件上载到由 URL 指定的远程位置
Quit	关闭图形文件,并退出 AutoCAD 应用程序
RealToString	将双精度实数值转换为字符串
RecomputeTableBlock	更新表格块
RefreshPlotDeviceInfo	更新打印机、规范介质和打印样式表信息,以反映当前系统状态
Regen	重生成整个图形并重计算所有对象的屏幕坐标和显示精度
Reload	重载外部参照
Remove	从字典中删除命名对象
RemoveCustomByIndex	删除与索引值对应的自定义属性名称(键)和值
RemoveCustomByKey	删除与名称(键)值关联的自定义属性值
RemoveEntry	从文件相关表中移除指定的条目
RemoveFromMenuBar	从 AutoCAD 菜单栏中删除弹出菜单
RemoveItems	从组或选择集中删除指定的项目
RemoveMenuFromMenuBar	从 AutoCAD 菜单栏中删除集合中所指定的弹出菜单
Rename	重新命名字典中的项目或保存的图层设置
RenameProfile	重命名指定的配置
Replace	用给定项目替换字典中的项目
ReselectSubRegion	返回表格中选择集的子图元
ResetBlock	重置动态块的默认状态
ResetProfile	将指定配置中的值重置为默认值
Restore	恢复一组图层属性设置
Reverse	反转样条曲线的方向

续表

方法	描述
Rotate	绕一点旋转图元对象
Rotate3D	绕三维直线旋转图元对象
RunMacro	从 Application 对象中运行 VBA 宏
Save	保存文档、菜单或图层属性设置，AutoCAD 2006 版不再支持菜单组的保存
SaveAs	将文档或菜单组保存到指定的文件，AutoCAD 2006 版不再支持菜单组的保存
ScaleEntity	按指定的基点和比例因子来缩放图元对象
SectionSolid	以给定三点创建三维实体对象的截面，将该截面返回为面域对象
Select	在选择集对象中调用，选择对象并将其添加到选择集中
Select	在表格对象中调用，选择表格中的单元格
SelectAtPoint	选择通过给定点的对象，并将其添加到选择集中
SelectByPolygon	栏选对象并将其添加到选择集中
SelectOnScreen	提示用户从屏幕上拾取对象
SelectSubRegion	选择表格中一些单元格
SendCommand	从 VB 或 VBA 应用程序向文档发送命令字符串以进行处理
SendModelessOperationEnded	指出非模态操作已经结束
SendModelessOperationStart	指出非模态操作即将开始
SetAlignment	设置指定行类型的单元格对齐方式
SetAutoScale	设置指定的行和列是否使用自动缩放比例的值
SetBackgroundColor	设置指定行类型的背景颜色值
SetBackgroundColorNone	设置指定的行类型是否没有背景颜色的值
SetBitmaps	设置用作工具栏项目图标的大位图和小位图
SetBlockAttributeValue	设置指定单元格的块中所包含的属性定义对象的属性值
SetBlockRotation	设置指定行和列的块的旋转角度
SetBlockScale	设置指定行和列的块的比例因子
SetBlockTableRecordId	设置单元格的块表格记录 ID
SetBulge	设置多段线在给定索引位置的凸度
SetCellAlignment	设置指定行和列的单元格的对齐方式
SetCellBackgroundColor	设置指定行和列的单元格的背景真彩色值
SetCellBackgroundColorNone	设置指定的行和列是否无背景颜色
SetCellContentColor	设置指定行和列中内容的真彩色值
SetCellGridColor	设置指定行和列的一个边的格线颜色值
SetCellGridLineWeight	设置指定行和列的一个边的格线线宽
SetCellGridVisibility	设置指定行和列的一个边的格线可见性值
SetCellTextHeight	设置指定行和列的文字高度
SetCellTextStyle	设置指定行和列的文字样式名称
SetCellType	设置指定行和列的单元格类型
SetColor	设置指定行类型的真彩色值
SetColorBookColor	将颜色设置为配色系统中的颜色
SetColumnWidth	设置表格中指定列索引所在列的列宽
SetContentColor	设置指定行类型的真彩色值

续表

方法	描述
SetControlPoint	设置样条曲线在给定索引的控制点
SetCustomByIndex	设置与索引值对应的自定义属性名称(键)
SetCustomByKey	设置与名称(键)值对应的自定义属性值
SetCustomScale	设置布局或打印配置的自定义比例
SetDatabase	设置图层状态管理器的工作数据库
SetFieldId	设置指定单元格的字段对象 ID
SetFitPoint	设置样条曲线的拟合点
SetFont	设置文字样式中字体的定义数据
SetGridColor	设置指定格线类型和行类型的格线颜色值
SetGridLineWeight	设置指定格线类型和行类型的格线线宽值
SetGridSpacing	设置视口的栅格间距
SetGridVisibility	设置指定格线类型和行类型的格线可见性值
SetInvisibleEdge	设置给定索引位置边的可见状态
SetLayoutsToPlot	指定要打印的布局
SetNames	指定颜色的颜色名和配色系统名称
SetPattern	设置图案填充的图案类型和名称
SetProjectFilePath	设置 AutoCAD 查找外部参照文件的目录
SetRelativeDrawOrder	设置要指定相对绘图顺序的对象,最底下的对象为第一个
SetRGB	指定真彩色的 RGB 值
SetRowHeight	设置表格中指定行索引所在行的行高
SetSnapSpacing	设置视口的捕捉间距
SetSubSelection	设置在子选择集中单元格的行和列索引
SetText	设置指定行和列的文字值
SetTextHeight	设置指定行类型的文字高度
SetTextRotation	设置行和列的文字旋转角度
SetTextStyle	设置指定行类型的文字样式名称
SetVariable	设置 AutoCAD 系统变量的值
SetView	将视口中的视图设置为保存在 Views 集合对象中的视图
SetWeight	设置给定控制点索引的样条曲线权值
SetWidth	设置多段线上某一段(给定索引)的起始和终止宽度
SetWindowToPlot	设置定义要打印的布局范围的坐标
SetXData	设置与对象关联的扩展数据(外部数据)
SetXRecordData	设置与词典相关联的扩展记录数据(XrecordData)
SliceSolid	以给定三点创建三维实体的剖切面
Split	将视口分割为给定数目的视图
StartBatchMode	调用批处理打印模式
StartUndoMark	标记操作块的开始
SwapOrder	交换两个对象的绘图顺序位置
TransformBy	给定 4×4 转换矩阵移动、缩放或旋转对象
TranslateCoordinates	将点从一个坐标系转换到另一个坐标系
Unload	卸载菜单组或者外部参照
UnloadARX	卸载指定的 AutoCAD ARX 应用程序

续表

方法	描述
UnloadDVB	卸载指定的 AutoCAD VBA 工程文件
UnmergeCells	拆散表格中的合并单元格
Update	更新图形屏幕的对象
UpdateEntry	在文件相关表中删除，然后重新创建一个指定的实体
WBlock	将给定选择集导出为新的图形文件
ZoomAll	用于缩放当前视图，显示整个图形
ZoomCenter	通过指定缩放中心点和比例缩放当前视图
ZoomExtents	将当前视图缩放到图形界限
ZoomPickWindow	由用户在绘图区域指定窗口缩放当前视图
ZoomPrevious	返回到最近使用过的缩放视图
ZoomScaled	按照指定的比例因子缩放当前视图
ZoomWindow	通过指定矩形的两个对角点来缩放当前视图

A.3 属性

属性列表如表 A.3 所示。

表 A.3 属性

属性	描述
Action	指定在图形中要执行的与安全相关的操作
Active	判断文档在进程中是否为活动的文档
ActiveDimStyle	指定活动的标注样式
ActiveDocument	指定活动的文档(图形文件)
ActiveLayer	指定活动的图层
ActiveLayout	指定活动的布局
ActiveLinetype	指定图形的活动线型
ActiveProfile	指定 AutoCAD 进程的活动配置
ActivePViewport	指定图形中活动的图纸空间视口
ActiveSelectionSet	获取图形中活动的选择集
ActiveSpace	在图纸空间和模型空间之间切换活动的空间
ActiveTextStyle	指定图形中活动的文字样式
ActiveUCS	指定图形中活动的 UCS
ActiveViewport	指定图形中活动的视口
ADCInsertUnitsDefaultSource	确定在 AutoCAD 设计中心源图形未分配插入单位的对象自动使用的单位
ADCInsertUnitsDefaultTarget	确定在 AutoCAD 设计中心目标图形未分配插入单位的对象自动使用的单位
AffectsGraphics	返回条目是否影响图形
Algorithm	指定加密算法标识符

续表

属性	描述
Alignment	指定属性、属性参照或文字的水平和垂直对齐
AlignmentPointAcquisition	指定如何获取 AutoAlignment(自动对齐)点
AllowedValues	指定属性的允许值
AllowLongSymbolNames	决定符号名是否可以包含扩展字符集,以及是否可以超过 31 个字符
AltFontFile	指定当 AutoCAD 找不到源字体,并且字体映射文件中未指定替换字体时所使用的字体文件的位置
AltRoundDistance	指定换算单位的距离舍入值
AltSuppressLeadingZeros	设置是否消去换算单位标注的前导零
AltSuppressTrailingZeros	设置是否消去换算单位标注的后续零
AltSuppressZeroFeet	设置是否消去换算单位标注的零英尺
AltSuppressZeroInches	设置是否消去换算单位标注的零英寸
AltTabletMenuFile	指定与标准 AutoCAD 数字化仪菜单交换的替换菜单的路径
AltTextPrefix	指定除角度标注以外的换算单位标注的文字前缀
AltTextSuffix	指定除角度标注以外的换算单位标注的文字后缀
AltTolerancePrecision	指定换算标注标注公差值的小数位数
AltToleranceSuppressLeadingZeros	设置是否消去换算单位公差值的前导零
AltToleranceSuppressTrailingZeros	设置是否消去换算单位公差值的后续零
AltToleranceSuppressZeroFeet	设置是否消去换算单位公差值的零英尺
AltToleranceSuppressZeroInches	设置是否消去换算单位公差值的零英寸
AltUnits	设置是否显示换算单位标注
AltUnitsFormat	指定除角度标注以外的换算单位标注的单位格式
AltUnitsPrecision	指定换算单位中小数位数精度
AltUnitsScale	指定换算单位的比例因子
Angle	指定直线的角度
AngleFormat	指定角度标注的单位格式
AngleVertex	指定三点角度标注的角顶点
Annotation	指定引线的注释对象
Application	获取 Application 对象
ArcEndParam	指定圆弧的终点参数
ArcLength	指定圆弧的弧长
ArcPoint	指定圆弧上的一个点
ArcSmoothness	指定圆、圆弧和椭圆的光滑度
ArcStartParam	指定圆弧的起点参数
Area	指定构成弧、圆、椭圆、填充图案、优化多段线、多段线、面域或平面闭合样条曲线的封闭区域的面积
Arrowhead1Block	指定用作尺寸线第一端自定义箭头的块
Arrowhead1Type	指定第一个标注箭头的类型
Arrowhead2Block	指定用作尺寸线第二端自定义箭头的块
Arrowhead2Type	指定第二个标注箭头的类型
ArrowheadBlock	指定径向标注或引线上作为自定义箭头的图块
ArrowheadSize	指定标注箭头、引线箭头和钩线的尺寸

续表

属性	描述
ArrowheadType	指定半径标注箭头或引线箭头的类型
AssociativeHatch	决定图案填充是否关联
AttachmentPoint	指定 MText 对象的附着点
Author	指定图形的作者
AutoAudit	指定在渲染 DXFIN 或 DTBIN 交换命令后 AutoCAD 是否执行核查
AutomaticPlotLog	指定是否自动保存打印或发布日志
AutoSaveInterval	指定自动保存间隔的分钟数
AutoSavePath	指定当用 AutoSaveInterval 属性启用自动保存功能时所创建文件的路径
AutoSnapAperture	切换自动捕捉靶框的显示
AutoSnapApertureSize	指定自动捕捉靶框的大小
AutoSnapMagnet	切换自动捕捉磁吸
AutoSnapMarker	切换自动捕捉标记
AutoSnapMarkerColor	指定自动捕捉标记的颜色
AutoSnapMarkerSize	指定自动捕捉标记的大小
AutoSnapToolTip	切换自动捕捉工具提示的显示
AutoTrackingVecColor	指定自动追踪矢量的颜色
AutoTrackTooltip	切换自动追踪工具提示的显示
BackgroundFill	指定多行文字是否带有背景填充
Backward	指定文字是否反向
BasePoint	指定射线或构造线基点
BatchPlotProgress	获取批处理打印的当前状态或终止批处理打印
BeepOnError	指定 AutoCAD 检测到无效输入时是否音响提示
BigFontFile	指定与文字或属性相关联的大字体文件的名称
BitFlags	指定表格样式的位标志值
Block	获得与布局关联的块
Blocks	获得图形的 Blocks 集合
BlockScaling	指定块的允许比例
Blue	指定真彩色的蓝色分量
BookName	指定配色系统的名称(如果存在)
Brightness	指定光栅图像的当前亮度值
CanonicalMediaName	按名称指定图纸尺寸
Caption	获取用户在应用程序或菜单项上看到的文字
CategoryName	指定视图的分类名称
Center	指定圆弧、圆、椭圆、视图或视口的中心
CenterMarkSize	指定半径和直径标注上圆心标记的大小
CenterPlot	指定在介质上居中打印
CenterPoint	指定圆弧的圆心
CenterType	指定半径和直径标注上圆心标记的类型
Centroid	获取面域或实体的质心
Check	指定弹出菜单项的选中状态
ChordPoint	指定圆弧上的附着点

续表

属性	描述
Circumference	指定圆的周长
Clipped	指定标准视口是否被用户定义的边界替代
ClippingEnabled	启用或禁用图像的剪裁边界
Closed	决定三维多段线、优化多段线、多段线或样条曲线是否闭合
Color	指定图元或图层的颜色
ColorBookPath	设置配色系统路径
ColorIndex	指定颜色索引
ColorMethod	指定颜色方式
ColorName	指定颜色名(如果存在)
Columns	用于 MinsertBlock 对象,指定块阵列中的列数
Columns	用于表格对象,指定表格中的列数
ColumnSpacing	指定块阵列中的列间距
ColumnWidth	设置表格中所有列的统一宽度
Comment	指定数字签名附带的注释
Comments	用于 Block 对象,指定块的注释
Comments	用于 SummaryInfo 对象,指定用于查找图形的注释
ConfigFile	获取用于保存硬件设备驱动程序信息的配置文件的位置
ConfigName	指定打印机配置名
Constant	指定属性或属性参照的固定模式
ConstantWidth	指定多段线的固定宽度
ContinuousPlotLog	指定是否保存一个连续打印日志
ContourLinesPerSurface	指定每个对象表面上轮廓素线的数目
Contrast	指定光栅图像的当前对比度值
ControlPoints	指定样条曲线的当前控制点
Coordinate	指定对象中单个顶点的坐标
Coordinates	指定对象中每个顶点的坐标
Count	获取集合、词典、组或选择集中项目的数目
CreateBackup	指定是否使用备份文件
CursorSize	以屏幕大小的百分比指定十字光标的大小
CustomDictionary	指定自定义字典
CustomScale	指定视口的自定义比例
Database	获取对象所属的数据库
DecimalSeparator	指定公制标注的小数点分隔符
DefaultInternetURL	指定默认 Internet 地址
DefaultOutputDevice	指定新布局和模型空间的默认输出设备
DefaultPlotStyleForLayer	指定新图形或用早期版本的 AutoCAD 创建且未用 AutoCAD 2000 格式保存的图形中的图层 0 的默认打印样式
DefaultPlotStyleForObjects	指定新创建对象的默认打印样式名称
DefaultPlotStyleTable	指定要附着到新图形的默认打印样式表
DefaultPlotToFilePath	指定打印到文件的默认路径
Degree	指定样条曲线的次数
Delta	指定直线的增量

续表

属性	描述
DemandLoadARXApp	指定若图形包含第三方应用程序所创建的自定义对象时，是否以及何时由 AutoCAD 按需加载该应用程序
Description	指定对象的说明
Diameter	指定圆的直径
Dictionaries	获取文档的 Dictionaries 集合
DimensionLineColor	指定标注、引线或公差对象的尺寸线颜色
DimensionLineExtend	指定当绘制的是短斜线而不是箭头时，尺寸线超出尺寸界线的量
DimensionLineWeight	指定尺寸线的线宽
DimLine1Suppress	设置是否显示第一条尺寸线
DimLine2Suppress	设置是否显示第二条尺寸线
DimLineInside	设置只在尺寸线内侧绘制尺寸线
DimLineSuppress	设置是否显示径向标注的尺寸线
DimStyles	获取文档的 DimStyles 集合
Direction	指定图形三维效果的查看方向，或表格的方向矢量
DirectionVector	控制块中夹点的显示
Display	通过矢量来指定射线、公差或构造线的方向
DisplayGrips	获取 PreferencesDisplay 对象
DisplayGripsWithinBlocks	控制拉伸、移动、旋转、缩放和镜像夹点模式的选择集夹点的显示
DisplayLayoutTabs	指定是否在图形编辑器中显示模型和布局选项卡
DisplayLocked	决定视口是否处于锁定状态
DisplayOLEScale	决定当图形中插入 OLE 对象时是否显示 OLE 缩放对话框
DisplayScreenMenu	指定是否在图形窗口的右边显示屏幕菜单
DisplayScrollBars	指定是否在图形窗口的下边和右边显示滚动条
DisplaySilhouette	控制在线框模式中是否显示实体对象的曲面轮廓
DockedVisibleLines	指定命令窗口所显示的文字行数
DockStatus	指定工具栏是固定还是浮动
Document	获取对象所属的文档(图形)
Documents	获取 Documents 集合
Drafting	获取 PreferencesDrafting 对象
DrawingDirection	指定多行文字段落的的阅读方向
DriversPath	指定 AutoCAD 查找视频显示、定点设备、打印机和绘图仪所用 ADI 设备驱动程序的目录
EffectiveName	指定原始块名
Elevation	指定图案填充或多段线的当前标高
ElevationModelSpace	指定模型空间中的标高设置
ElevationPaperSpace	指定图纸空间中的标高设置
Enable	启用弹出菜单项。指定对象名
EnableStartupDialog	指定当 AutoCAD 启动时或新图形创建时是否显示开始对话框
EndAngle	指定弧或椭圆的终止角度
EndParameter	指定椭圆的终止参数
EndPoint	指定弧、直线或椭圆的终点
EndSubMenuLevel	指定子菜单中的最后一项

续表

属性	描述
EndTangent	以方向矢量指定样条曲线的终点切向
EntityColor	指定真彩色的颜色值
Explodable	指定块是否可分解
ExtensionLineColor	指定尺寸界线的颜色
ExtensionLineExtend	指定尺寸界线超出尺寸线的距离
ExtensionLineOffset	指定尺寸界线偏移起点的距离
ExtensionLineWeight	指定尺寸界线的线宽
ExtLine1EndPoint	指定第一条尺寸界线的终点
ExtLine1Point	指定第一条尺寸界线的原点
ExtLine1StartPoint	指定第一条尺寸界线的起点
ExtLine1Suppress	指定隐藏第一条尺寸界线
ExtLine2EndPoint	指定第二条尺寸条界线的终点
ExtLine2Point	指定第二条尺寸界线的原点
ExtLine2StartPoint	指定第二条尺寸界线的起点
ExtLine2Suppress	指定隐藏第二第尺寸界线
Fade	指定光栅图像的当前褪色度值
Feature	返回文件相关表中条目的特征名称
FieldLength	指定属性或属性参照的字段长度
FileName	返回文件相关表中条目的文件名
Files	获取 PreferencesFiles 对象
FileSize	返回文件相关表中条目的文件大小
FingerprintGUID	返回文件相关表中条目的指纹 GUID
Fit	确定要移动哪个元素以使文字和箭头在尺寸界线间布局合适
FitPoints	指定样条曲线的拟合点
FitTolerance	按新的公差值重新拟合样条曲线的现有点
FloatingRows	指定浮动工具栏的行数
FlowDirection	指定标题行和页眉行是否在表格的底部或顶部
Flyout	获取与弹出式工具栏项目关联的工具栏
FontFile	指定主字体文件路径和名称
FontFileMap	指定用于定义当 AutoCAD 不能找到字体时替换字体的文件位置
ForceLineInside	当文字放在尺寸界线外侧时强制在尺寸界线之间绘制尺寸线
FoundPath	返回在文件相关表中条目的替换路径
FractionFormat	设置标注和公差的分数类型
Freeze	指定图层的冻结状态
FullCRCValidation	指定是否每次对象被读入图形时都进行循环冗余码校验(CRC)
FullFileName	返回文件相关表中条目保存的完整的文件名
FullName	获取应用程序或文档的名称,包括路径
FullScreenTrackingVector	切换全屏追踪矢量的显示
GradientAngle	指定渐变填充的角度
GradientCentered	指定是否按中心渐变填充
GradientColor1	指定渐变填充的起始颜色
GradientColor2	指定渐变填充的结束颜色

续表

属性	描述
GradientName	指定渐变填充的图案名称
GraphicsWinLayoutBackgrndColor	指定图纸空间布局的背景颜色
GraphicsWinModelBackgrndColor	指定模型空间窗口的背景颜色
Green	指定真彩色的绿色分量
GridOn	指定视口栅格的状态
GripColorSelected	指定选定夹点的颜色
GripColorUnselected	指定未选中夹点的颜色
GripSize	指定夹点的大小
Groups	获取文档的 Groups 集合
Handle	获取对象的句柄
HasAttributes	指定块中是否含有属性
HasExtensionDictionary	确定对象是否有扩展字典与其关联
HasLeader	指定标注是否有引线
HasSubSelection	指定表格是否含有子选择集
HasVpAssociation	指定视图是否与图纸空间视口相关联
HatchObjectType	指定图案填充的类型为标准或渐变填充
HatchStyle	设置图案填充的样式
HeaderSuppressed	指定是否消去表格的页眉
Height	用于表格对象，指定表格的高度
HelpFilePath	指定 AutoCAD 帮助文件的位置
HelpString	指定工具栏、工具栏项目或菜单项的帮助字符串
HistoryLines	指定保留在内存中的文本窗口中的文字行数
HorizontalAlignment	指定属性、属性参照或文字的水平对齐
HorizontalTextPosition	指定标注文字的水平对齐
HorzCellMargin	指定单元格水平合并的值
HWND	获取文档窗口框架或应用程序窗口框架的窗口句柄
HyperlinkBase	指定用于所有嵌入于图形中的相对超级链接的基地址
HyperlinkDisplayCursor	切换超级链接光标和快捷菜单的显示
HyperlinkDisplayTooltip	切换超级链接工具提示的显示
Hyperlinks	获取图元的 Hyperlinks 集合
ImageFile	指定光栅图像文件的完整路径和文件名
ImageFrameHighlight	控制选择过程中光栅图像的显示
ImageHeight	指定光栅图像的高度
ImageVisibility	指定图像可见性的开或关
ImageWidth	指定光栅图像的宽度
IncrementalSavePercent	指定图形文件中所允许的闲置空间的百分比
Index	指定菜单或工具栏项目的索引
Index	指定文件相关表中的条目索引
InsertionPoint	指定公差、文字、块或形的插入点
InsUnits	指定与块一起保存的插入单位
InsUnitsFactor	指定在块单位和图形单位间的转换比例因子

续表

属性	描述
Invisible	指定属性或属性参照的不可见模式
IsCloned	判断在 CopyObjects 运算中的源对象是否被复制
IsDynamicBlock	指定块是否为动态块
IsLayout	确定给定的块是否为布局块
IsModified	指定在文件相关表中的条目是否被修改
ISOPenWidth	指定 ISO 填充图案的 ISO 笔宽
IsOwnerXlated	确定 CopyObjects 运算中的宿主对象是否被转换
IsPartial	指定标注是否用于局部圆弧
IsPeriodic	指定给定的样条曲线是否周期曲线
IsPlanar	确定给定的样条曲线是否是平面的
IsPrimary	确定 CopyObjects 运算中的源对象是否是被复制对象主集的一部分，或者它仅从属于主集中某个成员
IsQuiescent	检查 AutoCAD 是否空闲并接受 Automation 请求之外的处理
IsRational	指定给定的样条曲线是否为有理的
Issuser	指定数字证书的发布者名称
IsXRef	判断给定块是否为外部参照块
JogAngle	指定折角的角度
JogLocation	指定折角位置或拾取点
Justification	指定多线的对齐方式
Key	CopyObjects 运算中源对象的对象 ID
KeyboardAccelerator	指定 Windows 标准或 AutoCAD 传统快捷键
KeyboardPriority	控制 AutoCAD 如何响应坐标数据的输入
KeyLength	指定加密密钥长度
Keywords	指定用于查找图形的关键字
Knots	获取样条曲线的节点矢量
Label	指定菜单项显示的内容和格式
LargeButtons	指定工具栏按钮显示大图标还是小图标
LastHeight	指定所使用的上一个文字高度
LastSavedBy	指定最后修改文件的人名
Layer	指定图元的图层
LayerOn	指定图层的状态
Layers	获取文档的 Layers 集合
LayerState	指定视图的图层状态名称
Layout	指定与模型空间，图纸空间或块对象关联的布局
LayoutCreateViewport	切换是否自动为新布局创建视口
LayoutCrosshairColor	指定图纸空间布局中十字光标和文字的颜色
LayoutDisplayMargins	切换布局中页边距的显示
LayoutDisplayPaper	切换布局中图纸背景的显示
LayoutDisplayPaperShadow	切换布局中图纸背景阴影的显示
LayoutID	指定视图的布局
Layouts	获取文档的 Layouts 集合
LayoutShowPlotSetup	切换创建新布局时“打印设置”对话框的显示

续表

属性	描述
Leader1Point	指定引线 1 的原点
Leader2Point	指定引线 2 的原点
LeaderLength	指定直径或半径标注的引线长度
Left	指定工具栏的左边界
Length	获取对象的长度
LensLength	指定透视视图使用的镜头长度
Limits	指定图形界限
LinearScaleFactor	指定线性标注测量单位的全局比例因子
LineSpacingDistance	指定多行文字的行间距
LineSpacingFactor	指定多行文字对象的相对行间距比例
LineSpacingStyle	指定多行文字对象的行间距样式
Linetype	指定图元的线型
LinetypeGeneration	指定二维多段线或优化多段线的线型生成方式
Linetypes	获取文档的 Linetypes 集合
LinetypeScale	指定图元的线型比例因子
Lineweight	指定个别图元的线宽或图形的默认线宽
LineweightDisplay	指定在图形的模型空间中是否显示线宽
LoadAcadLspInAllDocuments	指定 acad.lsp 是在启动时加载还是每一图形均加载
LocaleID	获取当前 AutoCAD 进程的局部 ID
Lock	锁定或解锁图层
LockPosition	指定块中的属性或属性参照是否可相对于几何图形移动
LogFileOn	指定是否将文字窗口的内容写入日志文件
LogFilePath	指定日志文件的位置
LowerLeftCorner	获取当前活动视口的左下角
Macro	指定菜单或工具栏项目所运行的宏
MainDictionary	指定用于拼写检查的当前词典
MajorAxis	指定椭圆的长轴方向
MajorRadius	指定椭圆的长轴半径
Mask	设置用于恢复指定图层状态的屏蔽
MaxActiveViewports	指定活动视口的最大数目
MaxAutoCADWindow	指定 AutoCAD 启动时是否布满整个屏幕
MClose	指定 PolygonMesh 在 M 方向是否封闭
MDensity	指定 PolygonMesh 在 M 方向的面密度
Measurement	获取标注的测量单位
MenuBar	获取进程的 MenuBar 对象
MenuFile	指定进程的 AutoCAD 菜单或自定义文件的位置
MenuFileName	获取菜单组所在菜单或自定义文件的名称
MenuGroups	获取进程的 MenuGroups 集合
Menus	获取 PopupMenus 集合
MinimumTableHeight	指定表格的最小高度
MinimumTableWidth	指定表格的最小宽度
MinorAxis	指定椭圆的短轴方向

续表

属性	描述
MinorRadius	指定椭圆的短轴半径
Mode	指定属性定义的模式
ModelCrosshairColor	指定模型空间十字光标和文字的颜色
ModelSpace	获取文档的 ModelSpace 集合
ModelType	获取打印配置是否仅应用到模型空间或应用到所有布局
MomentOfInertia	获取实体的惯性矩
MRUNumber	指定文件菜单中显示的最近使用文件的数目
MSpace	允许从浮动的图纸空间视口中编辑模型
MVertexCount	获取 PolygonMesh 的 M 方向的顶点数
Name	指定对象的名称
NameNoMnemonic	指定没有下划线助记符号的弹出菜单的名称
NClose	指定 PolygonMesh 在 N 方向是否封闭
NDensity	指定 PolygonMesh 在 N 方向的平滑面密度
Normal	指定图元的三维法向单位矢量
NumberOfControlPoints	获取样条曲线控制点的数目
NumberOfCopies	指定要打印的份数
NumberOfFaces	获取 PolyfaceMesh 的面数
NumberOfFitPoints	获取样条曲线的拟合点的数目
NumberOfLoops	获取图案填充的边界数量
NumberOfVertices	获取 PolyfaceMesh 的顶点数
NVertexCount	获取 PolygonMesh 在 N 方向的顶点数
ObjectID	获取对象的对象 ID
ObjectName	获取对象的 AutoCAD 类名
ObjectSnapMode	指定对象捕捉模式的设置
ObjectSortByPlotting	切换图形对象为按打印顺序排序
ObjectSortByPSOutput	切换图形对象为按 PostScript 输出顺序排序
ObjectSortByRedraws	切换图形对象为按重画顺序排序
ObjectSortByRegens	切换图形对象为按重生成顺序排序
ObjectSortBySelection	切换图形对象为按对象选择排序
ObjectSortBySnap	切换图形对象为按对象捕捉排序
ObliqueAngle	指定对象的倾斜角度
OLELaunch	确定当打印 OLE 对象时是否激活源应用程序
OLEQuality	指定 OLE 对象的打印质量
OnMenuBar	确定指定的弹出菜单是否在菜单栏上
OpenSave	获取 PreferencesOpenSave 对象
Origin	以 WCS 坐标指定 UCS、块、布局或光栅图像的原点
OrthoOn	指定视口的正交模式的状态
Output	获取 PreferencesOutput 对象
OverrideCenter	指定替代中心位置或拾取点
OwnerID	获取所有者(上层)对象的对象 ID
PageSetupOverridesTemplateFile	指定替换页面设置的默认样板
PaperSpace	获取文档的 PaperSpace 集合

续表

属性	描述
PaperUnits	指定显示布局或打印配置属性的单位
Parent	获取对象的上层对象
Password	指定加密口令
Path	获取块、文档、应用程序或外部参照的路径
PatternAngle	指定填充图案角度
PatternDouble	指定用户定义图案填充是否为双向图案填充
PatternName	指定填充图案名称
PatternScale	指定填充图案比例
PatternSpace	指定用户定义填充图案的间距
PatternType	获取用于图案填充的图案类型
Perimeter	获取内部和外部面域环的总长度
PickAdd	确定对象是否要使用 Shift 键加入选择集
PickAuto	控制在选择对象提示下自动显示选择窗口
PickBoxSize	指定对象拾取框的大小
PickDrag	控制绘制选择窗口的方式
PickFirst	控制在发出命令之前(先选择后执行)还是之后选择对象
PickfirstSelectionSet	获取命令执行前已经选定了的选择集
PickGroup	确定是否在选择组合中的单个对象时将整个组合都选中
Plot	获取文档的 Plot 对象
PlotConfigurations	获取文档的 PlotConfigurations 集合
PlotHidden	指定打印期间是否隐藏对象
PlotLegacy	切换是否允许运行传统打印脚本
PlotLogFilePath	指定打印日志文件的路径
PlotOrigin	指定 UCS、块、布局或光栅图像在 WCS 坐标下的原点
PlotPolicy	确定在创建新图形时对象颜色属性是否与打印样式名称关联
PlotRotation	指定布局或打印配置的旋转角度
PlotStyleName	指定一个对象、一组对象或图层的打印样式名称
Plottable	指定图层是否为可打印
PlotType	指定布局或打印配置的类型
PlotViewportBorders	指定视口的边界是否打印
PlotViewportsFirst	指定图纸空间视口上的几何体是否首先打印
PlotWithLineweights	指定对象是按在打印文件中分配的线宽打印,还是按图形文件中的线宽打印
PlotWithPlotStyles	指定对象是按在打印样式文件中分配的配置打印,还是按图形文件中的配置打印
PolarTrackingVector	切换极轴追踪矢量的显示
PostScriptPrologFile	在 acad.psf 文件中自定义前导部分的名称
Preferences	获取 Preferences 对象
Preset	指定属性或属性参照是否为预设
PrimaryUnitsPrecision	指定标注或公差的主单位显示的小数位位数
PrincipalDirections	获取实体或面域的主方向
PrincipalMoments	获取实体或面域的主力矩属性

续表

属性	描述
PrinterConfigPath	指定打印机配置文件的位置
PrinterDescPath	指定打印机描述文件的位置
PrinterPaperSizeAlert	指定当布局配置的页面尺寸与 PC3 文件的默认设置不同时是否提醒用户
PrinterSpoolAlert	指定当由于 I/O 端口冲突造成向设备的输出必须通过系统打印机缓冲时是否警告用户
PrinterStyleSheetPath	指定打印机样式表文件的位置
PrintFile	指定用作临时打印文件名的替换名称
PrintSpoolerPath	指定后台打印文件的目录，AutoCAD 将后台打印到此位置
PrintSpoolExecutable	指定用于后台打印的应用程序
ProductOfInertia	获取实体或面域的惯性积
Profiles	获取 PreferencesProfiles 对象
PromptString	指定属性的提示字符串
PropertyName	指定属性的名称
ProviderName	指定加密提供者名称
ProviderType	指定加密提供者类型
ProxyImage	控制图形中由第三方应用程序所创建对象的显示
QNewTemplateFile	设置 QNEW 命令的默认样板文件名称
QuietErrorMode	切换打印错误报告的静默错误模式
RadiiOfGyration	获取实体或面域的旋转半径
Radius	指定圆弧或圆的半径
RadiusRatio	指定椭圆的长短轴半径比
ReadOnly	指定文档或属性是只读还是可读写
Red	指定真彩色的红色分量
ReferenceCount	指定文件相关列表中条目的参照数
RegenerateTableSuppressed	启用或禁用表格块的重新生成
RegisteredApplications	获取文档的 RegisteredApplications 集合
RemoveHiddenLines	指定在图纸空间视口上是否隐藏线被打印
RenderSmoothness	指定着色、渲染和消隐对象的平滑度
RevisionNumber	指定修订次数
Rotation	指定对象的旋转角度
RoundDistance	指定标注单位舍入值
RowHeight	指定表格中所有行的统一高度
Rows	指定块阵列中的列数
Rows	指定表格中的行数
RowSpacing	指定块阵列中的行间距
SaveAsType	指定图形的保存类型
Saved	指定文档是否有未保存的修改
SavePreviewThumbnail	指定是否与图形一起保存 BMP 预览图像
ScaleFactor	指定对象的比例因子
ScaleLineweights	指定打印布局时线宽是否与其他几何图形一起缩放

续表

属性	描述
SCMCommandMode	决定处于命令模式(有一个命令正在执行)时绘图区域中右击的功能
SCMDefaultMode	决定处于默认模式(未选中对象且无命令在执行)时绘图区域中右击的功能
SCMEditMode	决定处于编辑模式(选中一个或多个对象且无命令在执行)时绘图区域中右击的功能
SCMTimeMode	控制计时右键动作
SCMTimeValue	通过设置按压持续时间显示快捷菜单来控制计时右键动作
SecondPoint	指定射线或构造线的第二点
SegmentPerPolyline	指定每条多段线所生成的直线段的数目
Selection	获取 PreferencesSelection 对象
SelectionSets	获取文档的 SelectionSet 集合
SerialNumber	指定数字证书的序列号
ShadePlot	指定视口的着色打印模式
ShortcutMenu	确定指定的弹出菜单是否为快捷菜单
ShortCutMenuDisplay	控制在图形区域右击时是显示快捷菜单还是执行回车
Show	指定属性是否显示在用户界面中
ShowPlotStyles	指定是否在打印中应用打印样式
ShowProxyDialogBox	指定当用户打开包含自定义对象的图形时 AutoCAD 是否显示警告信息
ShowRasterImage	控制实时平移和缩放时光栅图像的显示
ShowRotation	确定是否按旋转角度值显示光栅图像
ShowWarningMessages	重置所有包含"不再显示此消息"复选框的对话框,使它们再次显示
SingleDocumentMode	确定 AutoCAD 运行单文档模式或多文档模式
SnapBasePoint	指定视口的捕捉基点
SnapOn	指定捕捉状态
SnapRotationAngle	指定视口相对于当前 UCS 的捕捉旋转角度
SolidFill	指定多线、宽线、实体、所有图案填充(包括实体填充)和宽多段线是否填充
StandardScale	指定布局、视口或打印配置的标准比例
StartAngle	指定圆弧或椭圆的起点角度
StartParameter	指定椭圆的起点参数
StartPoint	指定圆弧、直线或椭圆的起点
StatusID	获取视口的当前活动状态
StoreSQLIndex	决定是否在图形中保存 SQL 索引
StyleName	指定对象的打印样式名
StyleName	指定表格样式的名称
StyleSheet	指定布局或打印配置的样式表
Subject	指定数字证书的主题名
Subject	指定图形的主题
SubMenu	获取与子菜单关联的弹出菜单

续表

属性	描述
SummaryInfo	指定图形的属性
SupportPath	指定 AutoCAD 搜索支持文件的目录
SuppressLeadingZeros	设置是否消去标注的前导零
SuppressTrailingZeros	设置是否消去标注的后续零
SuppressZeroFeet	设置是否消去标注的零英尺
SuppressZeroInches	设置是否消去标注的零英寸
SymbolPosition	指定弧长标注符号所放的位置
System	获取 PreferencesSystem 对象
TablesReadOnly	决定是否以只读模式打开数据库表
TableStyleOverrides	指定表格样式替代
TabOrder	指定一个布局的选项卡次序
TagString	指定对象的标记字符串
Target	指定视图或者视口的目标视点
TempFileExtension	指定临时文件的扩展名
TempFilePath	指定 AutoCAD 用于保存临时文件的目录
TemplateDWGPath	指定“启动”向导所使用样板文件的路径
TempXRefPath	指定外部参照文件的位置
TextAlignmentPoint	指定文字和属性参照的对齐点
TextColor	指定标注或公差对象文字的颜色
TextEditor	指定 MTEXT 命令所使用文字编辑器的名称
TextFont	指定新文字的字体
TextFontSize	指定新文字的字号
TextFontStyle	指定新文字的字体样式
TextFrameDisplay	在仅显示文字边框和显示文字本身之间切换
TextGap	指定在打断尺寸线、放入标注文字时，标注文字与尺寸线之间的距离
TextGenerationFlag	指定属性文字生成标志
TextHeight	指定标注或公差的文字高度
TextInside	设置是否在尺寸界线之间绘制文字
TextInsideAlign	设置除坐标标注外的所有标注样式是否在尺寸界线内侧显示标注文字
TextMovement	指定文字被手动或自动移动时的位置
TextOutsideAlign	设置除坐标标注外的所有标注样式是否将标注文字放在尺寸界线外侧
TextOverride	指定标注的文字字符串
TextPosition	指定标注文字位置
TextPrecision	指定角度标注文字显示的精度小数位数
TextPrefix	指定标注的文字前缀
TextRotation	指定标注文字的旋转角度
TextString	指定图元的文字字符串
TextStyle	指定标注或公差文字的文字样式
TextStyles	获取文档的 TextStyle 集合
TextSuffix	指定标注的文字后缀

续表

属性	描述
TextureMapPath	指定 AutoCAD 搜索渲染纹理贴图的目录
TextWinBackgrndColor	指定文本窗口的背景颜色
TextWinTextColor	指定文本窗口的文字颜色
Thickness	指定二维 AutoCAD 对象按其平面向上或向下拉伸的距离
TimeServer	指定用于数字签名的时间服务器名称
TimeStamp	指定文件相关列表中条目的时间戳
Title	指定用于查找图形的标题
TitleSuppressed	指定是否消去表格的标题
ToleranceDisplay	指定标注文字的标注公差显示模式
ToleranceHeightScale	指定公差值高度相对于标注文字高度的比例因子
ToleranceJustification	指定公差值相对于标注文字的垂直对正
ToleranceLowerLimit	指定标注文字的最小公差极限尺寸
TolerancePrecision	指定主单位标注公差值的小数位数
ToleranceSuppressLeadingZeros	设置是否消去主单位公差值的前导零
ToleranceSuppressTrailingZeros	设置是否消去主单位公差值的后续零
ToleranceSuppressZeroFeet	设置是否消去主单位公差值的零英尺
ToleranceSuppressZeroInches	设置是否消去主单位公差值的零英寸
ToleranceUpperLimit	指定标注文字的最大公差极限尺寸
Toolbars	获取 Toolbars 集合
ToolPalettePath	设置工具选项板路径
Top	指定工具栏的上边界
TotalAngle	获取圆弧的总角度
TranslateIDs	指定 deepClone 或 wblockClone 运算过程中任何内含对象 ID 的转换
Transparency	决定两色图像是否透明
TrueColor	指定对象的真彩色
TrueColorImages	决定是否以真彩色显示光栅和渲染图像
TwistAngle	指定视口的扭曲角度
Type	指定 Leader、MenuGroup、PopupMenuItem、ToolbarItem、Polyline 或 PolygonMesh 对象的类型
UCSIconAtOrigin	指定 UCS 图标是否显示在原点
UCSIconOn	指定 UCS 图标是否打开
UCSPerViewport	指定 UCS 是否随视口保存
Units	指定块的本地测量单位
UnitsFormat	指定除角度外的所有尺寸标注的单位格式
UnitsType	指定用于属性的当前显示单位类型
UpperRightCorner	返回当前活动视口的右上角
UpsideDown	指定文本的方向
URL	指定超级链接对象的 URL 地址
URLDescription	指定超级链接对象的 URL 地址描述
URLNamedLocation	指定超级链接对象的已命名位置
Used	指定图层是否在使用中
UseLastPlotSettings	决定是否使用最后一次成功打印时的打印设置

续表

属性	描述
User	得到用户参数选择(PreferencesUser)对象
UseStandardScale	指定打印使用标准比例还是用户选定的比例
Utility	提到文档的实用工具(Utility)对象
Value	指定属性的当前值，或 CopyObjects 运算中新创建对象的对象 ID
VBE	获取 VBAIDE 扩展对象
Verify	指定属性或者属性参照是否设置为验证
Version	获取正在使用的 AutoCAD 版本号
VersionGUID	指定文件相关表中条目的 GUID 版本名称
VertCellMargin	指定单元格垂直合并的值
VerticalAlignment	指定属性、属性参照或文字的垂直对齐方式
VerticalTextPosition	指定标注文字相对于尺寸线的垂直位置
ViewportDefault	指定图层在新视口中是否冻结
ViewportOn	指定视口的显示状态
Viewports	获得文档中视口集合
Views	获得文档中视图集合
ViewToPlot	指定要打印视图名称
VisibilityEdge1	指定 3DFace 边 1 的可见性
VisibilityEdge2	指定 3DFace 边 2 的可见性
VisibilityEdge3	指定 3DFace 边 3 的可见性
VisibilityEdge4	指定 3DFace 边 4 的可见性
Visible	指定对象或应用程序的可见性
Volume	获取实体体积
Weights	获得样条曲线的权值矢量
Width	指定表格的宽度
Width	指定文本边界、视图、图像、工具栏或主应用程序窗口的宽度
WindowLeft	指定应用程序窗口的左边界
WindowState	指定应用程序或文档窗口的状态
WindowTitle	获取文档窗口的标题
WindowTop	指定应用程序窗口的上边界
WorkspacePath	指定数据库工作空间文件的路径
XEffectiveScaleFactor	指定块的有效 X 比例因子
XRefDatabase	给出定义了 XRef 块内容的数据库对象
XRefDemandLoad	指定外部参照对象的按需加载性能
XRefEdit	判断当前图形被其他图形引用时是否可以在位编辑
XRefFadeIntensity	控制参照对象的褪色度百分比
XRefLayerVisibility	决定外部参照图形中图层的可见性和嵌套参照路径改变是否保存
XScaleFactor	指定插入的图块或者外部参照的 X 方向缩放比例
XVector	指定用户坐标系(UCS)的 X 方向向量
YEffectiveScaleFactor	指定块的有效 Y 比例因子
YScaleFactor	指定插入的图块或者外部参照的 Y 方向缩放比例
YVector	指定用户坐标系(UCS)的 Y 方向向量
ZEffectiveScaleFactor	指定块的有效 Z 比例因子
ZScaleFactor	指定插入的图块或者外部参照的 Y 方向缩放比例

A.4 事件

事件列表如表 A.4 所示。

表 A.4 事件

事件	描述
Activate	当前文档容器激活时被触发
AppActivate	主应用程序窗口激活前触发
AppDeactivate	主应用程序窗口取消激活前触发
ARXLoaded	ObjectARX 应用程序加载时触发
ARXUnloaded	ObjectARX 应用程序卸载时触发
BeginClose	在 AutoCAD 收到关闭图形的要求后立即被触发
BeginCommand	在命令发出后、完成前立即被触发
BeginDocClose	在 AutoCAD 收到关闭图形的要求后立即被触发
BeginDoubleClick	在用户双击图形中的对象后被触发
BeginFileDrop	当文件被放到主应用程序窗口中时被触发
BeginLISP	在 AutoCAD 收到运行 LISP 表达式的要求后立即被触发
BeginModal	在模态对话框显示前被触发
BeginOpen	在 AutoCAD 收到打开现有图形的要求后立即被触发
BeginPlot	在 AutoCAD 收到打印图形的要求后立即被触发
BeginQuit	在 AutoCAD 进程结束或文档关闭前被触发
BeginRightClick	用户在图形窗口上按下鼠标右键后被触发
BeginSave	在 AutoCAD 收到保存图形的要求后立即被触发
BeginShortcutMenuCommand	用户在图形窗口上按下鼠标键后，以及在快捷菜单以命令模式之前被触发
BeginShortcutMenuDefault	用户在图形窗口按下鼠标右键后，以及在快捷菜单以默认模式之前被触发
BeginShortcutMenuEdit	用户在图形窗口上按下鼠标右键后，以及在快捷菜单以编辑模式出现之前被触发
BeginShortcutMenuGrip	用户在图形窗口上按下鼠标右键后，以及在快捷菜单以夹点模式出现之前被触发
BeginShortcutMenuGrip	用户在图形窗口上按下鼠标右键后，以及在快捷菜单以对象捕捉模式出现之前被触发
Deactivate	当图形窗口释放时被触发
EndCommand	在命令执行完成后立即被触发
EndLISP	在 LISP 表达式执行完成时被触发
EndModal	在模态对话框关闭后被触发
EndOpen	在 AutoCAD 完成打开现有图形后立即被触发
EndPlot	在文档已经发送到打印机后被触发
EndSave	当 AutoCAD 完成保存图形时被触发
EndShortcutMenu	在快捷菜单出现后被触发
LayoutSwitched	用户切换到不同的布局后触发

续表

事件	描述
LISPCancelled	LISP 表达式求值被取消时触发
Modified	图形中的对象和集合被修改时触发
NewDrawing	新图形创建前触发
ObjectAdded	当对象添加到图形中时被触发
ObjectErased	当对象从图形中被删除时被触发
ObjectModified	当图形中的对象被修改后被触发
SelectionChanged	当前选择优先的选择集改变时触发
SysVarChanged	系统变量值改变时触发
WindowChanged	当应用程序或文档窗口被修改时被触发
WindowMovedOrResized	当应用程序或绘图窗口被移动或改变大小时触发该事件